Using Global Information for Load Balancing in DHTs*

Mikael Högqvist[1], Seif Haridi[2], Nico Kruber[1], Alexander Reinefeld[1], Thorsten Schütt[1]
[1]Zuse Institute Berlin
{hoegqvist, kruber, reinefeld, schuett}@zib.de
[2]Royal Institute of Technology (KTH)
seif@kth.se

Abstract

Distributed Hash Tables (DHT) with order-preserving hash functions require load balancing to ensure an even item-load over all nodes. While previous item-balancing algorithms only improve the load imbalance, we argue that due to the cost of moving items, the competing goal of minimizing the used network traffic must be addressed as well.

We aim to improve on existing algorithms by augmenting them with approximations of global knowledge, which can be distributed in a DHT with low cost using gossip mechanisms. In this paper we present initial simulation-based results from a decentralized balancing scheme extended with knowledge about the average node load. In addition, we discuss future work including a centralized auction-based algorithm that will be used as a benchmark.

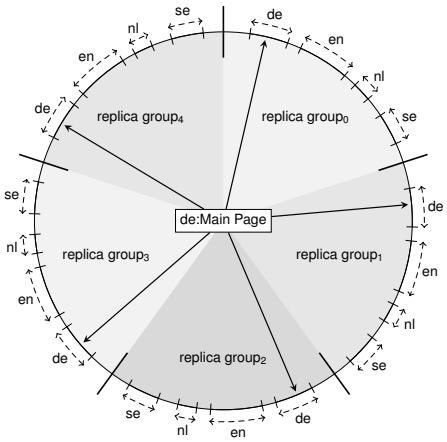

Figure 1. Geographic Load Balancing for Wikipedia.

1 Introduction

This work is motivated by research on self-managing distributed databases for use as a storage layer in large-scale Internet services. We envision that load balancing in such a system will not only consider node capacities, but can also be based on geographic location and application policies.

As an example, Wikipedia provides encyclopedias in different languages. Figure 1 shows how the Wikipedia articles and their respective replicas can be stored on a Distributed Hash Table (DHT) [12]. In such an application it is beneficial to host data nearby the users, i.e. in the geographic area where the language is used. We can use load balancing algorithms to implement location and application policies.

DHTs extend structured overlay networks (SON) with primitives for storing (key, value)-pairs and for retrieving the value associated with a key. Their functionality include support for both direct key lookups [13, 10] and range queries [11]. When a key is stored in a DHT with range queries, its location is decided using an order-preserving hash function. Depending on the distribution of the inserted keys the nodes in the DHT can quickly become unbalanced, which can lead to, for example, network congestion and unresponsive nodes.

Load balancing algorithms in DHTs focus on three different problems. First, when each item is hashed uniformly over the ID space, some nodes can have an $O(\log N)$ imbalance in terms of stored items [9, 6]. Second, when using order-preserving hash function [11], the items are mapped to the ID space such that they keep their original distribution. Therefore, for the system to be balanced, i.e. nodes storing an equal number of items, their node IDs must be distributed according to the key distribution [8, 4]. Third, independent of the item distribution, certain items can have much higher request rates than others. This is typically solved through caching, replication or by exploiting redundant network routes [3].

*This work was partly funded by the EU projects SELFMAN under grant IST-34084 and XtreemOS under grant IST-33576.

A common solution to the first problem is to maintain a set of virtual DHT nodes, or servers, at each physical machine. Virtual servers migrate between physical hosts to balance the system load [6, 13]. However, virtual servers have several issues such as increased churn when a physical host fails and increased state maintenance. In addition, in order-preserving DHTs, a single virtual server can still become overloaded when being responsible for a popular key range. In this paper, we are investigating solutions to the second problem, i.e. algorithms that are balancing the item-load at each node.

Since the network connecting the DHT nodes is the only shared resource, it is vital that DHT maintenance and tuning-algorithms use the network efficiently. This is especially the case for load balancing algorithms since their main operations trigger data movements. We aim to improve current algorithms by introducing approximations of global knowledge at each node, thereby helping them to take informed decisions in order to reduce data transfers. Examples of such information is the average node load or the network topology, which has already proved useful when balancing virtual servers [16]. Recent developments in gossiping for unstructured P2P networks and DHTs has shown that it is possible to obtain estimates of global properties with high confidence and low overhead [15, 14, 5].

In this paper, we argue for the benefits of introducing approximations of global knowledge to DHT load balancing algorithms with the goal of reducing the network utilization while maintaining a balanced system. To support this claim, we extend a well-known decentralized load balancing algorithm [8] to take the information about the average node load into account. The modified algorithm shows direct improvements on the overall items moved during balancing. We further argue for the use of centralized algorithms as a comparative benchmark.

2 Background

In this section we give an overview of current approaches for DHT load balancing with respect to virtual servers and item-balancing algorithms. This paper does not cover techniques for request balancing which is often solved through caching and/or replication.

Virtual Servers is a technique where each physical host maintains a set of virtual nodes. Load balancing is done by moving virtual servers, without changing their item range, from overloaded physical hosts to more lightly loaded hosts. The assignment of virtual nodes to physical hosts is typically performed by one or more directory nodes. A directory node periodically receives load information from random nodes in the system. When it has received load data from a sufficient amount of nodes it executes the load balancing algorithm [9, 6, 2]. An advantage of the virtual server scheme is that it does not require any changes to the DHT routing algorithm and allows for re-use of the join and leave overlay primitives.

An immediate issue with virtual servers is the increase of the routing table state maintained at each host. Godfrey et al. [7] introduce a scheme where a physical host maintains a set of virtual servers which have overlapping links in the routing table. With this placement restriction, a physical host only needs $\Theta(logN)$-links while hosting $\Theta(logN)$ virtual servers.

The above approaches use simple metrics for the cost of the load balancing operations, like the number of transferred items or bytes. A better cost metric could include the overall network utilization. In [16], Zhu et al. investigate how to minimize network usage by introducing proximity-aware load balancing algorithms for virtual servers. In [2], the assignment of virtual servers to physical hosts is modeled as an optimization problem which allows for an arbitrary cost function.

Another issue with virtual servers is that a physical host failure causes the hosted virtual nodes to fail as well. This increases the churn in the system and must be considered when selecting global parameters such as the replication factor.

Item-balancing Most of the research on load balancing in DHTs has focused on virtual servers. However, these approaches assume that items are uniformly distributed over the ID space using a hash function. In a DHT with an order-preserving hash function, a single virtual server can be overloaded if it is responsible for a popular key range. For example, when storing a dictionary, keys with the prefix "e" are more common than "w", resulting in the nodes storing items with prefix "e" being responsible for more items. The goal of item-balancing schemes is to adapt the location of the nodes in the system to correspond to the item distribution. This is performed using two operations, *jump* and *slide*. Jump transfers a node to an arbitrary ID in the system, while a slide operation only exchanges items with a node's direct neighbor.

Karger et al. [8] present a randomized item balancing scheme where each node contacts another random node periodically. If the load of the nodes differs by more than a factor $0 < \epsilon < \frac{1}{4}$, they share each others load by either jumping or sliding. Karger provides a theoretical analysis of the protocol, but does not evaluate the algorithms in an experimental or real-world setting. In addition, Karger's algorithm does not aim to minimize network traffic.

Ganesan et al. [4] use a reactive approach which triggers an algorithm when the node utilization super-cedes a threshold value. A node executing the algorithm first checks whether it should slide by comparing the load with

its neighbor's load. If this is not possible, it finds the least loaded node in the system and requests that it jumps to share the overloaded node's load. The least and most loaded node is located through a lookup to a separate DHT which stores all nodes sorted by their load.

We are basing our algorithms on the proactive approach presented by Karger, but aim to minimize the network utilization. This is achieved by making the algorithm aware of approximations of global parameters. While Ganesan stores the global knowledge in an additional DHT, we distribute this information through gossiping. This is more lightweight since it avoids the maintenance of another DHT and combines the strength of both structured and unstructured networks.

3 System model and problem definition

A DHT consist of N nodes, where each node has an ID in the range $[0, 1)$. This range wraps around at 1,0 and can be seen as a ring. A node, n_i has a *successor*-pointer to the next node in clockwise direction, n_{i+1}, and a *predecessor*-pointer to the first counter-clockwise node, n_{i-1}. The node with the largest ID has the node with the lowest ID as successor. Thus, the nodes and their pointers create a double linked list where the first and last node are linked.

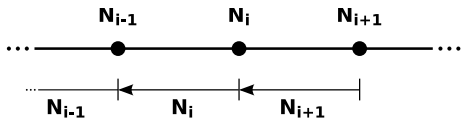

Figure 2. A node N_i with successor and predecessor and their responsibilities.

Each node in the DHT stores a subset of items, $I(n_i)$, where each item has a key in the range $[0, 1)$ and a uniform weight of one. A node n_i is *responsible* for a key iff it falls within the node's key range $(n_{i-1}, n_i]$. Each node has a load $l(n_i)$ indicating the number of stored data items. Figure 2 shows three nodes and their respective responsibilities.

A node is balanced when it is neither *underloaded* nor *overloaded* relative to any other node in the system times a factor ϵ [8]. That is, when $l(n_i) < \epsilon l(n_j)$, $l(n_j)$ is overloaded compared to n_i and n_i is underloaded compared to n_j. The goal of the load-balancing algorithm is to make all nodes balanced. ϵ is a system-defined parameter with values between 0 and $\frac{1}{4}$.

In order to change the load of nodes in the system, two types of operations are used: jump and slide.

Jump allows a node to move to an arbitrary position in the ID space. A jumping node n_i first leaves its current position and re-joins at its new location, ID_k, with n_j as its successor. Data is moved two times. First, the items in the range $(n_{i-1}, n_i]$ are transferred to n_{i+1}. Second, when n_i joins at ID_k, all data in the range $(n_{j-1}, \text{ID}_k]$ is transferred from n_j to n_i.

Slide is a specialized form of jumping where n_i moves to an ID in the range (n_i, n_j), assuming that the overloaded node n_j is n_{i+1}. Since a node does not need to leave and re-join the system, which results in extra data transfer, sliding is always preferred over jumping.

We define a load-balanced configuration as a system state where all nodes are balanced. The maximum load in a configuration, C_i is denoted by $l_{max}(C_i)$, while the minimum load is $l_{min}(C_i)$, respectively. We use the standard deviation of a configuration, $\sigma(C_i)$, as a measure to indicate its imbalance. A jump or slide changes a configuration C_i to a new configuration C_{i+1}. For a jump or a slide operation performed by any node the algorithm must meet the following properties for the load to converge towards a balanced configuration.

$$l_{max}(C_i) \geq l_{max}(C_{i+1}) \quad (1)$$

$$l_{min}(C_i) \leq l_{min}(C_{i+1}) \quad (2)$$

$$\sigma(C_i) > \sigma(C_{i+1}) \quad (3)$$

Following these properties, an algorithm will reach a balanced configuration after a finite number of iterations.

Problem definitions. The load balancing problem can be summarized as follows: given a configuration C_0 with a set of nodes $\mathbf{N}$ and items $\mathbf{I}$, where each item $i \in \mathbf{I}$ is assigned to a responsible node, find a configuration C_b that only contains balanced nodes using the operations jump and slide. A solution to the load balancing problem is a sequence of operations transforming C_0 to C_b.

In addition to the load balancing problem, we search for a solution that minimizes the data movement cost of the transition from C_0 to C_b. That is, given a set of solutions, $\mathbf{S}$, find a solution S_i with minimal cost. The cost-function is $cost(op)$, where op is either a $slide$ or $jump$ operation. The cost-function can be chosen arbitrarily, but is typically based on the number of bytes moved or the network utilization.

4 Decentralized Algorithms

Unlike a centralized algorithm, a decentralized algorithm can only use the information locally available at each node. We modify Karger's randomized item-balancing algorithm to work with different globally approximated parameters. In this paper we use the system's average load.

Global information is, by definition, not available in peer-to-peer systems, unless aggregation algorithms are employed. However, by using gossiping techniques such as Vicinity and Cyclon [15, 14] or DHT gossip [5] it is possible to get a good approximation locally at each node of a parameter's value with low network traffic overhead.

Karger's Algorithm. In order to reach a load-balanced configuration, we rely on the heuristics introduced for Karger's item balancing algorithm [8]. Expressed in our notation, a load-balance operation is only performed between any pair of nodes n_i, n_j, iff $l(n_i) < \epsilon l(n_j)$ or $l(n_j) < \epsilon l(n_i)$, $0 < \epsilon < \frac{1}{4}$. When these restrictions are satisfied, the following cases are possible (assuming $l(n_i) > l(n_j)$).

Case 1, $i = j + 1$ n_i is the successor of n_j. Slide n_j towards n_i, letting n_j take responsibility for $\frac{l(n_i)-l(n_j)}{2}$ of n_i's items.

Case 2, $i \neq j + 1$ If $l(n_{j+1}) > l(n_i)$, set $i = j + 1$ and go to case 1. That is, when the load of n_j's successor is larger then the load of n_i, slide n_j towards the overloaded node n_{j+1}. Otherwise, n_j jumps to a position in the range (n_{i-1}, n_i), taking half of $l(n_i)$.

Modified Karger. Karger's randomized algorithm is based on two decisions. (1) Which nodes should balance? (2) Should they use jump or slide? The new location of a node performing a jump or slide is calculated such that the load is shared evenly by the two participating nodes. We want to show that global information can be used to reduce the number of transferred items, which indirectly impacts the network usage. Therefore, we introduce a heuristics based on the average load. Our modification is a restriction on the position a node takes after an operation. Instead of as in Karger, sharing the load evenly, we ensure that an underloaded node never takes more than the average load, L_{avg}. This effectively limits the amount of unnecessary data item transfers.

The described strategy has the biggest advantage when a node is overloaded relative to another node and it's load is much larger than the average load. Figure 3 shows a scenario where a node N_{i+1} has a load much larger than the average load, i.e. $l(N_{i+1}) > 3L_{avg}$. Let two nodes, A and B, execute the load balancing algorithm in that order. In Karger, assuming that A balances with N_{i+1}, it would first take more than $1.5L_{avg}$ load. If B then chooses to balance with node A, which is possible if A is still overloaded in relation to B. Then the data in A's range is transferred twice, first from N_{i+1} to A and then from A to B. With our modified version, since node A would take at most L_{avg} load from N_{i+1}, the probability that B balances with A is lower as well as the transferred data items if A decides to balance with B.

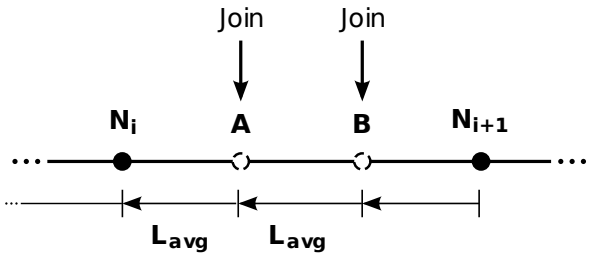

Figure 3. Two consecutive slots being filled by joining nodes, taking at most L_{avg} load.

5 Evaluation

We simulated Karger's algorithm and a version with knowledge of the systems average load. The effect of knowing the average load can be seen in Figures 4 and 5.

Experiments. The experiments are performed in a discrete time simulator where a single operation represents a step in time. The system contains 100 nodes, and the items are distributed such that the first 90 nodes have one item and the remaining ten nodes have 10000, 20000, ..., 90000 items, respectively. We measure the number of moved items as operation cost and the standard deviation is used to indicate the load imbalance of the system.

Figure 4 shows the sum of moved items for the operations necessary to go from the initial configuration to a load balanced configuration. Increasing ϵ values, between $0 < \epsilon < 0.25$ as suggested by Karger, shows a linear increase in the balance cost. Interestingly, the comparison between Karger and the modified Karger shows that in many cases the latter moves half as many items to reach a balanced configuration.

In Figure 5 we set $\epsilon = 0.21$ and study how each operation influences the load imbalance. The x-axis represents the aggregated number of moved items for each round and the y-axis is the standard deviation for the current configuration. The simulation is continued until no further balance operations can be performed. Our main conclusion from this experiment is that the modified Karger decreases the load imbalance of the system faster, even though it moves less items than the basic Karger.

6 Outlook

In this section we discuss the implications of load balancing algorithms for other DHT services. We also outline

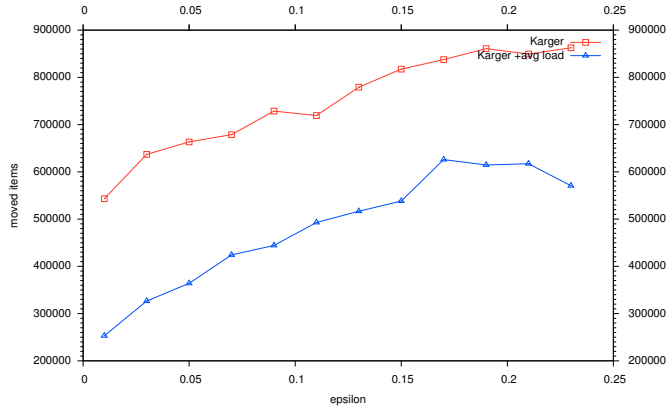

Figure 4. Number of moved items with epsilon between 0 and 0.25.

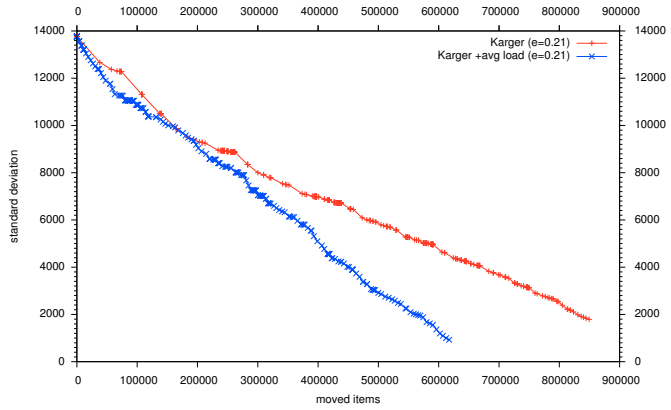

Figure 5. Load imbalance as a function of the number of moved items.

our approach to a centralized algorithm which we plan to used as a comparative benchmark for the decentralized algorithms.

Additional Global Estimates. In the evaluation section, we showed that using the average load can have an impact on the load balancing performance. In addition to the average load, we are interested in evaluating the following parameters.

Standard Deviation In section 3 we used the standard deviation as an invariant for the progress of an algorithm. If each node has knowledge of this value they could try to minimize it for each balancing operation they perform.

Location Proximity-information allows a node which will transfer load to select a target node which minimizes the network utilization [16].

Over- and Underloaded nodes A list of the k most overloaded and most underloaded nodes. These lists can be used for example in the Karger-algorithm to improve the convergence rate.

Implications of Load Balancing on a DHT. As argued throughout this paper, load balancing is an important part in an efficient and self-tuning DHT. However, the load balancing algorithms must work seamlessly together with other components in a DHT-based storage layer such as replication and transactions.

The jump and slide primitives are using the basic join and leave operations from the overlay. Since these operations are triggered by the load balancing algorithms, the balancing itself incurs extra churn in the system. Therefore, it is important that, for example, the systems replication factor is chosen with this in mind. Tuning the load balancing to work at a rate acceptable for the system is an important trade-off that needs to be evaluated for a working system.

A Centralized Auction-based Algorithm. In a centralized algorithm the global state of the system is known. A centralized algorithm can be used as a reference benchmark for decentralized algorithms. We aim to base our centralized algorithm on an auction algorithm [1] where overloaded and underloaded are matched to find an optimal assignment.

An auction algorithm finds an optimal one-to-one assignment of persons to objects in polynomial time. The assignment depends on the cost of the object and the benefit of the person being assigned to the object. For the load balancing problem this is analogous to finding a lowest cost match between underloaded and overloaded nodes.

More formally, each person i has a benefit a_{ij} of selecting an object j with price p_j. The net value for a person i of choosing object j is $a_{ij} - p_j$. The goal of the auction is to find an assignment where every persons find an object which maximizes the total net value. Thus, an auction is finished when the equilibrium $a_{ij} - p_j = \max_{j \in Objects}(a_{ij} - p_j)$ is reached.

Each iteration of the algorithm consists of a bidding phase followed by an assignment phase. During the bidding phase, each person finds an object resulting in maximum net value after which it computes a bidding increment. The value of the bidding increment is used after the assignment phase to increase the price of the object. In the assignment phase the persons with the highest bids are assigned to the respective objects. When all people are assigned to an object the algorithm terminates. This also means that the equilibrium has been satisfied.

An advantage of the auction algorithm is that the benefit function and the object price can be chosen arbitrarily. This allows us to explore more complicated costs than e.g. moved data items. Furthermore, the order of the load balancing operations slide and join are affecting the total price of the load balancing process. Due to the apparent advantages in computational complexity of the auction algorithm approach, we are actively investigating an appropriate cost-function which can include proximity information and the order of operations.

7 Conclusion

We showed that it is possible to reduce the cost of load balancing by introducing simple heuristics and knowledge about basic global parameters. We plan to continue this work by evaluating the effects of more properties such as the network topology. In addition, a centralized algorithm can give the optimal cost for balancing a given configuration. This can be used as a reference to evaluating the performance of the decentralized algorithms.

References

[1] D. P. Bertsekas. *Network Optimization: Continuous and Discrete Models (Optimization, Computation, and Control)*. Athena Scientific, 1998.

[2] C. Chen and K.-C. Tsai. The server reassignment problem for load balancing in structured p2p systems. *IEEE Trans. Parallel Distrib. Syst.*, 19(2):234–246, 2008.

[3] A. Datta, R. Schmidt, and K. Aberer. Query-load balancing in structured overlays. In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID'07)*, 2007.

[4] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. In *VLDB*, pages 444–455. Morgan Kaufmann, 2004.

[5] A. Ghodsi, S. Haridi, and H. Weatherspoon. Exploiting the synergy between gossiping and structured overlays. *Operating Systems Review*, 41(5):61–66, 2007.

[6] B. Godfrey, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica. Load balancing in dynamic structured p2p systems. In *INFOCOM*, 2004.

[7] B. Godfrey and I. Stoica. Heterogeneity and load balance in distributed hash tables. In *INFOCOM*, pages 596–606. IEEE, 2005.

[8] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *IPTPS*, volume 3279 of *Lecture Notes in Computer Science*, pages 131–140. Springer, 2004.

[9] A. Rao, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica. Load balancing in structured p2p systems. In *IPTPS*, volume 2735 of *Lecture Notes in Computer Science*, pages 68–79. Springer, 2003.

[10] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, 2001.

[11] T. Schütt, F. Schintke, and A. Reinefeld. Structured overlay without consistent hashing: Empirical results. In *CCGRID*. IEEE Computer Society, 2006.

[12] T. Schütt, F. Schintke, and A. Reinefeld. Scalaris: Reliable transactional P2P key/value store. In *ACM SIGPLAN Erlang Workshop*, 2008.

[13] 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 *SIGCOMM*, pages 149–160, 2001.

[14] S. Voulgaris, D. Gavidia, and M. van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *J. Network Syst. Manage.*, 13(2), 2005.

[15] S. Voulgaris, M. van Steen, and K. Iwanicki. Proactive gossip-based management of semantic overlay networks. *Concurrency and Computation: Practice and Experience*, 19(17):2299–2311, 2007.

[16] Y. Zhu and Y. Hu. Efficient, proximity-aware load balancing for dht-based p2p systems. *IEEE Trans. Parallel Distrib. Syst.*, 16(4):349–361, 2005.