Dynamic Load Balancing in Distributed Hash Tables *

Marcin Bienkowski† Miroslaw Korzeniowski † Friedhelm Meyer auf der Heide ‡

Abstract

In Peer-to-Peer networks based on consistent hashing and ring topology each server is responsible for an interval chosen (pseudo-)randomly on a circle. The topology of the network, the communication load and the amount of data a server stores depend heavily on the length of its interval.

Additionally the nodes are allowed to join the network or to leave it at any time. Such operations can destroy the balance of the network, even if all the intervals had equal lengths in the beginning.

This paper deals with the task of keeping such a system balanced so that the lengths of intervals assigned to the nodes differ at most by a constant factor. We propose a simple fully distributed scheme which works in a constant number of rounds and achieves optimal balance with high probability. Each round takes time at most $\mathcal{O}(\mathcal{D} + \log n)$, where $\mathcal{D}$ is the diameter of a specific network (e.g. $\Theta(\log n)$ for Chord [15] and $\Theta\left(\frac{\log n}{\log \log n}\right)$ for [12, 11]).

The scheme is a continuous process which does not have to be informed about the possible imbalance or the current size of the network network to start working. The total number of migrations is within a constant factor from the number of migrations generated by the optimal centralized algorithm starting with the same initial network state.

1 Introduction

Peer-to-Peer networks are an efficient tool for storage and location of data since there is no central server which could become a bottleneck and the data is evenly distributed among the participants.

The Peer-to-Peer networks that we are considering, are based on consistent hashing [6] with ring topology like Chord [15], Tapestry [5], Pastry [14] and a topology inspired by de Bruijn graph [12, 11]. The exact structure of the topology is not relevant. It is, however, important that each server has a direct link to its successor and predecessor on the ring and that there is a routine that lets any server contact the server responsible for any given point in the network in time $\mathcal{D}$.

A crucial parameter of a network defined in this way is its *smoothness* which is the ratio of the length of the longest interval to the length of the shortest interval. The smoothness is a parameter which informs about two aspects of load balance: the first one is the storage load of a server; the longer the interval is, the more data has to be stored in the server. The other aspect is the degree of a node; a longer interval has a higher probability of being contacted by many short intervals which increases its in-degree. Apart from that, in [12, 11] it is crucial for the whole system design that the *smoothness* is constant. Even if we choose the points for the nodes fully randomly, the smoothness can be as high as $\Omega(n \cdot \log n)$, whereas we would like it to be constant (n denotes the current number of nodes).

In this paper we concentrate our efforts on the aspect of load balancing related to the network properties, that is we aim at making the lengths of all intervals even. We do not balance the load caused by stored items or their popularity.

1.1 Our results

We present a fully distributed algorithm which makes the smoothness constant using $\Theta(\mathcal{D}+\log n)$ direct communication steps per node.

1.2 Related work

Load balancing has been a crucial issue in the field of Peer-to-Peer networks since the design of the first network topologies like Chord [15]. It was proposed that

*Partially supported by DFG-Sonderforschungsbereich 376 “Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen” and by the Future and Emerging Technologies programme of EU under EU Contract 001907 DELIS ”Dynamically Evolving, Large Scale Information Systems”.

†International Graduate School of Dynamic Intelligent Systems, Computer Science Department, University of Paderborn, D-33102 Paderborn, Germany. Email: {young,rudy}@upb.de.

‡Heinz Nixdorf Institute and Computer Science Department, University of Paderborn, D-33102 Paderborn, Germany. Email: fmadh@upb.de.

each real server works as $\log n$ virtual servers, thus greatly decreasing the probability that some server will get a large part of the ring. Some extensions of this method were proposed in [13] and [4], where more schemes based on virtual servers were introduced and experimentally evaluated. Unfortunately, such approach increases the degree of each server by a factor of $\log n$, because each server has to keep all the links of all its virtual servers.

The paradigm of many random choices [10] wasused by Byers *et al* [3] and by Naor and Wieder [12, 11]. When a server joins, it contacts $\log n$ random places in the network and chooses to cut the longest of all the found intervals. This yields constant *smoothness* with high probability.[1]

A similar approach was proposed in [1]. It extensively uses the structure of the hypercube to decrease the number of random choices to one and the communication to only one node and its neighbors. It also achieves constant *smoothness* with high probability.

The approaches above have a certain drawback. They both assume that servers join the network sequentially. What is more important, they do not provide analysis for the problem of balancing the intervals afresh in case when servers leave the network.

One of the most recent approaches due to Karger and Ruhl is presented in [7, 8]. The authors propose a scheme, in which each node chooses $\Theta(\log n)$ places in the network and takes responsibility for only one of them. This can change, if some nodes leave or join, but each node migrates only among the $\Theta(\log n)$ places it chose and after each operation $\Theta(\log \log n)$ nodes have to migrate on expectation. The advantage of our algorithm is that the number of migrations is always within constant factors from optimal centralized algorithm. Both their and our algorithms use only tiny messages for checking the network state, and in both approaches the number of messages in half-life[2] can be bounded by $\Theta(\log n)$ per server. Their scheme is claimed to be resistant to attacks thanks to the fact that each node can only join in logarithmically bounded number of places on the ring. However, in [2] it is stated that such a scheme cannot be secure and that more sophisticated algorithms are needed to provide provable security.

Manku [9] presented a scheme based on a virtual binary tree that achieves constant *smoothness* with low communication cost for servers joining or leaving the network. It is also shown that the *smoothness* can be diminished to as low as $(1+\epsilon)$ with communication per operation increased to $\mathcal{O}(1/\epsilon)$. All the servers form a binary tree, where some of them (called *active*) are responsible for perfect balancing of subtrees rooted at them. Our scheme treats all servers evenly and is substantially simpler.

2 The algorithm

In this paper we do not aim at optimizing the constants used, but rather at the simplicity of the algorithm and its analysis. For the next two subsections we fix a situation with some number n of servers in the system, and let $l(I_i)$ be the length of the interval I_i corresponding to server i. For the simplicity of the analysis we assume a static situation, i.e. no nodes try to join or leave the network during the rebalancing.

2.1 Estimating the current number of servers

The goal of this subsection is to provide a scheme which, for every server i, returns an estimate n_i of the total number of nodes, so that each n_i is within a constant factor of n, with high probability.

Our approach is based on [2] where Awerbuch and Scheideler give an algorithm which yields a constant approximation of n in every node assuming that the nodes are distributed *uniformly at random* in the interval $[0, 1]$.

We define the following infinite and continuous process. Each node keeps a connection to one random position on the ring. This position is called a marker. The marker of a node is fixed only for $\mathcal{D}$ rounds during which the node is looking for a new random location for the marker.

The process of constantly changing the positions of markers is needed for the following reason. We will show that for a fixed random configuration of markers our algorithm works properly with high probability. However, since the process runs forever, and nodes are allowed to leave and join (and thus change the positions of their markers), a bad configuration may appear at some point in time. We assure that the probability of failure in time step t is independent of the probability of failure in time step $t + \mathcal{D}$, and this enables the process to recover even if a bad event occurs.

Each node v estimates the size of the network as follows. It sets initially $l := l_v$ which is the length of its interval and $m := m_v$ which is the number of

[1] With high probability (w.h.p.) means with probability at least $1 - \mathcal{O}(\frac{1}{n^l})$ for arbitrary constant l.

[2] Half-life of the network is the time it takes for half of the servers in the system to arrive or depart.

markers its interval stores. As long as $m < \log \frac{1}{l}$, the next not yet contacted successor is contacted and both l and m are increased by its length and the number of markers, respectively.

After that, l is decreased so that $m = \log \frac{1}{l}$. This can be done locally using only the information from the last server on our path.

The following Lemma from [2] states how large l is when the algorithm stops.

Lemma 1 *With high probability, $\alpha \cdot \frac{\log n}{n} \leq l \leq \beta \cdot \frac{\log n}{n}$ for constants α and β.*

In the following corollary we slightly reformulate this lemma in order to get an approximation of the number of servers n from an approximation of $\frac{\log n}{n}$.

Corollary 2 *Let l be the length of an interval found by the algorithm. Let n_i be the solution of $\log x - \log\log x = \log(1/l)$. Then with high probability $\frac{n}{\beta^2} \leq n_i \leq \frac{n}{\alpha^2}$.*

In the rest of the paper we assume that each server has computed n_i. Additionally there are global constants l and u such that we may assume $l \cdot n_i \leq n \leq u \cdot n_i$, for eeach i.

2.2 The load balancing algorithm

We will call the intervals of length at most $\frac{4}{l \cdot n_i}$ *short* and intervals of length at least $\frac{12 \cdot u}{l^2 \cdot n_i}$ *long*. Intervals of length between $\frac{4}{l \cdot n_i}$ and $\frac{12 \cdot u}{l^2 \cdot n_i}$ will be called *middle*. Notice that *short* intervals are defined so that each *middle* or *long* interval has length at least $\frac{4}{n}$. On the other hand, *long* intervals are defined so that halving *long* interval we never obtain a *short* interval.

The algorithm will minimize the length of the longest interval, but we also have to take care that no interval is too short. Therefore, before we begin the routine, we force all the intervals with lengths smaller than $\frac{1}{2 \cdot l \cdot n_i}$ to leave the network. By doing this, we assure that the length of the shortest interval in the network will be bounded from below by $\frac{1}{2 \cdot n}$. We have to explain why this does not destroy the structure of the network.

First of all, it is possible that we remove a huge fraction of the nodes. It is even possible that a very long interval appears, even though the network was balanced before. This is not a problem, since the algorithm will rebalance the system. Besides, if this algorithm is used also for new nodes at the moment of joining, this initialization will never be needed. We do not completely remove the nodes with too short intervals from the network. The number of nodes n and thus also the number of markers is unaffected, and the removed nodes will later act as though they were simple *short* intervals. Each of these nodes can contact the network through its marker.

Our algorithm works in rounds. In each round we find a linear number of *short* intervals which can leave the network without introducing any new *long* intervals and then we use them to divide the existing *long* intervals.

The routine works differently for different nodes, depending on the initial server's interval's length. The *middle* intervals and the *short* intervals which decided to stay, help only by forwarding the contacts that come to them. The pseudocodes for all types of intervals are depicted in Figure 1.

Theorem 3 *The algorithm has the following properties, all holding with high probability:*

1. *In each round each node incurs a communication cost of at most $\mathcal{O}(\mathcal{D} + \log n)$.*
2. *The total number of migrated nodes is within a constant factor from the number of migrations generated by the optimal centralized algorithm with the same initial network state. Moreover, each node is migrated at most once.*
3. *$\mathcal{O}(1)$ rounds are sufficient to achieve constant smoothness.*

Proof. The first statement of the theorem follows easily from the algorithm due to the fact that each *short* node sends a message to a random destination which takes time $\mathcal{D}$ and then consecutively contacts the successors of the found node. This incurs additional communication cost of at most $r \cdot (\log n + \log u)$. Additionaly in each round each node changes the position of its marker and this operation also incurs communication cost $\mathcal{D}$.

The second one is guaranteed by the property that if a node tries to leave the network and join it somewhere else, it is certain that its predecessor is *short* and is not going to change its location. This assures that the predecessor will take over the job of our interval and it will not become *long*. Therefore, no *long* interval is ever created. Both our and the optimal algorithm have to cut each *long* interval into *middle* intervals. Let M and S be the upper thresholds for the lengths of a *middle* and *short* interval, respectively, and $l(I)$ the length of an arbitrary *long* interval. The optimal

```
short
    state := staying
    if (predecessor is short)
        with probability 1/2 change state to leaving
    if (state = leaving and predecessor.state = staying)
    {
        p := random(0..1)
        P := the node responsible for p
        contact consecutively the node P and its 6·log(u·n_i) successors on the ring
        if (a node R accepts)
            leave and rejoin in the middle of the interval of R
    }
    At any time, if any node contacts, reject.

middle
    At any time, if any node contacts, reject.

long
    wait for contacts
    if any node contacts, accept
```

Figure 1: The algorithm with respect to lengths of intervals (one round)

algorithm needs at least $\lceil l(I)/M \rceil$ cuts, wheras ours always cuts an interval in the middle and performs at most $2^{\lceil \log(l(I)/S) \rceil}$ cuts, which can be at most constant times larger because M/S is constant.

The statement that each server is migrated at most once follows from the reasoning below. A server is migrated only if its interval is *short*. Due to the gap between the upper threshold for *short* interval and lower threshold for *long* interval, after being migrated the server never takes responsibility for a *short* interval, so it will not be migrated again.

In order to prove the last statement of the theorem, we show the following two lemmas. The first one shows how many *short* intervals are willing to help during a constant number of rounds. The second one states how many helpful intervals are needed so that the algorithm succeeds in balancing the system.

Lemma 4 *For any constant $a \geq 0$, there exists a constant c, such that in c rounds at least $a \cdot n$ nodes are ready to migrate, w.h.p.*

Proof. As stated before, the length of each *middle* or *long* interval is at least $\frac{4}{n}$ and thus at most $\frac{1}{4} \cdot n$ intervals can be *middle* or *long*. Therefore, we have at least $\frac{3}{4} \cdot n$ nodes responsible for *short* intervals.

We number all the nodes in order of their position in the ring with numbers $0, \ldots, n-1$. For simplicity we assume that n is even, and divide the set of all nodes into $n/2$ pairs $P_i = (2i, 2i+1)$, where $i = 0, ..., \frac{n}{2} - 1$. Then there are at least $\frac{1}{2} \cdot n - \frac{1}{4} \cdot n = \frac{1}{4} \cdot n$ pairs P_i, which contain indexes of two *short* intervals. Since the first element of a pair is assigned state `staying` with probability at least $1/2$ and the second element state `leaving` with probability $1/2$, the probability that the second element is eager to migrate is at least $1/4$. For two different pairs P_i and P_j migrations of their second elements are independent. We stress here that this reasoning only only bounds the number of nodes able to migrate from below. For example, we do not consider first elements of pairs which also may migrate in some cases. Nevertheless, we are able to show that the number of migrating elements is large enough. Notice also that even if in one round many of the nodes migrate, it is still guaranteed that in each of the next rounds there will still exist at least $\frac{3}{4} \cdot n$ *short* intervals.

The above process stochastically dominates a Bernoulli process with $c \cdot n/4$ trials and single trial success probability $p = 1/4$. Let X be a random variable denoting the number of successes in the Bernoulli process. Then $E[X] = c \cdot n/16$ and we can use Chernoff bound to show that $X \geq a \cdot n$ with high probability if we only choose c large enough with respect to a. ■

In the following lemma we deal with cutting one *long* interval into *middle* intervals.

Lemma 5 *There exists a constant b such that for any* long *interval I, after $b \cdot n$ contacts are generated overall, the interval I will be cut into middle intervals, w.h.p.*

Proof. For the further analysis we will need that $l(I) \leq \frac{\log n}{n}$, therefore we first consider the case where $l(I) > \frac{\log n}{n}$. We would like to estimate the number of contacts that have to be generated in order to cut I into intervals of length at most $\frac{\log n}{n}$. We depict the process of cutting I on a binary tree. Let I be the root of this tree and its children the two intervals into which I is cut after it receives the first contact. The tree is built further in the same way and achieves its lowest level when its nodes have length s such that $\frac{1}{2} \cdot \frac{\log n}{n} \leq s \leq \frac{\log n}{n}$. The tree has height at most $\log n$. If a leaf gets $\log n$ contacts, it can use them to cover the whole path from itself to the root. Such covering is a witness that this interval will be separated from others. Thus, if each of the leaves gets $\log n$ contacts, interval I will be cut into intervals of length at most $\frac{\log n}{n}$.

Let b_1 be a sufficiently large constant and consider first $b_1 \cdot n$ contacts. We will bound the probability that one of the leaves gets at most $\log n$ of these contacts. Let X be a random variable depicting how many contacts fall into a leaf J. The probability that a contact hits a leaf is equal to the length of this leaf and the expected number of contacts that hit a leaf is $E[X] \geq b_1 \cdot \log n$. Chernoff bound guarantees that, if b_1 is large enough, the number of contacts is at least $\log n$, w.h.p.

There are at most n leaves in this tree, so each of them gets sufficiently many contacts with high probability. In the further phase we assume that all the intervals existing in the network are of length at most $\frac{\log n}{n}$.

Let J be any of such intervals. Consider the maximal possible set K of predecessors of J such that their total length is at most $2 \cdot \frac{\log n}{n}$. Maximality assures that $l(K) \geq \frac{\log n}{n}$. The upper bound on the length assures that even if the intervals belonging to K and J are cut ("are cut" in this context means "were cut", "are being cut" and/or "will be cut") into smallest possible pieces (of length $\frac{2}{n}$), their number does not exceed $6 \cdot \log n$. Therefore, if a contact hits some of them and is not needed by any of them, then it is forwarded to J and can reach its furthest end. We consider only the contacts that hit K. Some of them will be used by K and the rest will be forwarded to J.

Let b_2 be a constant and Y be a random variable denoting the number of contacts that fall into K in a process in which $b_2 \cdot n$ contacts are generated in the network. We want to show that, with high probability, Y is large enough, i.e. $Y \geq 2 \cdot n \cdot (l(J) + l(K))$. The expected value of Y can be estimated as $E[Y] = b_2 \cdot n \cdot l(K) \geq b_2 \cdot \log n$. Again, Chernoff bound guarantees that $Y \geq 6 \cdot \log n$, with high probability, if b_2 is large enough. This is sufficient to cut both K and J into *middle* intervals.

Now taking $b = b_1 + b_2$, finishes the proof of Lemma 5. ■

Combining Lemmas 4 and 5 with $a = b$, finishes the proof of Theorem 3. ■

3 Conclusion and future work

We have presented a distributed randomized scheme that continously rebalances the lengths of intervals of a Distributed Hash Table based on a ring topology. We proved that the scheme works with high probability and that its cost measured in the number of migrated nodes is comparable to the best possible.

Our scheme still has some deficiencies. The constants which emerge from the analysis are huge. We are convinced that these constants are much smaller than their bounds implied by the analysis. In the experimental evaluation one can play with at least a few parameters to see which configuration yields the best behaviour in practice. The first parameter is how well we approximate the number of servers n present in the network. Another is how many times a help-offer is forwarded before it is discarded. And the last one is the possibility to redefine the lengths of *short*, *middle* and *long* intervals. In future we plan to redesign the scheme so that we can approach the *smoothness* of $1 + \epsilon$ with additional cost of $1/\epsilon$ per operation as it is done in [9].

Another drawback at the moment is that the analysis demands that the algorithm is synchronized. This can probably be avoided with more careful analysis in the part where nodes with *short* intervals decide to stay or help. On the one hand, if a node tries to help, it blocks its predecessor for $\Theta(\log n)$ rounds. On the other, only one decicion is needed per $\Theta(\log n)$ steps.

Another issue omitted here is counting of nodes. Due to the space limitations we have decided to use the scheme proposed by Awerbuch and Scheideler in [2]. We developed another algorithm which is more compatible to our load balancing scheme. It inserts $\Delta \geq \log n$ markers per node and instead of evening the lengths of intervals it evens their weights defined as the number of markers contained in an interval. We can prove that such scheme also rebalances the whole system in constant number of rounds, w.h.p.

As mentioned in the introduction our scheme can be proven to use $\Theta(\log n)$ messages in half-life, provided that half-life is known. We omit the proof due to space limitations.

References

[1] M. Adler, E. Halperin, R. Karp, and V. Vazirani. A stochastic process on the hypercube with applications to peer-to-peer networks. In *Proc. of the 35th ACM Symp. on Theory of Computing (STOC)*, pages 575–584, June 2003.

[2] B. Awerbuch and C. Scheideler. Group spreading: A protocol for provably secure distributed name service. In *Proc. of the 31st Int. Colloquium on Automata, Languages, and Programming (ICALP)*, pages 183–195, July 2004.

[3] J. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 80–87, Feb. 2003.

[4] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in dynamic structured p2p systems. In *23rd Conference of the IEEE Communications Society (INFOCOM)*, Mar. 2004.

[5] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *Proc. of the 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 41–52, Aug. 2002.

[6] D. R. 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 *Proc. of the 29th ACM Symp. on Theory of Computing (STOC)*, pages 654–663, May 1997.

[7] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004.

[8] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proc. of the 16th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 36–43, June 2004.

[9] G. S. Manku. Balanced binary trees for id management and load balance in distributed hash tables. In *Proc. of the 23rd annual ACM symposium on Principles of Distributed Computing (PODC)*, pages 197–205, 2004.

[10] M. Mitzenmacher, A. W. Richa, and R. Sitaraman. The power of two random choices: A survey of techniques and results. In *Handbook of Randomized Computing. P. Pardalos, S.Rajasekaran, J.Rolim, and Eds. Kluwer*, 2000.

[11] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *Proc. of the 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 50–59, June 2003.

[12] M. Naor and U. Wieder. A simple fault tolerant distributed hash table. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2003.

[13] 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)*, Feb. 2003.

[14] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[15] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the ACM SIGCOMM*, pages 149–160, 2001.