A Data Layout Strategy for Parallel Web Servers*

Jörg Jensch, Reinhard Lüling, and Norbert Sensen

Department of Mathematics and Computer Science
University of Paderborn, Germany
{jaglay, rl, sensen}@uni-paderborn.de

Abstract. In this paper a new mechanism for mapping data items onto the storage devices of a parallel web server is presented. The method is based on careful observation of the effects that limit the performance of parallel web servers, and by studying the access patterns for these servers.
On the basis of these observations, a graph theoretic concept is developed, and partitioning algorithms are used to allocate the data items. The resulting strategy is investigated and compared to other methods using experiments based on typical access patterns from web servers that are in daily use.

1 Introduction

Because of the exponential growth in terms of number of users of the World Wide Web, the traffic on popular web sites increases dramatically. To resolve capacity requirements on the server systems hosting popular web sites, one way to strengthen the capability of a server system is to use parallel server systems that contain a number of disks attached to different processors which are connected by a single bus or a scalable communication network. The same solution can be applied for very large web sites which cannot be stored on one disk but use a number of disks connected to a single processor system.

The problem that shows up if a number of disks are used to store the overall amount of data is to balance the number of requests issued to the disks as evenly as possible in order to achieve the largest overall throughput from the disks. The requests that have to be served by one disk are strictly dependent on the data layout of the server system, i.e. the mapping of the files stored on the server onto the different storage subsystems (disks). The investigation of a new concept for the data layout of parallel web servers is the focal point of this paper.

The problem is of major relevance for the performance of web servers and can be assumed to become even more important if the current technological trends concerning the bandwidth offered by storage devices, processors and wide area

* This work was partly supported by the MWF Project "Die Virtuelle Wissensfabrik", the EU project SICMA, and the DFG Sonderforschungsbereich 1511 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen".

communication networks are regarded: Whereas the performance of processors and external communication networks (ATM, Gigabit Ethernet) has been dramatically increased over the last years, the read/write performance of storage devices shows only little increase over the last years. Thus, the performance of a server system can only be increased by the use of a larger number of disks that delivers data elements in parallel to the external communication network.

Different approaches can be found in the literature in order to increase the performance of web servers: NCSA [7] and SWEB [4] have built a multi-workstation HTTP server based on round-robin domain name resolution(DNS) to assign requests to a set of workstations. In [5] this DNS-strategy is extended, the time-to-live (TTL) periods are used to distribute the load more evenly. In the past different researchers tried to identify the workload behavior with emphasis on the development of a caching strategy. NCSA developers analyzed user access patterns for system configurations to characterize the WWW traffic in terms of request count, request data volume, and requested resources [9]. Arlitt and Williamsion [3] studied workloads of different Internet Web servers. Their emphasis placed on finding universal invariants characterizing Web servers workloads. Bestavros et. al [2] tried to characterize the degree of temporal and spatial locality in typical Web server reference streams. A general survey on file allocation is given in [13]. In [9] the problem of declustering is solved by transforming the problem onto a MAX-CUT problem.

The data layout method that is presented in this paper is based on the following idea: The goal of the layout is to minimize the number of items that resides on the same disk and are requested often simultaneously. So we examine the access pattern of the past and place those items on different disks. Since the items which are often simultaneously requested are roughly the same at following days, this strategy leads to a good data layout.

The model of our parallel web server is presented in section 2. In section 3 we show that typical access patterns remain constant over some time, and collisions are typical patterns of a web server. In section 4 the data layout strategy is presented and the strategy is studied in detail, compared with other methods and limits of the strategy are shown. The paper finishes on some concluding remarks in section 5.

2 Model

This section presents the model of the parallel web server that is used to describe the data layout algorithm in the next sections. Thus, in our model we concentrate on the aspects that are important for the presentation in the rest of the paper.

A parallel web server is built by the following entities: A number of processing modules that are connected by some kind of network or bus architecture, a number of communication devices that connect the processing modules to the external clients accessing the server and requesting information, and a number of storage devices (disks) that are connected to the processing modules.

The parallel web server stores data items (files) on the disks and works as follows:

- The server is able to accept one or more requests arriving via the communication devices from the external clients per time step.
- The server forwards a request to the disk holding the requested item. We assume here, that each item is only stored once on the whole disk pool.
- Every disk can accept only one request per time step. If more than one request is sent to a disk per time step, these requests queue up.
- The processing time for each request on the disk is constant and takes one time unit, so one disk can process only one request in one time unit.
- All disks are independent, so that the server can process a maximum of n requests per time step if n is the number of disks.

In case that two requests are forwarded to the same disk in one time step, one request can be served and the other request has to wait for one time unit. This conflict resolution can be done arbitrarily, i.e. we do not assume a specific protocol here. In case that two (or more) requests access the same storage device (disk) we say, that these requests are *colliding*, i.e. these requests are in *collision*. We say that two requests collide if they arrive on the web server within a time interval of time δ. We have chosen δ to be one second.

The aim of our work is now to develop a data layout strategy in order to map the data items in a way onto the disks, that the requests that arrive at the server lead to a minimal number of collisions and therefore to a minimal latency in answering the data requests issued by the external clients.

3 Monitoring the access to web servers

In order to increase the overall performance of a parallel web server it is not important to balance the overall number of requests issued to the disks as evenly as possible, but to avoid that a larger number of requests are submitted to a single disk. This means that in each small time interval the load has to be distributed as evenly as possible minimizing the latency time for a request this way. Thus, the aim is to minimize the number of collisions on the disks.

In order to get an impression of the collisions that occur we have taken the log files from the web server of the University of Paderborn (www.uni-paderborn.de) for November and December 1997. We built all tuples (i, j) of files i and j stored on the web server in Paderborn and measured the number of collisions, i.e. the number of hits that we made on files i and j in a time interval δ. Figure 1 shows the number of collisions for all pairs of files for two consecutive days sorted by the collisions of the first day. The x-axis lists all tuples (i, j) of files according to the collisions on the first day, and the y-axis lists the number of collisions for each pair of files. Now it is interesting to observe that the collisions are very similar on the two consecutive days.

So if we develop an algorithm, that minimizes the collisions for a typical access pattern of the web server monitored at one day, these collisions will also

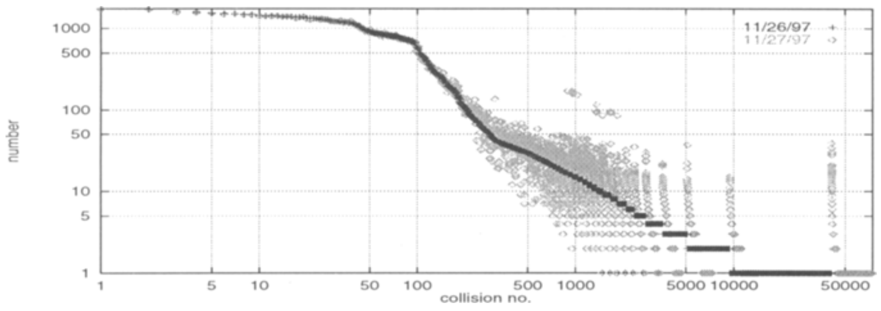

Fig. 1. Distribution of collisions

be eliminated on the next day. Thus, we can take the similarity of access patterns into account for the construction of our data layout strategy. This is the basic observation and the foundation of our data layout principle.

4 Data layout strategies

As described above the data layout strategy aims at minimizing the number of collisions for a typical access pattern. The mapping that is computed in this way can be assumed to also avoid a large number of collisions in the future as the distribution of collisions is very similar on consecutive days.

In the following we will firstly explain the algorithm used to compute the data layout, and secondly evaluate the performance of this algorithm in detail.

4.1 Algorithm

Throughout the rest of the paper we define F to be the set of data items (files) and $\{(t_1, f_1), (t_2, f_2), \ldots, (t_m, f_m), \ldots\}$ be an access pattern for a web server with t_i being the time when the request for data item (file) $f_i \in F$ arrives on the server. Then $c(i,j) = \mid \{\{(t,i),(t',j)\} \mid\mid t - t' \mid \leq \delta\} \mid$ is defined as the number of collisions of files i and j for the given access pattern.

Our strategy is to distribute the objects stored on the web server onto the given disks in such a way so that collisions are minimized. For a given access pattern this leads to the following algorithmic problem:

given: A set of data items F, the access pattern, and a given number of storage devices n.

question: Determine mapping π, $\pi = \min_{l:F \to \{1,\ldots,n\}} \sum_{i,j \in F, l(i)=l(j)} c(i,j)$.

It is easy to map this problem to the MAX-CUT-problem. The MAX-CUT-problem is defined as follows:

given: A graph $G = (V, E)$, weights $w(e) \in I\!N$ and a number of partitions n.

question: Determine a partition of V into n subsets, such that the sum of the weights for the edges having the endpoints in different subsets is maximal.

The mapping is done in a way that the data items (files) are the nodes of the graph that has to be partitioned, and the number of collisions $c(i,j)$ determines the weight of edge $\{i,j\}$. The MAX-CUT problem is known to be NP-hard [6]. However, there are good polynomial approximation algorithms which deliver good solutions. In our experiments we make use of the PARTY-Library [12] containing an efficient implementation of an extension from the partitioning algorithm described in [8].

4.2 Collision Resolving if access pattern are known in advance

In a first step we examine the gain of our algorithm if the access pattern and therefore the collisions are known in advance. Therefor we take the access statistics of one day and determine the number of collisions of each pair of data items. On the base of these statistics, we build the graph, partition the graph, and look how many collisions have remained and how many have been resolved.

In the following we compare the results of our algorithm with the random mapping strategy for a number of access patterns. Each access patterns exactly represents all requests that were issued to the server during one day. We compare the number of collisions induced by the access pattern (k_a) with the number of remaining collisions that occur when applying the mapping algorithm described above (k_r). The factor f describes the ratio between the number of remaining collisions for the random mapping (which is $\frac{k_a}{n}$) and the mapping that is determined by the algorithm (k_r).

Table 1. Results for a number of access patterns, each representing one day

				results, $n = 8$		
day	nodes	edges	k_a	k_r	$\frac{k_r}{k_a}[\%]$	f
sun 11/23/97	5533	15148	75334	3152	4.18	2.99
mon 11/24/97	8877	48136	239365	12100	5.06	2.50
tue 11/25/97	7932	43720	228870	11367	4.97	2.52
wed 11/26/97	8206	41172	215825	10800	5.00	2.50
thu 11/27/97	7464	44656	231364	12021	5.20	2.40
fri 11/28/97	6976	30919	174120	8671	4.98	2.51
sat 11/29/97	5065	9059	37702	1305	3.46	3.61
sun 11/30/97	4798	8657	42544	1579	3.71	3.37

In Table 1 the statistics and results of the partition of one week are shown. The table shows that all collisions up to 4 - 5 percent can be resolved and that the algorithm has about 2 to 3 times the performance of the random mapping.

4.3 Realistic optimization

In this section we examine how many collisions can be resolved if we use the access-statistics of the past. This approach can only be successful if the collisions of successive days have some similarity. We already examined this similarity in section 3.

Table 2 presents the average results for a number of days where the mapping was determined by the access pattern of day $i-1$ and this mapping was used for collision avoiding of day i. The table shows the factor f_{pd} comparing the performance of the random mapping method with the algorithm presented above in respect to the number of disks n. It also shows the percentage of remaining collisions $\frac{k_{r,pd}}{k_a}[\%]$ that could not be resolved. It is shown that the number of remaining collisions with this mapping is clearly lower than with a random mapping. The advantage of our mapping increases with the number of available disks.

Table 2. Comparison of random placement and algorithm using access pattern of the previous day and algorithm using access pattern of the same day

n	$\frac{k_{r,pd}}{k_a}[\%]$	f_{pd}	$\frac{k_{r,sd}}{k_a}[\%]$	f_{sd}	$\frac{\frac{k_a}{n}-k_{r,pd}}{\frac{k_a}{n}-k_{r,sd}}$
2	45.2	1.11	43.1	1.16	0.70
3	27.4	1.22	24.9	1.34	0.70
4	18.7	1.34	16.3	1.54	0.72
5	13.7	1.47	11.3	1.78	0.72
6	10.5	1.61	8.2	2.05	0.73
7	8.2	1.76	6.0	2.41	0.73
8	6.7	1.87	4.7	2.73	0.74

The table also shows the loss which occurs from the fact that the access pattern of successive days are not identical. To see this the according percentage of remaining collisions $\frac{k_{r,sd}}{k_a}[\%]$ and the factor f_{sd} are given which could be achieved if the access pattern of each day would be known in advance.

The value of the term $\frac{\frac{k_a}{n}-k_{r,pd}}{\frac{k_a}{n}-k_{r,sd}}$ shows the relative performance difference of the random mapping and the data layout determined by the algorithm presented above, for the case that the algorithm knows the exact access pattern or only knows the access pattern of the day before. The results show that the performance of our method decreases by about 30 percent if the data layout is computed on the basis of the access pattern from day $i-1$ instead of day i if the access pattern of day i is applied. This loss seems to be nearly independent from the number of disks but becomes smaller for larger number of disks.

In general the results show, that the data layout that is based on the access pattern of a previous day leads to a large reduction of the collisions later on.

4.4 Using a number of access patterns to determine the data layout

Up to now we only used the access pattern of one day to compute the data layout. Here we will present results for the case that a number of access patterns

Table 3. Results of using larger access patterns for determination of data layout

days	2 partitions avg	2 partitions max	2 partitions min	8 partitions avg	8 partitions max	8 partitions min
1	44.65	44.99	43.95	6.58	7.08	5.96
2	44.49	44.92	43.60	6.40	6.90	5.80
3	44.35	44.91	43.26	6.33	6.66	5.71
4	44.38	44.85	43.43	6.35	6.93	5.78
5	44.48	45.21	43.47	6.28	6.83	5.63
6	44.48	45.15	43.66	6.26	6.85	5.64
7	44.48	45.24	43.51	6.30	6.88	5.53
8	44.42	44.94	43.73	6.26	6.88	5.68
9	44.46	45.37	43.63	6.24	6.86	5.58
10	44.39	45.22	43.32	6.25	6.71	5.58
11	44.46	45.45	43.36	6.27	6.80	5.63
12	44.38	45.07	43.40	6.28	6.75	5.64
13	44.40	45.30	43.47	6.33	6.83	5.73
14	44.37	44.91	43.43	6.29	7.00	5.67

were used to determined this layout. All collisions from these patterns were taken into account to determine the data layout.

The results in Table 3 were determined using m previous days of a fixed date i and determining the data layout using all the collisions that occur in these m previous days. The experiments were made for different start dates i, so average, maximum and minimum could be studied. The table presents the percentage of the collisions that could not resolved for the access pattern of date i using the different data layouts. It is shown, that the best results were achieved with a data layout that is determined using only a few days of access patterns. A larger number of access patterns does not increase the overall performance of the data layout method.

4.5 Update frequency for data layout

Using the results of the previous sections we know that the data layout method presented here provides reasonable performance improvements to randomization strategies when it considers the access pattern. The results also show that a typical access pattern which is used to determine the data layout should contain the access information of only a few days. No better results could be made with more information. The question is now, if the data layout is determined on a day i taking the access pattern of day $i-1$, $i-2$, ... $i-x$ into account (small x) how will the performance be on a day $i+d$, thus d days in the future. This will answer the question how often the data layout has to be updated to achieve the best results.

The following experiment was done to answer this question: A data layout was determined taking the access pattern of data i into account. Then the access patterns of the days $i+d$, was applied to this data layout and the number of

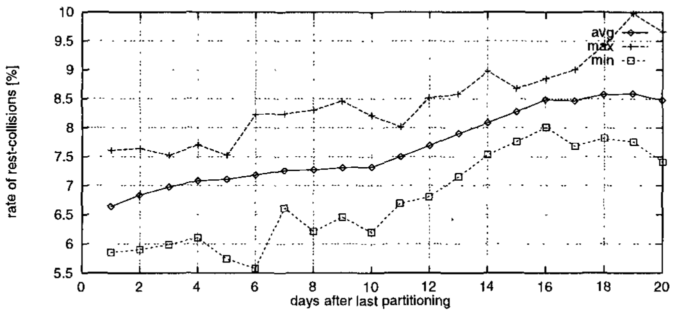

Fig. 2. Influence of time since last update of data layout on performance

collisions that had not been resolved was measured. For different values (x-axis) the results are shown in Figure 2. As the results were gained for different starting days i, average values, maximum and minimum could be determined for every d. The results show, that the number of unresolved collisions is steadily increasing, thus the data layout should be updated as often as possible (taking the last access pattern into account) to achieve the best results.

5 Conclusions

This paper presents a new strategy for allocating data items onto the storage devices of a parallel or sequential web server that contains a number of disks to store this data. The method is based on the observation that requests that often arise in parallel on the web server should be forwarded to different storage devices. Using this observation a graph-theoretic formulation is developed and the problem is reduced to a MAX-CUT problem that is solved using heuristics.

The results show that the performance improvements of the strategy are considerable compared to randomization strategies that are usually used and that the performance improvement scales up with the number of disks used in the parallel server. The various investigations presented in the paper show that it makes no sense to observe the access to the parallel web server for a long time to determine a typical access pattern which then builds the basis for the determination of a data layout and that the data layout has to be updated from time to time as for a fixed data layout the number of resolved collisions decreases steadily.

Future work will focus on the investigation of an extended method that determines the number of duplicates for data items considering different restrictions (in terms of disk capacity, or number of copies) into account.

References

1. M. Adler, S. Chakrabarti, M. Mitzenmacher, L. Rasmussen: Parallel Randomized Load Balancing. *Proc. 27th Annual ACM Symp. on Theory of Computing (STOC '95)*, 1995.
2. Virgilio Almeida, Azer Bestavros, Mark Crovella and Adriana de Oliveira: Characterizing Reference Locality in the WWW. *Proceedings of PDIS'96: The IEEE Conference on Parallel and Distributed Information Systems*, December 1996.
3. Martin Arlitt and Carey Williamson: Web server workload characterization: The search for invariants. *Proceedings of ACM SIGMETRICS'96*,1996.
4. D. Andresen, T. Yang, V. Holmedahl and O. Ibarra: SWEB: Towards a Scalable WWW Server on MultiComputers, *Proceedings of the 10th International Parallel Processing Symposium (IPPS'96)*, April 1996.
5. M. Colajanni and P.S. Yu: Adaptive TTL schemes for Load Balancing of Distributed Web Servers, *ACM SIGMETRICS Perfermance Evaluation Review*, 1997, volume 25,2
6. R. M. Karp: Reducibility among combinatorial problems, in R. E. Miller and J. W. Thatcher, *Complexity of Computer Computations*, pages 85–103, 1972
7. Eric Dean Katz, Michelle Butler and Robert McGrath: A scalable HTTP server: The NCSA prototype. *Computer Networks and ISDN Systems*, volume 27, pages 155–164, 1994.
8. B.W. Kernighan and S.Lin. An effective heristic procedure for partitioning graphs. *The Bell Systems Technical Journal*, pages 192–308, Feb 1970.
9. Kwan, T.T., R.E. McGrath and D.A. Reed: NCSA World Wide Web Server: Design and Performance. IEEE Computer, November 1995, 28(11), pages 68-74.
10. Y.H. Liu and P. Dantzig and C.E. Wu and J. Challenger and L.M. Ni, A distributed {Web} server and its performance analysis on multiple platforms. *Proceedings of the 16th International Conference on Distributed Computing Systems*, IEEE, 1996
11. D.R. Liu and S. Shekhar. Partitioning Similarity Graphs: A Framework for Declustering Problems, *Information Systems*, 1996, volume 21,6, pages 475–496
12. R. Preis and R. Diekmann. The PARTY Partitioning – Library User Guide. *Technical Report tr-rsfb-96-024*, University of Paderborn.
13. Benjamin W. Wah., File placement on distributed computer systems. *Computer Magazine of the Computer Group News of the IEEE Computer Group Society*, 17(1), January 1984.