SAN Optimal Multi Parameter Access Scheme

Christian Schindelhauer* and Gunnar Schomaker

Heinz Nixdorf Institute, Paderborn University

Abstract. Storage area networks consist of a set of n data servers that handle a collection of m documents. Such SANs can minimize the access time to documents by distributing each document among the servers. So, users can access (read or write) documents in parallel. This paper describes an efficient solution for providing parallel access to multiple hard disks for popular content. In extension to previous approaches we provide an efficient and elegant hash table data structure for utilizing the full capacity of each data server. For the dynamics documents as well as server may be added or removed from the system causing only local changes. Our scheme bounds the effort for data replacement by a constant factor with respect to the size of the document added or removed and the size of the server added or removed. Furthermore, we bound the number of data fragments by $\mathcal{O}((n+m)\log n)$. All properties hold with high probability, i.e. $1-n^{-c}$ for any fixed c. As a generalization we present a weighted consistent hashing where the data is distributed according to a general matrix. In the second part we concentrate on how to define this matrix to optimize access time for accessing one document. We consider sequential and parallel access to data in the average case. For the average time model we present a fast optimal algorithm. At last we show how these method can be applied to storage-area-networks.

1 Introduction

Storage systems become ever more important in times where the availability and exchange of information soar up. Growing capacities of storage devices and their parallel use by storage servers or even desktop computers can cope with this growth. The main advantage of parallel used storage devices is the growth in speed and robustness to hardware failure. However, the question of a fairly balanced and efficient distribution of data on a parallel array of storage servers is not completely settled, yet. One of the main problems states the heterogeneity of servers in capacity and bandwidth, combined with the varying popularity of differently sized documents. The other main problem is the dynamics of data added and removed from the system as well as the storage devices may be added and removed during run-time.

This last type of dynamics becomes crucial if one considers peer-to-peer-networks implementing storage systems where the bottleneck is the uplink bandwidth to the Internet (and not the mechanical limitations of the hard disk). In such networks unlike to Storage Area Networks (SANs) the transient attendance is the usual case, while other questions like optimization the download speed or distributing the data, are more or less equivalent important for SANs, too.

We shortly exemplify the relevance of our approach. It regularly occurs that a previously unimportant document becomes such much attention that the access to this document swamp the server or a small group of peers. For web links holding suddenly increasing requested information this effect is sometimes called ”slashdotted”. Thus strategies to avoid these effects are highly needed and and if known it is worthwhile to combine them without negative side effects.

This effect was the motivation for mapping a set of data elements to a dynamic set of hosts by using the notion of Consistent Hashing, aka. Distributed Hash Tables is introduced by Karger et al. [7]. Such a scheme gives a mapping from a set of data elements to a dynamic set of hosts. In the original setting data was to be distributed among different sets of hosts, called views. The goal was to avoid swamped servers by decreasing the usage of memory and to balance data fairly among the views. For this, the hosts are mapped to some range using

* Institute of Computer Science, {schindel,pinsel}@uni-paderborn.de. Partially supported by the DFG-Sonderforschungsbereich 376 and by the EU within 6th Framework Programme under contract 001907 “Dynamically Evolving, Large Scale Information Systems” (DELIS).

a hash function. Also data elements are mapped to the very same range using an appropriate hash function. Now in every view (relevant sub-set of hosts) a data element is stored on a host, if this host's image in the hash range minimizes the distance to the image of the data element. If a sufficient number of copies of the hosts are mapped to the range, one can show that this leads to a fair balance of data elements [10, 3]. Now, if a new host is added to this system, then only data elements need to be reassigned which will be stored on the new host. This feature is called consistency.

Such distributed hash tables are universally applicable to many areas of distributed computing. First they are introduced to distribute web sites among servers distributed around the globe [7] relieving hot spots in the Internet. Besides the area of Web Caching where they are used for the first time, and they are popular in Peer-to-Peer Networks, see CAN [11], Chord [14], Pastry [5], Tapestry [6], and many more.

Distributed Hash Tables are intrinsically homogeneous and it takes some effort to adopt mechanism for dealing with heterogeneous servers or documents, which is the main goal of this paper. For SANs Brinkmann et. al. [2] have presented two adaptive hashing strategies called *SHARE* and *SIEVE* dealing with heterogeneity of server storage capacities to overcome problems induced by huge RAID systems with heterogeneous disk capacities. Both methods ensure that the probability for a request to be sent to disk i is closed (SHARE) or equal (SIEVE) to the capacity of disk i given by a parameter w_i, where $w_i \in [0, 1]$ for each i and $\sum_i w_i = 1$ denotes the capacity distribution of the system. This method was recently improved by the authors in [13] by an elegant, fairer and more efficient approach, called weighted distributed hash table (aka. weighted consistent hashing or distributed heterogeneous hash tables). In this paper we considerably extend the degree of heterogeneity to documents **and** servers being differently sized at the same time. Furthermore, we optimize time for accessing the documents, which has not been done before in the context of DHT.

Our approach can be applied wherever data needs to be assigned to a distributed set of storage servers. We show that especially in the area of storage area networks and peer-to-peer networks our approach implies optimal results.

1.1 Related Research

Heterogeneity in SAN Networks using parallelism In SAN many strategies are known to distribute content among several disk, but less are able to respect different disk capacities or bandwidths and only a few include dynamics if the system configuration changes. The more classical approaches are described in the RAID level scheme. They were introduced first in [9] to compensate the cost for huge and fast disks with smaller once using parallelism. Over time other aspects like fault-tolerance derived by including copies, parity information, and erasure codes are added and combined with each others according to the mentioned requirements.

Several aspects of data partitioning and load balancing in parallel disk systems are discussed in Scheuermann et al. [12]. They have discussed striping including technical aspects like head positioning and arm movement with respect to average request size over all files. Unfortunately with less devices compared to nowadays, where a SAN system contains tens or hundreds of disks. Nevertheless many of their observations are still important if parallelism is an issue.

Typically such block device systems utilizes n disks by defining a virtual block of size $|b_v|$ which describes the fraction of data each disk derives or has to deliver if a request arrives. The original strategy was to divide such blocks in equal sized peace of $|b_v|\frac{1}{m}$ where $\frac{n}{c} = m \in \mathbb{N}$. Thus only disks with same capacities are reasonable in this scheme if no remaining capacity should be left unused. Cortes et al. [4] presented a static solution to overcome the problem of uniform disks by introducing *AdapRraid0*. They used virtual blocks where the fraction of each disk reflects its capacity ratio compared to each disk included in such block. Nevertheless they assumed that disks with higher capacities are faster and thus the result improves the bandwidth. Zimmermann et al. [15] have used the similar approach, but they focussed in *HERA* on reliability and presented a scheme that includes redundancy based on parity information and presented an analytical Marcow model of reliability in heterogeneous disk arrays.

If such systems are changed by replacing a disk d with a new disk d' and $|d| < |d'|$ the additional capacity $|d'| - |d|$ is left unused or intensive recalculations must be done to preserve consistency. Similar effects happen if disks are added or removed. Furthermore the size of a virtual block grows linear in the number of disks it joins,

on the one hand this leads to wasted storage if files are small and on the other hand if the block size is constant the benefit of parallelism decreases caused by the seek time on each disk.

Di Marco et al. presented a distributed disk array architecture called *DRAID* [8] accessing $N + K$ disks in parallel. They used a gigabit Ethernet cluster infrastructure focussed improved the fault tolerance compared to RAID IV. Therefor they included Reed-Solomon error correction organized in a two dimensional matrix. and argued that the need for such multi fault tolerance is justified by the average uptime of systems in such an infrastructure. They have considered scaling of their system too, but eventually adding single disks is inoperative, because of the recalculation needed for the Reed-Solomon encoding. To overcome this they suggest to add disks in multiples of $N + K$

Another important issue is scaling or more generally adaptability. As seen before all mentioned schemes have tremendously problems if disks are added or removed from the system and raising problems occur if these disks have different capacities or bandwidths. All these classical approaches have in common that they are unable to react on changes with an appropriate effort concerning data movement to obtain consistency. New strategies are needed to overcome this. As mentioned before Brinkmann et al. [1] tried to overcome this by introducing and implementing consistent hashing for SAN. They claim to completely solve the case for dynamics in SAN. Nevertheless, their solution is not consistent and some technical problems are caused by the algorithmic approach used. These problems are pointed out in [13] and are completely resolved there. They use an easy to implement model comprising may additional features missed before like guaranteed consistency, capacity fading, self-scaling balancing in the *Logarithmic Method*, the potential for implicit fault tolerance and many more.

Eventually, a general SAN strategy including popularity of differently sized documents distributed over heterogeneous storage devices with different capacities and bandwidths states an open problem, which will be discussed and partially solved by this paper.

1.2 Organization of the Paper

In the next section we present three methods for distributing documents among multiple servers. The first method restates the results presented in [13] where the heterogeneity is one-side, i.e. either same sized documents are distributed fairly to servers with different capacities or documents with varying sizes are distributed to a homogeneous set of servers. The second method tackles the natural extension of two-sided heterogeneously sizes (servers and documents). The third method discussed in the next section receives as input a matrix $A_{i,j}$ describing the amount of data of document i to be distributed to server j. This matrix is dynamic, i.e. entries change as well as rows and columns may appear or disappear. This matrix generalizes the two-sided heterogeneous setting to arbitrary weightings.

In Section 3 we show how to define the entries of this matrix to optimize the time for accessing to documents. We investigate sequential and parallel as well as worst case and average case leading to linear optimization problems. In Section 4 we present some optimal solutions for optimizing the average measures (while the worst case measures can be solved by linear optimization). In Section 5 we show how to apply this to storage area networks and to peer-to-peer-networks. In the concluding Section 6 we summarize and discuss our results and present some open questions.

2 Distributed Heterogeneous Hash Tables (DHHT)

DHHT is an generaliziation of the *Consistent Hashing* introduced by Karger et. al. in [7]. It uses a similar technique to map a dynamic set of documents D to a dynamic set of servers S, by using a hash range M $[0, 1)$, whereas the assignment of documents to servers strongly differs. In the original consistence hashing scheme S only consists of n homogeneous servers. The newer approach [13] overcomes this artificial restriction and adds to each $s_i \in S$ a corresponding weight w_i that reflects the capacity of each server. This extensions allows to solve the following Problem.

Definition 1. *The Heterogeneous Distribution Problem: Given a dynamic set of servers $S = \{s_1, \cdots, s_n\}$, a weighting function $w : S \mapsto \mathbb{R}^+$, and a dynamic set of documents $D = \{d_1, \cdots, d_m\}$. Find a mapping function $f_{S,w} : D \mapsto S$ with following properties:*

- *Simplicity, which means, f() uses S, w, d as input and is calculated without the knowledge of $D \setminus \{d\}$*
- *Fairness, such that each server gets a comparative portion of data with respect to its weight: $\forall u, v \in S$, $f^{-1}_{S,w}(v)/w(v) \approx f^{-1}_{S,w}(u)/w(u)$, where $f^{-1}_{S,w}(s) := \{d \in D : f_{S,w}(d) = s\}$*
- *Consistency means, if $|S|$ or $w(s)$ changes, the number of data reallocation steps that are needed to preserve fairness are minimal. With other words no unnecessary data movements!*

As mentioned before the DHHT approach offers two different schemes, called the *Linear Method* and the *Logarithmic Method*, which both can solve the heterogeneous problem. For regarding different properties of both schemes in detail, we would like to refer to the article [13]. For easer comprehension the basic steps of how-to assign a document to a set of servers are explained here:

1. Choose for each $s \in S$ a random position rs in M via hash function, $rs = h(s)$
2. Choose for a document $d \in D$ a random position rd in M via hash function, $rd = h(d)$
3. Compute the height of the document d at the position rd for each server s. Depending on the scheme use $h(d) = ((rs - rd) \ mod \ 1)/w$ for the *Linear Method* and $h(d) = -ln((1 - (rs - rd)) \ mod \ 1)/w$ for the *Logarithmic Method*, where $a \ mod \ 1 := a - \lfloor a \rfloor$
4. Assign the document to the server which minimize the height of d at the position rd

The resulting mapping and the thus responsibilities of the hash interval for servers is illustrated in Fig. 1. One can see how the capacity of servers and the length of the appropriate interval sections are correlating.

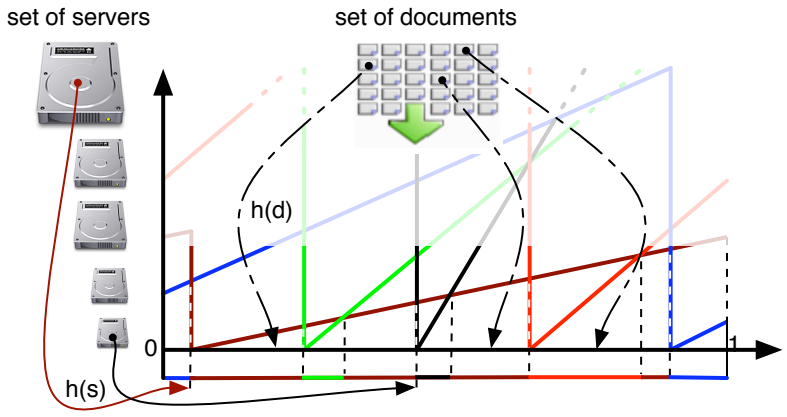

Fig. 1. DHHT Server and Data mapping scheme using the *Linear Method*

Compared to existing approaches like SHARE or SIVE [2], which also try to solve the heterogeneous problem, the new models within the DHHT approach promises useful and none overhead causing benefits for SANs like:

- Coverage of the hash interval for data assignment is guaranteed since $|S| \geq 1$ and is independent of joining or leaving servers.
- Direct and unique data assignment for new data or data reallocation caused by joining or leaving Servers.
- Capacity variations of servers during runtime are allowed and adjustable with minimal effort.

Actually the *Logarithmic Method* can also be used for homogeneous servers and heterogeneous documents, by exchanging the roles of Servers and Documents.

Corollary 1. *Consider n same sized servers $s_1, \ldots, s_n$ with and m heterogeneous documents $d_1, \ldots, d_m$. For all $\epsilon > 0$ and $c > 0$ there exists $c' > 0$, where we apply the* Logarithmic Method *with $c' m \log n$ document fragments. Then with high probability, i.e. $1 - n^{-c}$, we achieve ϵ-fairness.*

2.1 Distributing Documents according a given Matrix

In some applications high fragmentation is no disadvantage and for weighted consistent hashing it improves the performance. For the rest of this paper we abandon the objective of small fragmentation. A straight-forward generalization of fair balance is to distribute data according to a $m \times n$ distribution matrix A where

$$\forall s: \quad \sum_d A_{d,s} \leq |s| \qquad \text{and} \qquad \forall d: \quad \sum_s A_{d,s} = |d| \; . \tag{1}$$

Then, the goal is to distribute $A_{d,s}(1 \pm \epsilon)$ data elements of d to server s and provide a lookup operation that can tell for each bit of the document on which server it has been stored using only the document IDs, the server IDs and A as input. Furthermore if entries of the matrix A change yielding to the new matrix A' then at most $(1+\epsilon)\sum_{d,s}\left|A_{d,s} - A'_{d,s}\right|$ data elements need to be reassigned between servers. This is the matrix version of the consistency of the weighted hash function.

Using the above introduced hashing the solution is: Apply the *Logarithmic Method* with $\mathcal{O}(\log n)$ hash spaces $M_1, M_2, \ldots$ with weights $(A_{d,s})_{s\in\mathcal{S}}$ on each document to determine the server s for each data element d. Then, the data elements are stored on the server using either weighted consistent hashing again (or with some internal centralized algorithm if applicable).

Theorem 1. *For all $\epsilon > 0$ and $c > 0$ there exists $c' > 0$, such that if we apply the above method with $\mathcal{O}(nm(\log n)^2)$ document fragments, then with high probability, i.e. $1 - n^{-c}$, we achieve ϵ-consistency for any given matrix, i.e. $|d \cap s| = (1 \pm \epsilon)A_{d,s}$.*

Proof. follows by the construction above, the logarithm method and by Chernoff bounds.

3 Measures for Time

Clearly, chosing $A_{d,s} = |d| \cdot \frac{|s|}{\sum_{s'\in\mathcal{S}}|s'|}$ gives an alternative solution of providing a fair balance to each server. A fair balance is not necessarily the best thing to do when documents are stored on heterogeneous servers with differing bandwidth. In many applications it is more desirable to utilize the full storage of high performance servers before storing data on slower servers. In this section we discuss possible optimization objectives. As general restraints we have 1. Further, let $b(s)$ denote the available bandwidth of server s and $p(d)$ denote the popularity of a document. We will now present four different optimization functions resulting in linear programming problems.

We assume that these sizes can be modeled by positive real numbers motivated by the fact that we face large documents and servers. We are aware that a discrete model increases the problem complexity and our solution is not applicable in general any more. We leave the problem of a discrete approach as an open problem for further research.

We distinguish two cases, servers that can be used only sequentially like zones on a single hard disk, where each zone has different bandwidths and two zones cannot be used at the same time. Then the time to read a document is the sum of times for each server (zone) giving by the amount of data $A_{d,s}$ divided by the bandwidth $b(s)$. Furthermore, we consider also the parallel case, where all servers work independently and the bottleneck is given by the slowest server.

Definition 2. *The sequential time and parallel time for a document d and assignment A are defined by*

$$\text{SeqTime}_A(d) := \sum_{s\in\mathcal{S}} \frac{A_{d,s}}{b(s)} \quad \textit{and} \quad \text{ParTime}_A(d) := \max_{s\in\mathcal{S}} \left\{ \frac{A_{d,s}}{b(s)} \right\} \; .$$

4 Optimal Solutions

We now discuss how to obtain the optimal choice for the matrix A depending on the set of documents, set of servers and the underlying measure. All worst case optimization problems and all time optimization problems

Measure	Linear programm	Add. variables	Additional restraint	Optimize
AvSeqTime	yes	—	—	min $\sum_{s\in\mathcal{S}}\sum_{d\in\mathcal{D}} p(d)\frac{A_{d,s}}{b(s)}$
AvParTime	yes	$(m_d)_{d\in\mathcal{D}}$	$\forall s\in\mathcal{S},\forall d\in\mathcal{D}:\frac{A_{d,s}}{b(s)}\le m_d$	min $\sum_{d\in\mathcal{D}} p(d)m_d$

Table 1. The Optimization Problems

can be formulated as linear problems and therefore can be solved in polynomial time if the documents can be fragmented arbitrarily. Since all documents are rather large this leads to a good approximation. For nearly all of the measures we can give closed solutions, which can be computed efficiently.

Table 1 shows all optimization functions and additional restraints besides the restraints (1).

4.1 Solutions for the Average Case

Average Case Sequential Time This is the easiest case. The intuition behind the solution is the following. A document of size $|d|$ is $p(d)$ popular, so storing a bit $\mathtt{b}\in d$ of the document costs $\text{cost}(\mathtt{b}) := p(d)$. This way we receive a virtual cost of every bit.

We can represent the costs by

$$\text{AvSeqTime}(p,A)=\sum_{d\in\mathcal{D}}\sum_{s\in\mathcal{S}} p(d)\frac{A_{d,s}}{b(s)}=\sum_{d\in\mathcal{D}}\sum_{s\in\mathcal{S}}\sum_{\mathtt{b}\in d\cap s}\frac{\text{cost}(\mathtt{b})}{b(s)}\,.$$

Now, if we have two bits b_1 and b_2 with $\text{cost}(b)<\text{cost}(b')$ and two servers with different bandwidth, then the overall sum is smaller if b_1 is stored on the faster server and b_2 is stored slower server (if there is a choice).

This implies that the optimal solutions has to sort all bits according to costs and then store them in that order on the servers sorted according to the bandwidth. We call this the *serial time assignment* and it is provided by the greedy algorithm shown in Fig. 2 and these considerations sketch the proof of the following theorem.

Serial Time Assignment$(\mathcal{D},\mathcal{S})$
begin
  Sort $\{s_1,\dots,s_n\}=\mathcal{S}$ such that $b(s_i)\ge b(s_{i+1})$ for $i\in[n-1]$
  Sort $\{d_1,\dots,d_m\}=\mathcal{D}$ such that $p(d_i)\ge p(d_{i+1})$ for $i\in[m-1]$
  $A_{i,j}\leftarrow 0$ for all $i\in[n], j\in[m]$
  $i\leftarrow 1; j\leftarrow 1$
  $\ell\leftarrow|d_1|; r\leftarrow|s_1|$
  while $j\le m$ **and** $i\le n$ **do**
    if $\ell\le r$ **then**
      $A_{i,j}=\ell$
      $r\leftarrow r-\ell; \ell\leftarrow|d_i|; i\leftarrow i+1$
    else if $\ell>r$ **then**
      $A_{i,j}=r$
      $\ell\leftarrow\ell-r; r\leftarrow|s_j|; j\leftarrow j+1$
    fi
  od
  if $j>m$ **then return**"Capacity exceeded"
  else return A
end

Fig. 2. Algorithm for Serial Time Assignment

Theorem 2. *The Serial Time Assignment provides an optimal solution for* AvSeqTime.

Average Case Parallel Time When the servers $s_1, \ldots, s_n$ can be used in parallel the best choice is to use all servers in parallel for the same time. This leads to a maximal available bandwidth $B := \sum_{i=1}^{m} b(s_i)$. To utilize this bandwidth a document d_i needs to be distributed on all the servers such that the share $A_{i,j}$ is proportional to the server's bandwidth s_j. Then, each document can be delivered in time $|d_i|/B$. However, this is only possible if each server can store enough data with respect to its bandwidth, i.e. $|s_j| \geq \frac{b_j}{B} \sum_{i=1}^{n} d_i$. If this is not the case the situation becomes more difficult (and interesting).

We rearrange all parallel servers to a set of virtual sequential servers $s'_1, \ldots, s'_n$ where server s'_j represents j parallel servers $s_j, \ldots, s_m$ where the servers are ordered such that $\frac{|s_j|}{b(s_j)|} \leq \frac{|s_{j+1}|}{b(s_{j+1})|}$. For each server s'_j the sub-servers are used at optimal bandwidths, which leads to a bandwidth $b(s'_j) = \sum_{i=j}^{m} b(s_i)$. The size of each server s'_j can be computed by solving the equations for $j \in [m]$:

$$|s'_j| = \left(\left(\sum_{i=1}^{m} |s_i| \right) - \left(\sum_{i=1}^{j-1} |s'_i| \right) \right) \frac{b(s_j)}{b(s'_j)}$$

We call this rearrangement of parallel servers to a set of serial virtual servers the **virtual server** method, an illustration can be found in Fig. 3. If we now compute an assignment A' to these virtual sequential servers, it corresponds to an assignment A such that $A_{i,j} = \sum_{k=1}^{j} A'_{i,k} \frac{b(s_k)}{b(s'_j)}$.

Theorem 3. *Applying the serial time assignment to the servers of virtual server method gives an optimal assignment for* AvParTime.

Proof. First note that increasing the document set by partitioning a document d into k documents $d'_1, \ldots, d'_k$ with popularity $p(d)$ and assignment $A_{d'_i,s} = A_{d,s}/k$ does not change the average time behavior. So, we can assume an arbitrarily fine partitioned document set.

Assume an assignment which optimizes the average parallel time. Then, the time of any document cannot be improved unless at least the time of another document increases. Consider the i-th row of the matrix $A_{i,\bullet}$

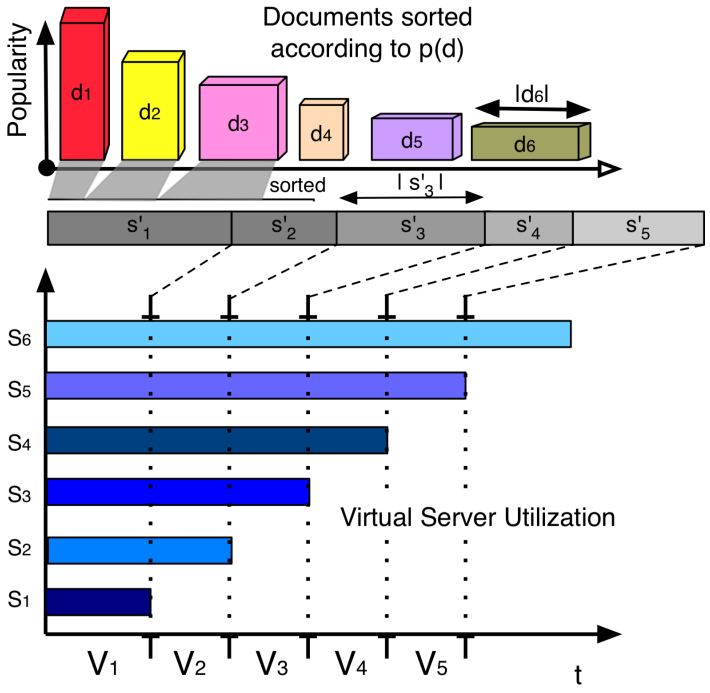

Fig. 3. Virtual server utilization and document assignment

describing how one a document is assigned to the (parallel) servers. Consider two such documents $A_{i,\bullet}, A_{k,\bullet}$ and the set of servers $Q_i := \{s \in \mathcal{S} : A_{i,s} > 0\}$. Then either $Q_i \subseteq Q_j$ or $Q_j \subseteq Q_i$. This follows from the fact if each document uses an extra server s not used by the other document, then the time behavior of both documents can be decreased by sharing these extra servers.

This observation leads to the fact that there is an ordering of the documents such that

$$[m] = Q_1 \supseteq Q_2 \supseteq \ldots \supseteq Q_n \ .$$

Now consider the case $Q_i = Q_{i+1}$ and two documents d_i, d_{i+1}. Now let $M_i := \{s \in Q_i \ : \ A_{i,s}/b(s) = \max_{s'} A_{i,s'}/b(s')\}$. If neither $M_i \subseteq M_{i+1}$ nor $M_{i+1} \subseteq M_i$ then again there is some parallelism left to improve the time of i and $i+1$.

This leads to the ordering

$$[m] \supseteq M_1 \supseteq M_2 \supseteq \ldots \supseteq M_n \ .$$

Now order the servers such that s_1 is the first server that drops out. As long s_1 was in the sets $M_1, \ldots, M_{i-1}$ it contributed to the parallelism and the time behavior. In $M_i, M_{i+1}, \ldots$ the server s_1 does not have any effect to the running time. So we can refrain from using it in $M_{i+1}, M_{i+2}, \ldots$ at all without increasing the overall time. So, we can assume that only in M_i this server is used. Since we can divide the document set into arbitrarily fine partitions without changing the time behavior, the missing contribution of s_1 in M_1 to the parallelism can be neglected and s_1 occurs with all its capacity in $M_1, \ldots, M_i$. So the only choice for the first server s_1 is that one that minimizes $\frac{|s|}{b(s)}$. Following this argument we can also identify s_2, s_3, etc. and end up with the construction of the virtual servers in the order according to $\frac{|s|}{b(s)}$.

Then, it remains to apply Theorem 2 to show that the serial time assignment algorithm optimizes the sequential average time on the virtual servers.

5 Applications

As mentioned in the introduction there are existing several application areas, but in specific we believe that the most benefit of our scheme can be found in huge storage area networks.

Our described technique seems to be applicable for many spacial purpose Storage Systems. One example might be an object storage with different popularity zones. For instance movies are always more or less popular over time. If each movie knows its rank within such Storage System, provided by an external meta data controller, it can be placed on a specific virtual disk in such manner as we considered before. The segmentation is done automatically by the virtualization of the physical disks with its heterogeneous fractions. If such file is evicted by more popular files from a faster to the next slower virtual drive only the content from the "lost disk" needs to be replaced for this file. In addition if such system uses a reduced parallelism by randomly placed vectors of dimension $m \ll n$ and each vector utilizes the storage for a file in a given level, it might happen that there is no need to replace anything at all. Consider that a virtual disk containing n disks consists of $\binom{n}{m}$ vectors!

A second example could be the consolidation of disks zones of independent disks to improve the maximum parallel I/O performance from a specific constellation of disks. This might by possible if zone classes from different disks with nearly same speed, but possibly different capacities are exported. Within our scheme that means that each disk is divided in its zones and these zones are sorted similar to the previous scheme, than virtual disks are created with respect to such zones and used as virtual disks.

6 Conclusions

In this paper we present a distributed method for assigning large documents to data servers. Our method generalizes the approach of distributed hash tables to a completely heterogeneous setting where the documents as well as the data server may have different sizes. Nevertheless, in our Distributed Heterogeneous Hash Tables (DHHT) have perfect consistency and allow the dynamic addition and deletion of servers and document while keeping the rearrangement cost at a minimum.

In addition to this scheme we optimize the parallel use of the data server with respect to average case sequential/parallel time measures. Table 2 summarizes these optimization results. From our point of view the average parallel time measure provide the most natural choices from these measures. The average time measure is invariant to joining or partitioning documents. The matrix assignment can be optimally computed by a greedy algorithm and the changes by dynamics in this matrix can be well predicted.

Measure	**parallel / sequential**	**worst case / average case**	**linear program**	**closed form in this paper**	**virtual servers**
Time	sequential	worst case	yes	no	—
		average case	yes	yes	no
	parallel	worst case	yes	no	—
		average case	yes	yes	yes

Table 2. Results of this paper

An interesting feature of the proposed weighted consistent hashing, in particular the *Logarithmic Method*, is the chance of smoothly fading in of new servers without any overhead (according to data re-assignments). This avoids allocating a large chunk of storage for a new server in one step. The alternative is a new server starts with a very small weight $w = \epsilon$ and slowly increases this weight. If this is the only server entering the system then only data will be assigned to this server which anyway would end there. So, bursty traffic can be avoided this way and the transmission bandwidth can be used more efficiently. Another advantage is that during the insertion of the new server the system state is always defined and in every time step only a little portion of data needs to be kept on two servers.

References

1. A. Brinkmann, F. Meyer auf der Heide, K. Salzwedel, C. Scheideler, M. Vodisek, and U. Rückert. Storage management as means to cope with exponential information growth. In *Proceedings of SSGRR 2003*, L'Aquila, Italy, 28 July - 3 Aug. 2003.
2. A. Brinkmann, K. Salzwedel, and C. Scheideler. Compact, adaptive placement schemes for non-uniform distribution requirements. In *Proc. of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 53–62, Winnipeg, Manitoba, Canada, 11 - 13 Aug. 2002.
3. J. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. Technical report, BU Computer Science, 2002.
4. Cortes and Labarta. A case for heterogeneous disk arrays.
5. P. Druschel and A. Rowstron. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In R. Guerraoui, editor, *Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001, Proceedings*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, 2001.
6. K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *Proceedings of the 14th Annual ACM Symposium on Parallel ALgorithms and Architectures (SPAA-02)*, pages 41–52, New York, Aug. 10–13 2002. ACM Press.
7. D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 654–663, El Paso, Texas, 4–6 May 1997.
8. A. D. Marco, G. Chiola, and G. Ciaccio. Using a gigabit ethernet cluster as a distributed disk array with multiple fault tolerance. In *LCN '03: Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks*, page 605, Washington, DC, USA, 2003. IEEE Computer Society.
9. D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 109–116, New York, NY, USA, 1988. ACM Press.

10. A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. *In 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.
11. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Computer Communication Review*, volume 31, pages 161–172. Dept. of Elec. Eng. and Comp. Sci., University of California, Berkeley, 2001.
12. P. Scheuermann, G. Weikum, and P. Zabback. Data partitioning and load balancing in parallel disk systems. *VLDB Journal: Very Large Data Bases*, 7(1):48–66, 1998.
13. C. Schindelhauer and G. Schomaker. Weighted distributed hash tables. In *Proc. of the 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2005.
14. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In R. Guerin, editor, *Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01)*, volume 31, 4 of *Computer Communication Review*, pages 149–160, New York, Aug. 27–31 2001. ACM Press.
15. R. Zimmermann and S. Ghandeharizadeh. Hera: Heterogeneous extension of raid, 1998.