Optimizing the Access to Read-Only Data in Grid Computing

Alek Opitz, Hartmut Koenig

Computer Science Department
Brandenburg University of Technology Cottbus
Germany
{ao,koenig}@informatik.tu-cottbus.de

Abstract. A fundamental problem of grid computing is the communication overhead. One reason of this overhead is the access to remotely stored data. Caching read-only data is a possible alleviation of the problem. In case of grid computing caching can be optimized by using allocation schemes considering the contents of the caches. Possible ways to achieve such an allocation in a grid are the topic of this paper. The paper proposes to use allocation schemes preferring resources with the required data in their caches. In doing so the hit rate of the caches will be increased and as a consequence the average response time of the jobs and the network load will be reduced. Two new possible allocation approaches are discussed and compared with classical allocation schemes. The performance and the costs of the schemes (when applied to large grids) are evaluated using a simulation environment.

1 Introduction

In recent years strong efforts have been made to use idle computing resources distributed over the Internet. The idea of grid computing was born. It aims at giving users with compute-intensive applications the possibility to submit their jobs to a grid that provides the required resources. A fundamental problem of this approach is the access to the data. The communication overhead makes a lot of jobs unsuitable for grid computing. Therefore it is a quite obvious goal to reduce this communication overhead.

In this paper we focus on optimizing the access to read-only data.[1] Such immutable data play an important role in grid applications ([11], [1]). A typical example is virtual screening ([32]), which is used in the development of drugs for medical treatments. Virtual screening analyzes chemical compounds described in databases. These databases are not specific for single applications and describe hundreds of thousands of compounds. Those compounds that do not show any activity against the target receptor (this is the majority) are excluded. Subsequent drug design phases only consider the remaining compounds. Because the analysis is a computationally complex task, it is desirable to analyze the compounds in parallel, which makes grid computing very reasonable. Other grid applications which access read-only data are, for example, the rendering of animation movies, where the program code and the model data must be accessed ([28]), and applications accessing the protein data base ([5], [30], [3]).

[1] Note that in grid computing the program code is also a kind of read-only data that has to be transferred to the executing resources.

Since data often do not exclusively belong to a certain application, it is possible that different jobs access the same data, e.g. the same databases or the same program libraries. To accelerate the access to the data caches might be useful. Caches exploit the fact that many files are accessed multiple times. They store the recently used files in the hope that some of these are required soon again. Usually it cannot be predicted, whether the data will really be reused. There is only a limited probability. However, in the case of grid computing this probability can be increased by preferably allocating the jobs to those resources that already have at least some of the needed data. Possible ways to achieve such an optimized allocation in a grid are the topic of this paper. We especially pay attention to the costs of the allocation when considering large numbers of resources.

The paper is organized as follows. In section 2 we discuss two possible allocation schemes for this problem. Section 3 describes the simulation environment used for evaluating the performance of the two schemes. The results of this evaluation are summarized in Section 4. In Section 5 we give an overview on related work. The final remarks give an outlook on future work.

2 Optimized Allocation According to Cache Contents

In the following discussion we consider a grid with resources provided, for example, by enterprises, universities or individuals. We further assume that there is a broker between the users and the resources which is responsible for allocating appropriate resources to the jobs. After allocation the jobs are executed and the results are finally returned to the users. As we focus in this paper on optimizing the access to read-only data stored elsewhere in the Internet and because we want to keep the model as simple as possible, only serial jobs are considered. This is, of course, a simplification, but as discussed in Section 5, allocation mechanisms for parallel jobs concentrate on the parallel allocation of machines. This type of optimization is largely orthogonal to the optimizations discussed in this paper. The scenario is shown in Fig. 1.[1]

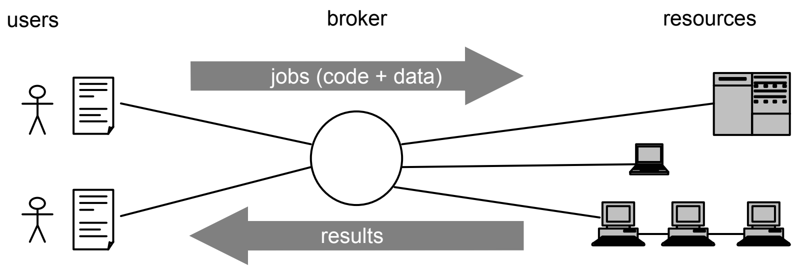

Fig. 1: Assumed scenario

In order to allocate resources that have the needed data locally available, we use directory servers. These servers store information about the locations of the existent replicates. According to the terminology used in [11] the servers are called *RLIs*

[1] Note that the broker is not necessarily involved in the transport of the data. It is possible that the resources load the data directly from the sources.

(*replica location indices*). They receive their information from *local replica catalogs* (LRCs). An LRC belongs to a single place and contains information about the data that are locally available to the resources at this place.

Since allocation schemes for grid computing are considered, attention has to be paid to the scalability problem, i.e. the solution must also work efficiently for large numbers of resources and jobs. For that reason, an infrastructure with several RLIs seems more appropriate for managing replicates than a single directory server. Obviously there are (at least) two possibilities for portioning the replicate management data. The first one is to partition the resources and to make each RLI responsible for such a partition. This possibility is discussed in Section 2.1. An alternative approach is to partition the data objects and to make each RLI responsible for one partition. This approach is considered in Section 2.2.

2.1 RLIs with Distinct Sets of Resources

The basic idea of this approach is that the individual RLIs only store information about distinct subsets of LRCs. Hence a single RLI can only return resources from a subset of the available resources. In order to avoid this, a hierarchical arrangement of the RLIs is proposed (see Fig. 2). The search for a resource with a certain data item starts at the top level RLI. This RLI (and any else) contains only condensed information about the locations of the data items, i.e. an RLI knows for each child only the set of data items that are available at any resource being descendant of this child node. Thus it is even possible to search for a resource with several needed data items instead of searching for a resource with a single data item. Obviously in this case the hierarchical approach does not guarantee to find the optimal resource, i.e. the resource with the most of the desired data items. On the other hand, the space requirements per server are reduced and the allocation process is made faster.

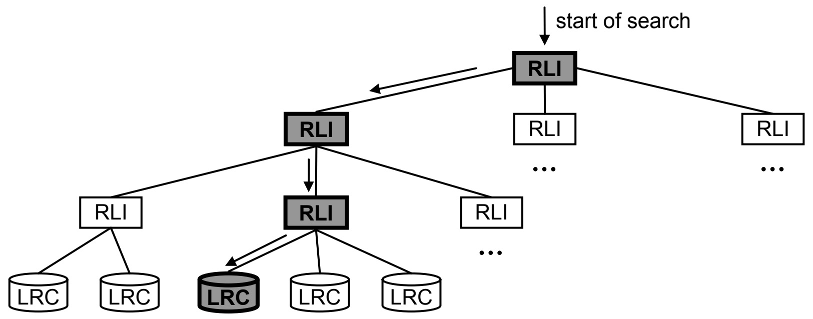

Fig. 2: Hierarchical selection of resources

Using Bloom Filters. The described hierarchical structure reduces the space requirements on the individual servers. An additional reduction seems possible by using a lossy compression of the information about the replicates. For this purpose, the use of Bloom filters was proposed ([7], [11]). Bloom filters are able to store information about the availability of data objects in an efficient way. This is achieved by applying several hash functions to a single data object. The hash functions are usually independent, but they all map onto the same domain, e.g. the range 1...*l*., i.e. onto a bit string of length *l*. When new data objects are stored at a resource, the bits calculated by the hash functions are set to 1.[1] To test the availability of a certain data item the corresponding bit positions have to be checked. Obviously this can lead to the erroneous assumptions about the availability of data items, since the same bits can be set by other items, too.

Several Top Level RLIs. Since the top level node receives all the queries, the model depicted in Fig. 2 is neither scalable nor reliable. This problem can be alleviated by introducing some additional access points. Preferably each resource should be reachable from each access point. Several topologies are conceivable, especially topologies of space division switches ([29]). For our approach, we use a simple model which is depicted in Fig. 3. Each resource belongs to exactly one of several distinct trees. The roots of these trees notify all the access points about the available data (and possibly further information). The result is a topology in which each resource can be reached from each access point on exactly one path.

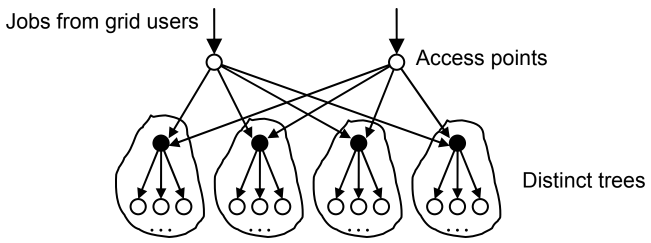

Fig. 3: Model with several access points

Using Additional Information. Finally the question arises, whether the allocation of resources should only depend on the required data objects. Our simulations showed that this is inefficient, because knowledge about the current load of the resources is extremely important. Therefore a hierarchical load balancing was added, i.e. the resources notify the RLIs not only about the available data items but also about their current load. Thus an RLI can select the appropriate child taking both the load and the available data items into account.

Costs of the Approach. In order to assess the benefit of the approach, the additional overhead of the proposed allocation scheme has to be compared with its advantages. Possible advantages are a reduced average response time, a reduced network load, and a reduced load at the data sources. The additional costs can be divided in:

(1) costs for the notification of the load and available data items at the resources
   (1a) number of bytes sent,
   (1b) costs at the resources (time busy with sending notifications),
   (1c) costs at the directories (time busy with receiving notifications)
(2) costs for selecting suitable resources
   (2a) number of bytes sent,

[1] To allow the removal of data items, a counter is needed for each bit position. These counters are only required at in the LRCs, not in the RLIs.

(2b) costs at the directories for selecting the resources.

We now discuss the individual parts of these costs. The simulation results are presented in Section 4.

Costs for Notifying the Directory. First the messages sent from the resources to the directory servers are considered. The indication of available data is mapped into Bloom tables with *n* bits. The load information is very small, e.g. 4 bytes. Both data are sent from the children nodes to their fathers. An overhead of *h* bytes is added to each message. In our simulations we used a value of $h = 50$ bytes derived from the size of the IP-header and additional overheads for other protocols. Different values of *h* did not greatly influence the results.

The hosts sending and receiving these messages have only a small computational overhead for handling these messages. Therefore the busy times are approximated by the serialization delays, as it can be assumed that NIC and CPU work in parallel.

Costs for Selecting a Resource. To select suitable resources the RLIs must be queried. A query contains a characterization of the set of the needed read-only data plus an overhead of *h* bytes. As the directory servers store only the Bloom tables it is sensible to send only the hash values of the data objects. The necessary size of the hash values can be derived from the size of the used Bloom tables. Such a query is sent to an access point and from there down through the hierarchy. Finally the answer containing the selected resource is returned to the source of the query.

Each queried RLI has to select the most appropriate child. It has to compare the current load and the Bloom tables of the children. As only the best child node has to be identified, a linear search through the children is sufficient. Usually there are only few children, so the selection of the most appropriate child is not very complex. Therefore the serialization (and deserialization) delays of the messages are equated with the aggregated costs at the involved RLIs.

2.2 RLIs with Distinct Sets of Data Items

We now consider the second approach in which the servers contain information about distinct sets of data items. The structure is depicted in Fig. 4. In this solution the amount of data per RLI can be kept constant by increasing the number of RLIs proportionally to the number of replicates. Of course there must be an efficient mechanism for identifying the RLI which contains the information about a certain data item. This can be done by means of hash functions, in case of varying sets of RLIs consistent hash functions ([24]).

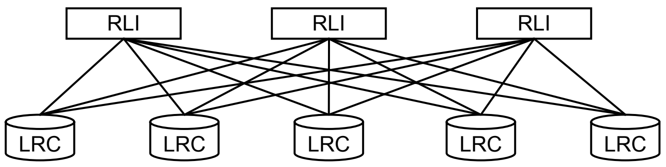

Fig. 4: Partitioning of the data sets

Selecting Suitable Resources. A problem of the depicted scenario is that a job might require several data items. However, a single RLI knows only the locations of data items from a limited subset. That means, in order to find the optimal resource for a job with multiple needed data objects it were necessary to query several RLIs for resources with the needed items. Finally the resource lists returned by the RLIs had to be unified. Sending and unifying these lists is potentially expensive. Therefore in this paper we do not follow this approach. Instead we use a mechanism that searches for a resource with the most important (i.e. biggest) of the needed data items. We regard this approach as a first step, which delivers a lower bound for the effectiveness of the method. An examination of possibilities considering several data items is planned for the future.

Using Additional Information. For the same reasons as in Section 2.1, load balancing is added. A load server is introduced that informs about the load of the available resources. The resources have to update these data when getting idle or loaded. The combined approach actually searches for resources that are unloaded and have the most important data item locally available. If no such resource can be found, any unloaded resource will be used.

Costs of the Approach. The costs of this approach are analyzed analogously to Section 2.1.

Costs for Notifying the Information Servers. The information about available data items is sent from the LRCs to the RLIs. To identify the data items URLs with an average length of $u = 100$ bytes are used. Alternatively hash values could be used what is not considered in this paper. The load information messages are sent to the load servers. It is sufficient to update the server only when the load changes. As in Section 2.1 an overhead of h bytes per message is added and the busy times of the involved hosts are approximated by the serialization delays.

Costs of Selecting Resources. To select a suitable resource a request is sent to the RLI responsible for the most important (largest) data item. The request contains the item's identifier which is assumed to be a URL with an average length of u bytes. The RLI finds the resources with the required item and sends the list to the load information service where one of the resources is selected. The response with the selected resource is returned to the requesting host. For each message, an overhead of h bytes is added. Analogously to the other estimations the time for processing the messages is approximated by the (de)serialization delays.

Optimization. A simple optimization of this allocation method is to ignore small data items. For the simulations described below a limit of 1 MB was assumed, i.e. resources with a size below 1 MB are not indicated to RLIs. Consequently, if the largest read-only data item needed by a job is smaller than 1 MB, any unloaded resource is selected.

3 Simulation

To assess the benefit of the proposed mechanisms we performed extensive simulations. The underlying model is explained in this section.

3.1 Grid Model

The environment developed for these simulations was kept as simple as possible. Thus it was possible to simulate even large grids on desktop PCs. The model consists of a configurable number of computational resources which are modeled as single-processor machines. As only serial jobs are considered this should be no problem. In the standard configuration 4096 resources were simulated. It was assumed that they have all the same properties, e.g. the same speed. Failures in the grid are not considered. Each machine is connected to the Internet and has its own cache for storing data. A cache stores each needed (read-only) data object. In case of shortness of space the least recently used objects will be removed from the cache. As at the beginning of the simulation the caches are empty, we started the measurements only after simulating the grid for a certain amount of time.

Usually an Internet connection is shared by machines of the same organization. Thus actually a single connection (with a relatively high data rate) for a set of machines should be modeled. For simplicity reasons and thus enabling simulations of large grids such shared connections were not modeled. Instead a separate link with a lower bandwidth was assumed for each machine. This model rather corresponds to a grid with privately used PCs than to a grid using the PCs of enterprises. The consequences of this simplification still has to be analyzed what is planned for the future.

Besides the resources there is the infrastructure for distributing the jobs. For simplicity, the machines of this infrastructure are connected to the Internet with the same low bandwidth as the resources. This might underestimate the available data rate, but actually this data rate is mainly used for approximating the load of the machines of the infrastructure (see Section 2.1). An increased network bandwidth does not increase the computational speed of the machines.

3.2 Workload Model

Besides modeling the resources it is at least equally important to model the workload properly. For this, a special library for generating jobs was developed. The parameters of a job are the pure execution time and additionally the required read-only data items. Each item is characterized by the pair (identifier;size). It is assumed that the needed data items are loaded completely into the cache of the executing resource.

The information for modeling the workload has been taken from several papers about workloads. The papers belong to the areas of grid computing, parallel machines and web caching. Because it is not sure that the found properties taken from other research really applies to the grid computing as discussed, quite a few variations in the workload have been tested for validating the results (see Section 4).

Inter-Arrival Times. According to [12] and [23] the distribution of inter-arrival times can be modeled by a hyper-exponential distribution, i.e.:

$$F(x) = p_1 \cdot (1 - e^{-\lambda_1 x}) + (1 - p_1) \cdot (1 - e^{-\lambda_2 x})$$

Variations due to daytime as reported, for example, in [9] are ignored. In the simulation the parameter values $\lambda_1 = 0.000204$, $\lambda_2 = 0.0038$ and $p_1 = 3.46\,\%$ were used ([23]). For scaling the load of the grid the parameters, λ_1 and λ_2 are multiplied by a factor *f*. Thus the mean value μ of the time between two successive job submissions is reduced by the factor *f*, whereas the coefficient of variation (*CV*) stays constant.

Runtimes. Different models exist for the distribution of job runtimes. [23] proposes a hyper-exponential distribution, [14] and [20] suggest a log-uniform distribution, [12] and [16] use a Weibull distribution. Fortunately, the models are very similar. In our simulation a log-uniform distribution was used. Additional tests with the Weibull distribution were made. The distribution function of a log-uniform distribution can be expressed by the following formula:

$$F(x) = a + b \cdot \ln x$$

The values of *a* and *b* are very similar in [14] und [20], for the simulation the values $a = -0.24$ and $b = 0.111$ were used.

Popularity of Data Items. For the files of the WWW, it has been shown that the popularity of the individual files follows a Zipf-like distribution ([8], [27], [4]):

$$P(i) = \frac{\Omega}{i^\gamma} \quad \text{with} \quad 0 < \gamma \le 1 \quad \text{and} \quad \Omega = \left(\sum_{i=1}^{N} \frac{1}{i^\gamma} \right)^{-1}$$

According to the literature the value of parameter γ is somewhere between 0.6 und 0.9, in our simulations $\gamma = 0.7$ was assumed. Besides parameter γ the number of different files is important. For our simulations we selected this number in such a way that the used data items were accessed on average 3.6 times during the simulation which is in line with values reported in literature ([8], [27], [10], [22]).

Because it is unclear, whether this Zipf-like popularity distribution can be applied to files in a grid environment, additional tests with other parameter values and with a uniform distribution were carried out.

Needed Read-Only Files. To simulate the delays to access read-only files their sizes must be modeled. The distribution of files sizes is usually not uniform. Instead [15] proposes a log-uniform distribution:

$$F(x) = \Phi\left(\frac{\ln x - \mu'}{\sigma'} \right) \quad \text{with} \quad \sigma' > 0$$

Following the studies presented in [31] the values $\sigma' = 0.01826$ and $\mu' = 8.699$ were chosen for the standard configuration. Besides the sizes of single files the number of files per job is important. As no empirical data was found for this number, the simulations were conducted with very different ranges of this number. In the standard configuration a uniform distribution over the interval [1;5] was chosen.

4 Simulation Results

The results of our simulations are presented in Table 1. We simulated five allocation schemes which are explained below. The second column shows the average load of the grid resources. Further, the average response time (*ART*) was measured. This is the time between the submission of a job and the return of the results. The overhead of a job is the time waiting in a job queue plus the time for loading the read-only data from a remote location. It also includes the time for notifying the used allocator. The overhead and the ART are given in seconds. The column “kbps/resource” indicates the network load. For conceivability reasons, the aggregated network load is divided by the number of resources. The rightmost column shows the demands on the infrastructure. It gives the number of needed machines and is determined by the total busy time divided by the whole simulated time. The demands are only estimates and are quite low for the simulated number of resources and for the used average job length. But increasing the number of grid resources and decreasing the average job length increases the demands. Thus the column gives an indication of the relation among the different allocation schemes.

Allocation scheme	**Load**	**ART**	**Overhead**	**kbps/resource**	**Nodes**
Random	69.7 %	49037	41134	12.1	0.00000
Round Robin	69.7 %	36415	28495	12.1	0.00000
Load Balancing	69.5 %	8053	137	12.0	0.00074
Resource partitioning (see sect. 2.1)	69.5 %	8052	137	11.9	0.02005
Data partitioning (see sect. 2.2)	69.3 %	8026	111	9.6	0.00410

Table 1: Results of the allocation schemes

The first three allocation schemes of Table 1 were introduced for comparison. The first scheme selects the resources randomly, the second one in a round robin fashion. The third allocation scheme named “Load Balancing” uses a load directory as mentioned in section 2.2 to select unloaded resources. The results clearly show the benefit of taking the resource load into account. For that reason, we use the “Load Balancing”-allocator as reference for our two allocation schemes which are shown in the last two rows.

As it can be seen from the table, the approach with RLIs responsible for distinct partitions of the resources (see section 2.1) cannot achieve the desired optimization (in relation to the reference allocation scheme). The main reason is the hierarchical selection of resources, which leads to unfavorable decisions in the process of resource allocation. In contrast the second approach (with RLIs responsible for distinct partitions of the data items, see section 2.2) indeed helps to significantly reduce the overhead and network load.

Due to limited space we present in Table 1 only the results of the standard configuration (as described in Section 3). However, simulations with various parameters showed that the relation between the five allocation schemes is largely independent of the exact values of most parameters, as, for example, the overhead per message or the size of the grid. The most significant parameters are the network bandwidth and the sizes of the data items. That means the ratio between the overheads of the individual approaches remains largely constant. As the reduction of the ART directly relates to the reduction of the overhead, it is clear that the relative reduction of the ART depends mainly on the ratio between the overhead and the ART. If this ratio is increased, the relative reduction is also increased. Correspondingly, if the ratio is decreased, the relative reduction is decreased, too. It is not quite clear, whether the ratio of our standard configuration is realistic in respect of this point. Therefore further investigations are needed.

5 Related Work

To the best of our knowledge no papers exist on resource allocation schemes as proposed in this paper. However there are other proposals with at least partially the same goal, i.e. optimizing the access to data or optimizing the resource allocation. Hence in this section a brief overview of these proposals is given. First we discuss proposals related to caching. Thereafter efficient methods for accessing remote data are reviewed and then we take a look at allocation schemes for parallel systems. Finally we discuss the applicability of methods used in the area of distributed databases.

In the WWW caching is used extensively to optimize the access to read-only data. Karger et al., for example, discuss the arrangement of the Web caches [24]. Instead of using a single hierarchy they propose to use individual hierarchies for different data objects. The trees have to be derived from the data identifiers by using hash functions. Because the set of participating caches is variable, a consistent hashing is used for reducing the costs of adding or removing a cache. Caching may also be applied to grid computing. In [2] a data grid is described that enables to access the same data from all over the world. To achieve a better efficiency the data are cached. GASS [6] is a data movement and access service for wide area computing systems which is optimized for grid applications. It exploits the fact that strong consistency among replicates is usually not necessary. Consequently the service uses caching for both reading and writing of the data.

Caching can increase the probability that a certain data item is locally available. If it is not locally available, it has to be retrieved from a suitable source. To do this in an efficient manner, two things are required [1]: an appropriate protocol for the transmission of the data and an efficient management of the copies (replicates). For the latter, a directory can be used containing for each data object the locations where it is stored. As argued in [1] users prefer working with groups of files instead with individual files. Therefore directories should contain the locations of data collections instead of locations of single files. Chervenak et al. also propose the usage of directory servers [11]. The paper describes a framework for the construction of services for localizing replicates. It is rather a classification of possibilities than a tangible concept for such a service.

Besides work focussing on data access there are a lot of papers about scheduling and resource allocation. A survey of such methods for parallel machines is given in [19]. The methods concentrate on the allocation of processors for jobs that need multiple processors at the same time. These methods are mainly orthogonal to the idea followed up in this paper. The same applies to [33] which proposes an allocation scheme that groups the processors into pools with fast connections inside the pools. Thus a parallel job should be preferably allocated to resources from a single pool. Fewer papers exist on the allocation of machines in the context of grid computing. In [21] it is emphasized that centralized methods are not suitable and some mechanisms are compared. An economical approach has been proposed in [17]. The individual jobs have utility values, the machines have machine values (i.e. costs) and the jobs are assigned to machines in an auction-like manner. [20] shows that due to the separation of local and global job queues an efficient global allocator needs regular notifications about the current state of the resources.

In the context of distributed databases the problem of finding suitable places for executing jobs arises, too. The approaches usually focus on the execution of the parts of a request at the place of the data – as far as this is possible. Only the unification is done on a coordinator ([13]). Such an approach is much more difficult in case of grid computing. At first general purpose programs cannot be decomposed as simple as SQL queries. SQL queries have a certain structure that delimits the computational power, but greatly alleviates the analysis and decomposition. Furthermore the most important idea of grid computing is the utilization of otherwise idle resources. However a location having the data locally available does not necessarily possess the needed computational resources.

6 Conclusions and Future Work

The paper discussed a possibility for optimizing the access to read-only data in case of grid computing. It presented the idea of increasing the cache hit rate by using allocation schemes regarding the contents of the caches. In order to proof the usefulness of the idea two possible allocation schemes were presented and evaluated by simulation. The calculation made by the developed simulation environment included also the costs of the allocation scheme – something that is hardly done in other work on this topic. It was shown that one of the two allocation schemes (the one with RLIs for partitions of data items, see section 2.2) indeed helps increasing the hit rate and thus reduces the overhead as well as the network load. This achievement is especially remarkable when considering that the simulated allocation scheme has not been optimized, yet. For example, the demands to the infrastructure might be reduced by using hash values instead of URLs for the data items. Furthermore, considering multiple data items instead of only the largest one could improve the effectiveness of the allocation scheme. The integration of these optimizations is planned for future work.

Even though we tried to follow the reality by simulating the execution of the jobs, it is desirable to verify the results in more realistic simulations. Especially failures in the grid and the heterogeneity should be considered what has not been done in this paper. As a starting point the results from [25], [26], and [18] could be used which discuss the generation of realistic grids.

References

1. Bill Allcock, Joe Bester, John Bresnahan, …: „Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing“ , IEEE Mass Storage Conference, 2001,
2. „Avaki Data Grid 3.0 Conceptual Overview“, Avaki Corporation December 2002, www.avaki.com
3. Kim Baldridge, Philip E. Bourne: „The new biology and the Grid", "Grid Computing - Making the Global Infrastructure a Reality", edited by F. Berman, A. Hey, G. Fox, 2003 John Wiley & Sons
4. Paul Barford, Azer Bestavros, Adam Bradley, Mark Crovella: „Changes in Web Client Access Patterns. Characteristics and Caching Implications", World Wide Web, Volume 2, Issue 1-2, 1999
5. Helen M. Berman, David S. Goodsell, Philip E. Bourne: „Protein Structures: From Famine to Feast", AMERICAN SCIENTIST (July-August 2002), http://www.sdsc.edu/pb/papers/amer_sci.pdf
6. Joseph Bester, Ian Foster, Carl Kesselman, Jean Tedesco, Steven Tuecke: „GASS: A Data Movement and Access Service for Wide Area Computing Systems", Sixth Workshop on I/O in Parallel and Distributed

Systems, May 5, 1999

7. Burton H. Bloom: „Space/Time Trade-offs in Hash Coding with Allowable Errors“, Communications of the ACM, Volume 13, Number 7, July, 1970, pp. 422-426
8. Lee Breslau, Pei Cao, Li Fan, Graham Phillips, Scott Shenker: „Web Caching and Zipf-like Distributions: Evidence and Implications“, IEEE Infocom '99, pages 126-134, New York, NY, March, 1999
9. Maria Calzarossa, Giuseppe Serazzi: „A Characterization of the Variation in Time of Workload Arrival Patterns“, IEEE Transactions on Computers, 34(2): 156-162, 1985
10. Ludmila Cherkasova, Magnus Karlsson: „Dynamics and Evolution of Web Sites: Analysis, Metrics and Design Issue“, Sixth IEEE Symposium on Computers and Communications (ISCC'01), p.64, July 03-05, 2001
11. Ann Chervenak, Ewa Deelman, Ian Foster, …: „Giggle: A Framework for Constructing Scalable Replica Location Services“, IEEE Supercomputing 2002
12. Su-Hui Chiang, Mary K. Vernon: „Characteristics of a Large Shared Memory Production Workload”, 7th Workshop on Job Scheduling Strategies for Parallel Processing, Cambridge, MA, June 2001, Lecture Notes in Computer Science, Vol. 2221, Springer-Verlag
13. Peter Dadam: „Verteilte Datenbanken und Client-/Server-Systeme. Grundlagen, Konzepte und Realisierungsformen”, Springer-Verlag, Berlin Heidelberg, 1996
14. Allen B. Downey, Dror G. Feitelson: „The Elusive Goal of Workload Characterization“, Performance Evaluation Review 26(4), pp. 14-29, March 1999, http://www.cs.huji.ac.il/~feit/pub.html
15. Allen B. Downey: „The structural cause of file size distributions“, Technical Report CSD-RT25-2000, Wellesley College, 2000, http://allendowney.com/research/filesize/
16. Darin England, Jon B. Weissman: „Costs and Benefits of Load Sharing in the Computational Grid“, Workshop on Job Scheduling Strategies for Parallel Processing with Sigmetrics 2004
17. Carsten Ernemann, Volker Hamscher, Ramin Yahyapour: „Economic Scheduling in Grid Computing“, from: D.G. Feitelson, L. Rudolph, U. Schwiegelshohn (Eds.): „Job Scheduling Strategies for Parallel Processing“, 8th International Workshop, JSSPP 2002 Edinburgh, Scotland, UK, July 24, 2002, LNCS 2537, Springer-Verlag
18. Michalis Faloutsos, Petros Faloutsos, Christos Faloutsos: „On Power-Law Relationships of the Internet Topology“, ACM SIGCOMM, Cambridge, MA, September 1999
19. Dror G. Feitelson, Larry Rudolph: „Job Scheduling in Multiprogrammed Parallel Systems. Condensed Version”, Research Report RC 19790 (87657), IBM T. J. Watson Research Center, Oct 1994, Revised version from August 1997, http://www.cs.huji.ac.il/%7Efeit/pub.html
20. Jörn Gehring, Thomas Preiß: „Scheduling a Metacomputer With Uncooperative Sub-schedulers“, from: „Job Scheduling Strategies for Parallel Processing”, Springer Verlag, Editors: Dror G. Feitelson and Larry Rudolph, pages 179-201, 1999
21. Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, Ramin Yahyapour: „Evaluation of Job-Scheduling Strategies for Grid Computing“, 1st IEEE/ACM International Workshop on Grid Compuing, Springer Verlag, LNCS 1971, Bangalore, India, December 17, 2000
22. Adriana Iamnitchi, Matei Ripeanu: „Myth and Reality: Usage Behavior in a Large Data-Intensive Physics Project”, SC2002, November 11-16, 2002, Baltimore, Maryland (poster), GriPhyN TR-2003-4
23. Joefon Jann, Pratap Pattnaik, Hubertus Franke, …: „Modeling of Workload in MPPs”, from: "Job Scheduling Strategies for Parallel Processing", Springer Verlag, editors: Dror G. Feitelson and Larry Rudolph, pp. 95-116, 1997
24. David Karger, Eric Lehman, Tom Leighton, …: „Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web”, Symposium on Theory of Computing, 1997
25. Yang-Suk Kee, Henri Casanova, Andrew Chien: „Realistic Modeling and Synthesis of Resources for Computational Grids”, ACM Conference on High Performance Computing and Networking, SC2004, Pittsburgh , Pennsylvania, November 2004
26. Dong Lu, Peter A. Dinda: „Synthesizing Realistic Computational Grids”, ACM/IEEE Supercomputing 2003 (SC 2003), November, 2003, Phoenix
27. Norifumi Nishikawa, Takafumi Hosokawa, …: „Memory-based architectur for distributed WWW caching proxy", 7th International Conference on World Wide Web, Brisbane, Australia, 1998
28. Pixar Animation Studios: „How We Make A Movie. Pixar's Animation Process“, http://www.pixar.com/howwedoit/index.html, Download: 7.6.2004
29. Martin de Prycker: „Asynchronous Transfer Mode“, Prentice Hall, 1996
30. Volker Strumpen: „Volunteer Computing”, Software - Practice and Experience, Vol. 25(3), 291–304, März 1995
31. Feng Wang, Qin Xin, Bo Hong, Scott A. Brandt, Ethan L. Miller, Darell D. E. Long, Tyce T. McLarty: „Large-scale virtual screening for discovering leads in the postgenomic era“, 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies, pages 139–152, College Park, MD, April 2004
32. B. Waszkowycz, T. D. J. Perkins, R. A. Sykes, J. Li: „Large-scale virtual screening for discovering leads in the postgenomic era“, IBM Systems Journal, Volume 40, Number 2, 2001, ("Deep computing for the life sciences"), http://www.research.ibm.com/journal/sj/402/waszkowycz.html
33. Songnian Zhou, Timothy Brecht: „Processor Pool-Based Scheduling for Large-Scale NUMA Multiprocessors”, 1991 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, San Diego, California, USA, 21st-24th May 1991