Load Balancing in P2P Networks: Using Statistics to Fight Data and Execution Skew

Daniel Warneke
Technische Universität Berlin, Berlin, Germany
Email: daniel.warneke@tu-berlin.de

Christian Dannewitz
University of Paderborn, Paderborn, Germany
Email: christian.dannewitz@uni-paderborn.de

Abstract—
In recent years, structured peer-to-peer (P2P) have gained an important role in the design of large-scale distributed systems. However, due to their strict data placement rules, they are often prone to three main load imbalances, i.e., range, data, and execution skew. Many of today's load balancing algorithms focus only on range skew and assume the network data rate to be the bottleneck. In applications that focus on distributed request processing, those assumptions are not valid as messages are typically small but can produce significant load at the application level. Examples of such applications are complex name resolution mechanisms that, e.g., involve security checks, or multi-dimensional search. Here, data skew and execution skew are most important and the system performance is limited by the number of application requests a peer can process.

To provide a solution for such scenarios, we have developed a new load balancing algorithm which is based on ID management. Our algorithm collects statistics of overlay link usage during normal operation and uses this information to provide suitable IDs to joining peers. Without using regular maintenance messages, it improves the rate of successfully answered requests by a factor of up to 3 in typical scenarios. We have evaluated the algorithm via extensive simulation that also includes scenarios with churn and heterogeneous peers. This work presents the first load balancing algorithm that can handle all three types of skew in scenarios that focus on processed application requests as the bottleneck.

Index Terms— **load balancing, ID management, data skew, execution skew, range skew, structured P2P networks**

I. Introduction

Deterministically assigning data items to responsible peers is key to efficient lookups in structured P2P networks. However, it also makes these network prone to skewed load distribution. Even under the assumption that all peers participating in the network have comparable resources, load imbalance can be caused by one or possibly a combination of the following three issues:

Range skew is the ratio between the size of the smallest and the largest partition of the identifier (ID) space one peer has to manage [1]. Many popular structured P2P networks [2]–[4] assign IDs to their peers (pseudo) randomly with a uniform distribution. This approach is easy to implement, however, it yields a range skew of $O(\log N)$ with N being the number of peers [5]. For this reason, particular peers might be responsible for more data items than others even if the items are distributed uniformly in the ID space.

Data skew refers to an uneven distribution of data items across the partitions of peers [6]. Data skew typically occurs in situations when uniform hashing of data items cannot be applied. E.g., applications that require to perform range queries in the P2P network do not use uniform hash functions since these functions destroy the locality of the data items and the queries could not be evaluated in an efficient manner. Instead, special locality-preserving hash functions can guarantee to map adjacent data items to adjacent IDs but are likely to provide a much worse data distribution. A particular peer may therefore be responsible for considerably more data items than others and, consequently, receive more requests.

Execution skew describes non-uniform data access across the partitions of peers [6]. For Web traffic [7] and multiple P2P applications [8], the popularity of data items follows a Zipf-like distribution; peers in charge of popular data items receive significantly more requests than others.

While many previous load balancing mechanisms evaluate a data dissemination scenario in P2P networks and, therefore, consider the network data rate to be the bottleneck [9], [10], we assume a different scenario where processing a request at the application level, not its routing through the overlay network, limits the performance of the overall system. This assumption is reasonable when request messages are small and cheap to forward measured by today's commodity hardware/network connections (as typical for most applications based on structured P2P networks) but answering the requests involves computationally intensive operations or even hard disk accesses. A prominent example of this type are information-centric

This paper is based on "Statistics-based ID Management for Load Balancing in Structured P2P Networks" by D. Warneke and C. Dannewitz, which appeared in the Proceedings of the 34th IEEE Local Computer Networks Conference (LCN), Zurich, Switzerland, October 2009.

doi:10.4304/jait.2.1.40-49

network architectures [11]–[13] that are becoming an important option for the future Internet architecture. Many of these approaches are based on naming data with flat names that require some kind of name resolution service to translate flat names into locators. distributed hash tables (DHTs) are good candidates for implementing such a name resolution service. In such a scenario, the DHT system has to handle many small name resolution requests that typically involve some kind of computational security check [14], dictionary lookup, and locator selection. The recent example of the Soccer World Championships 2010 where a set of servers performing geoblocking and locator (i.e. streaming server) selection turned out to be the bottleneck of the overall system of a large TV provider illustrates that such small requests can produce significant load problems [15]. Additional examples where the bottleneck is request processing include multi-dimensional search [16], [17] and DHTs that include complex features like public/private key authentication [18]. In all these cases the cost for forwarding a request to the next overlay hop is negligibly small compared to the cost that incurs at the destination for processing it.

Our proposed algorithm is based on *ID management* [1], a subclass of load balancing algorithms that uses the network's natural churn to respond to skewed load. ID management algorithms aim at proposing IDs to newly arriving peers in a way that is helpful to reduce load imbalance in the P2P network. These algorithms are characterized by a very small messaging and processing overhead compared to classical approaches.

Unfortunately, most ID management algorithms are limited to range skew and are, therefore, not applicable in scenarios where load imbalance may also result from additional data or execution skew [9]. Therefore, we propose a new ID management approach that can handle all three types of skews. Our approach observes message flows during network operation and uses the gathered information to identify popular regions of the ID space. Compared to existing algorithms, our approach is distinguished by the following characteristics:

- Our approach considers the actual load of the peers for ID management. As a result, it can deal with range, data, and execution skew, even in presence of heterogeneous peers. In particular, our algorithm is the first ID management algorithm that is able to respond to data skew.
- Our approach only causes an overhead of $O(R)$ messages per peer arrival with R being the routing complexity required by the underlying P2P routing protocol to deliver a message (e.g. $R = \log N$). No other additional messages are required.
- Our approach is applicable to any structured P2P network that routes messages via multiple overlay hops and uses IDs of fixed length. This is important because we use a normalization algorithm that has to be able to calculate the portion of the overall ID space that a certain overlay link bridges. Most networks like Chord [2] and Pastry [3] fulfill this requirement. If desired, our load balancing method can be complemented by other existing load balancing mechanisms such as virtual servers [19], ID space adjustment [6], or caching schemes [10].

This paper is an extended version of [20]. It provides a more thorough discussion of the presented ID management algorithm and related approaches, presents source code for the main algorithm aspects, and features additional experimental results including an evaluation of the system behavior in presence of churn. The rest of the paper is structured as follows: In the next section, we discuss related work. In Section III, we describe our load balancing mechanism in detail. To evaluate our approach, we have conducted numerous simulations that are discussed in Section IV.

II. Related Work

In recent years, many load balancing schemes have been developed for structured P2P networks. A very early idea has been virtual servers, i.e. multiple peers run on the same physical node. First approaches were designed to mitigate range skew [2] alone; more advanced algorithms (e.g. [21], [22]) also tackle data and execution skew. However, to establish a rendez-vous between overloaded and less-loaded peers, these algorithms require recurring message exchange. Schemes that do not require additional communication typically start/destroy virtual servers based on local knowledge [23] and are likely to increase churn. In general, virtual servers increase the network diameter, making them a doubtful choice when latency matters.

The load balancing scheme of Ganesan et al. [6] can distribute load among its neighbors by altering partition sizes during operation. The paper has a strong theoretical foundation but omits a reasonable churn model. Simulations with churn [9] indicate that frequent peer arrival/departure can significantly limit the performance of this approach.

Many previous papers on ID management also have a theoretical background and focus on range skew. E.g., Noar and Wieder [24] as well as Abraham et al. [25] have proposed an algorithm where an arriving peer randomly chooses $\Theta(N)$ IDs from the ID space, contacts the responsible peers and joins within the largest partition one of the contacted peers is responsible for. Both approaches are able to limit range skew to a constant factor. Further approaches have been able to consider peer departure [19], further decrease the range skew factor [1], or incorporate heterogeneous peers [5]. All these ID management algorithms assume that a peer's load is proportional to the partition size it manages. However, this assumption is wrong in scenarios with data and execution skew.

A more practical approach is taken by Ledlie and Seltzer [9]. Similar to our scheme, their *k-Choices* algorithm considers the actual load of a peer for load balancing. Upon arrival a new peer contacts several distinct IDs (i.e., the peers responsible for them) and starts virtual servers within the partition of those peers which exceed their

target workload the most. The authors demonstrate a good load balancing performance for a Zipf-like popularity distribution of data items. Their emphasis on trustworthy, reproducible IDs for every peer, however, makes it hard to apply the algorithm in scenarios with skewed data distribution because peers are not allowed to join at arbitrary positions.

The work by Bianchi et al. [10] is related to our approach since the authors also propose to observe the utilization of overlay links for load balancing. Unlike ours, their algorithm is limited to overlay routing protocols that offer neighbor selection flexibility, (e.g., Pastry [3]). Based on this flexibility, they use the collected information to distribute the load of message forwarding among peers from the same region (i.e., with the same ID prefix). To deal with request load, the authors discuss a caching scheme on top of the routing layer.

III. ID Management Algorithm

The ID management algorithm we present in the following is a greedy distributed algorithm that directs joining peers to highly-frequented regions of the ID space. It is based on the idea that peers responsible for these regions are most likely to be overloaded. To identify these highly-frequented regions, our algorithm collects statistics on the utilization of the peers' overlay links during the regular operation of the P2P network, i.e., without generating additional messaging overhead. At first, we will discuss the collection of these statistics, and later on demonstrate how to leverage them for load balancing.

A. Statistic collection

Collecting statistics on the utilization of a peer's overlay links is straightforward and can be integrated in the regular overlay routing procedure. Fig. 1 illustrates the method ROUTE, which is in charge of either delivering an incoming message *msg* to an upper tier if the current peer is responsible for the destination ID *destId* or to forward it closer to its destination, otherwise.

ROUTE(msg, $destId$)
1: $j \Leftarrow$FINDNEXTHOP($destId$)
2: **if** $j \neq NULL$ **then**
3: INCREMENT($RT[j].count$)
4: FORWARD($RT[j].ip$, msg, $destId$)
5: **else**
6: DELIVER(msg, $destId$)
7: **end if**

Figure 1. Schematic overlay routing with statistics collection.

Each peer maintains its overlay links in a Routing Table RT with m entries. In addition to the standard information for each overlay link j that is typically stored in the Routing Table (j's ID ($RT[j].ID$) and IP address ($RT[j].ip$)), we maintain a counter for each overlay link j (*Link Utilization Counter*) that stores how often it was used ($RT[j].count$).

The method ROUTE first determines the next hop for message msg via the method FINDNEXTHOP; it encapsulates the concrete routing strategy depending on the specific overlay routing algorithm. FINDNEXTHOP returns an index j identifying the routing entry in RT to be used to forward msg. In case the peer itself is responsible for $destId$, FINDNEXTHOP returns NULL (line 2). Next, the method increments $RT[j].count$. Incrementation can be weighted depending on the message type. Finally, the message is forwarded to the next hop $RT[j].ip$ or delivered to the upper tier.

Periodically, each peer calculates its so-called *Join Link Table* (JT). The Join Link Table contains all Routing Table entries, ordered by link utilization. It is later consulted to direct joining peers towards highly utilized regions of the ID space. Computing JT mainly involves normalizing the Link Utilization Counters in RT according to the range that each overlay link covers in the ID space. For example, in Chord, the range that each overlay link in the Routing Table covers is increasing exponentially, i.e., the routing entry $RT[m-1]$ is used for all messages in an ID range of approximately size $\frac{|D|}{2}$, whereas the entry $RT[m-2]$ only covers an ID range of size $\frac{|D|}{4}$. Without normalization, the utilization of overlay links would (incorrectly) appear to be highly skewed even in scenarios with uniform load distribution. The concrete normalization process depends on the routing algorithm. Fig. 2 shows the normalization algorithm we used for the implementation of our ID management algorithm based on the overlay network Chord.

NORMALIZECHORD(RT)
1: **for** $1 \leq j < RT.size$ **do**
2: $r \Leftarrow$ GETRANGECHORD($RT[j], RT[j+1]$)
3: $RT[j].count \Leftarrow \frac{RT[j].count}{r}$
4: **end for**

GETRANGECHORD($link1, link2$)
1: **if** $link1 \leq link2$ **then**
2: **return** $link2 - link1$
3: **else**
4: **return** $|D| - (link1 - link2)$
5: **end if**

Figure 2. Normalization of overlay link utilization for the Chord.

For Chord the normalization process is relatively simple as a result of its straightforward routing algorithm. For other overlay networks, the normalization algorithm can become more complex. E.g., in P2P networks which are based on prefix routing [26] like Pastry [3] or Tapestry [27], the utilization of each overlay link must be normalized according to the length of the common prefix between the link and the destination ID. However, every structured multi-hop P2P network that we know of uses a deterministic scheme to reduce the distance between the current peer's ID and the destination ID on every overlay hop. As long as the IDs of the ID space $|D|$

have a fixed length, it is possible to determine the range r that a specific overlay link bridges. This value can then be used to normalize the number of messages routed via this link.

Note that for calculating the Join Link Table JT only the *ordering* of the Link Utilization Counters matters, not their concrete value, i.e., we only use the Link Utilization Counter of each overlay link to order the links by their relative utilization. Apart from that purpose the concrete value of the Link Utilization Counter does not matter. Therefore, we are able to construct a valid Join Link Table independent of the specific system load a peer actually experiences.

Besides RT and JT, each peer maintains a third table, the so-called *Destination Table* DT. It keeps track of the peer's workload W. As indicated in the introduction, we focus on applications where the performance of the overall system is limited by the number of requests a peer can process at the application level. We assume the cost for forwarding requests towards their destinations to be negligible small in comparison. Hence, we define workload as the number of application requests a peer receives to process per time unit. The number of requests a peer can process during that time is limited by its capacity C. Incoming requests beyond that capacity can be detected but are discarded. To cope with temporary workload fluctuations, a peer may want to operate below its capacity and strive for a target workload T. The concrete role of C and T is explained in the next subsection.

The Destination Table DT partitions the ID range a peer is responsible for in k evenly sized sub ranges and keeps track of how popular these sub ranges are. DT can be considered a histogram with k bins. It counts the requests arriving at the peer that fall into each of the k bins per time unit and periodically calculates a moving average. Those values can then be used to estimate how changing the peer's ID range would affect its current workload. This allows overloaded peers to suggest IDs to joining peer's in a very fine-grained way as illustrated in the next subsection. Each time the responsibility range of peer changes, e.g., because an adjacent peer is joining or leaving, the table is reset.

B. Choosing IDs for joining peers

Having explained the statistics collection on each peer, we now describe how joining peers can leverage these statistics to join at IDs beneficial for other overloaded peers. Our goal is to make a new peer j join within the partition of an overloaded peer i in a way that brings i as close as possible to its target workload T_i.

Before peer j joins the P2P network, it sends an *ID Request* to its bootstrap peer. The ID Request initially contains T_j and a time-to-live (TTL) field which states how often the ID Request is allowed to be passed on to another peer. It is decreased on every overlay hop.

Upon reception of an ID Request, a peer i runs the algorithm PROCESSIDREQUEST as illustrated in Fig. 3. First, the request's TTL field is decreased by one (line 1). If peer i is overloaded (line 2), it uses the information stored in DT_i about the workload distribution within its ID range to suggest an ID for the joining peer j. The ID is chosen so that it brings peer i as close as possible to its target workload T_i. The concrete implementation of a corresponding algorithm depends on the overlay routing protocol and the way ID ranges are assigned to peers. For Chord, the ID range of peer i is divided in two parts so that the load of the remaining part for peer i matches i's target workload. The workload of the other part estimates the expected workload of j (line 4).

PROCESSIDREQUEST(msg)

1: $msg.TTL \Leftarrow msg.TTL - 1$
2: **if** $W_i > T_i$ **then**
3: $sID \Leftarrow$ SUGGESTJOINID
4: $eW \Leftarrow$ GETEXPECTEDWORKLOAD
5: **if** $eW > msg.expWorkload$ **then**
6: $msg.suggestedID \Leftarrow sID$
7: $msg.expWorkload \Leftarrow eW$
8: **end if**
9: **end if**
10: **if** $msg.TTL = 0$ OR $msg.expectedWorkload > msg.targetWorkload$ **then**
11: SENDRESPONSE(msg)
12: **return**
13: **end if**
14: $joinLink \Leftarrow JT_i$[RANDOM$(0, s-1)$]
15: FORWARD($joinLink.ip, msg$)

Figure 3. Processing of an incoming ID Request.

If this expected workload is higher than the workload estimated on one of the request's previous hops, peer i overwrites the so far suggested join ID and respective workload with its own suggestion (line 6,7). That way, the new peer j will eventually unburden the peer with the highest load on the ID Request's path.

If the message's TTL has expired or the expected workload for j already exceeds its target workload, peer i immediately sends a response message to peer j and does not forward the request any longer. Otherwise, peer i passes the ID Request on to a peer it assumes to be potentially overloaded. This is done by randomly choosing a so called join link from the first s entries of JT_i which represent the s most utilized overlay links of peer i. Without randomization, the algorithm tends to direct ID Requests to fixed positions of the ID space, e.g., the ID of the most popular keyword. Choosing this link randomly from a preselected set of suitable links helps our algorithm to respond to multiple distinct peaks in the workload distribution.

Finally, when peer j receives a response to its initial ID Request, it joins the P2P network with the suggested ID. If the suggested ID is NULL, i.e., no overloaded peer has been discovered, it joins with a random ID. Fig. 4 illustrates the join process using our algorithm.

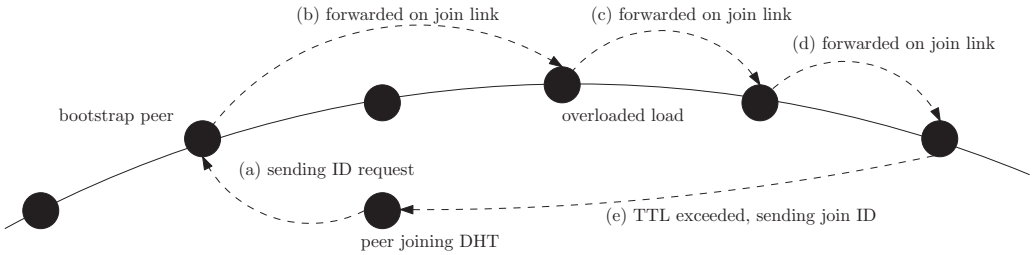

Figure 4. The new peer joining the DHT first sends an ID Request to its bootstrap peer (a). Each peer in the DHT forwards the request via its join link (b, c, d). If the request passes an overloaded peer, the respective peer suggests an ID for the new peer to join (c). Finally, when the request's hop count exceeds the specified TTL, the determined join ID is sent back to the new peer (e).

IV. Evaluation

In this section, we want to evaluate the performance of our ID management algorithm through various simulations based on the Oversim P2P framework [28]. This paper wants to illustrate that a significantly improved load balancing can be achieved via a clever placement of joining nodes without much overhead compared to the original DHT system (e.g. Chord). Therefore, the original Chord system is the main comparison case for use. Our approach does not intend to replace other load balancing mechanisms. Instead, it can be combined with such approaches to further improve load balancing.

We begin with describing our simulation model and the workload scenarios we consider. Afterwards, we discuss the experiments' results.

A. Simulation Model

In our simulation we set up a structured P2P network based on the Chord overlay routing protocol, initially with an average number of $N = 1024$ peers. Each peer repetitively sends requests to specific IDs of the ID space D ($|D| = 2^{160}$). As primary metric we observe the impact of our ID management algorithm on the ratio of successfully answered to totally issued requests per time unit (the *success rate*). The distribution and frequency of those requests depends on the workload model. We have defined three workload models which are predominately intended to capture the effects of data and execution skew:

Keyword workload model: Our first workload model represents applications that make use of keyword search. It includes execution skew as a result of varying keyword popularity. The keywords themselves are represented by 1000000 uniformly randomly generated IDs from the ID space D. Based on previous studies on keyword popularity in P2P networks [8], [29], we approximate the keyword popularity by a Zipf distribution and vary the skew α from 0.1 to 1.0 during our experiments. α is 0.4 by default unless explicitly stated otherwise. Please note that several P2P load models differ from a Zipf distribution, especially because of their "fetch-at-most-once" behavior [30]. However, based on our use cases described in the introduction, a Zipf-like Web traffic load model [7] is more realistic here.

Gaussian workload model: Our second workload model intends to capture the effects of data skew. Data skew is likely to appear when uniform hashing cannot be applied, e.g., for applications that offer support for partial keyword, wildcard, or range queries. As a result, the data distribution may center around specific regions of the ID space. Since we are not aware of any empirically confirmed workload model for data skew, we have taken a generic approach. We have generated 1000000 distinct IDs from a Gaussian distribution with mean $\frac{|D|}{2} = 2^{159}$ and a variance of 2^v. v is then varied in the course of our experiments, by default it is 157. The resulting data distribution is depicted in Fig. 5. The popularity of the data items is uniformly distributed.

GPS workload model: The initial idea of our ID management algorithm has been motivated by a P2P application where users can attach digital information to geographical locations using its GPS coordinates. Location detection may be ambiguous or inaccurate, so the underlying P2P network must support range queries. We have mapped the longitude value l of a GPS coordinate to an ID id of the ID space D with the locality-preserving hash function $id = |D| \cdot \frac{l+180}{360}$ [1]. To approximate the distribution of IDs in our system on a global scale we generated 1000000 IDs according to earth's population density [31] (Fig. 6). In addition, we assume that some GPS coordinates (and consequently IDs) are more popular than others depending on the population density, e.g., tourist attractions in densely populated regions. Thus, the model represents both data and execution skew.

We start with a homogeneous scenario: Responding to an application request takes 0.1 seconds, so each peer in the P2P network has a processing capacity of $C = 10$ requests/second. Any request that arrives while another one is currently processed is appended to a drop-tail queue with a maximum length of 20. Further incoming requests are immediately dropped. Peers detect dropped application requests by a timeout and mark them as unsuccessful. No retransmission is used. The target workload T of each peer is always set to 90% of its capacity.

In accordance with our workload model where the number of requests a peer can process at the application level is the bottleneck, the network in the simulation is configured to have a sufficiently high data rate, so no messages are dropped due to network congestion.

[1] Longitude is given as an angular measurement ranging from -180 to $+180$ degree, so the denominator is 360.

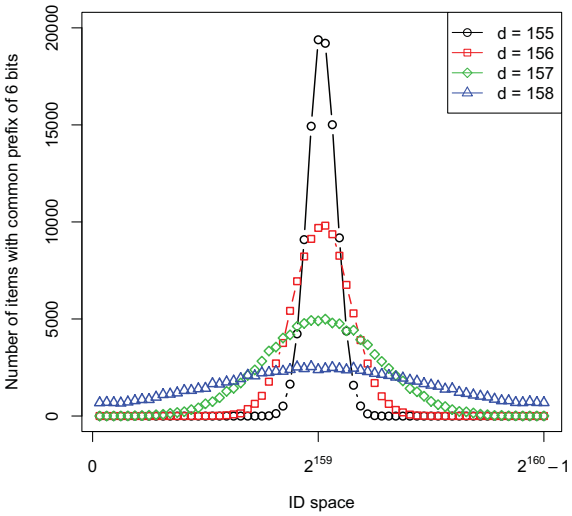

Figure 5. Data distribution of 1000000 data items in the ID space D for the Gaussian workload model.

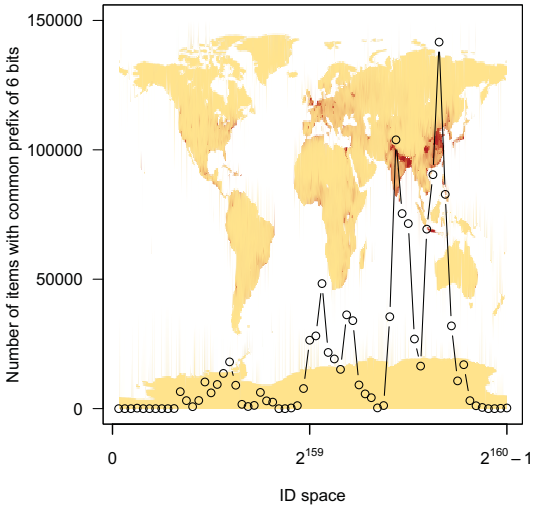

Figure 6. Data distribution of 1000000 data items in the ID space D for the GPS workload model. The distribution is derived from population density on earth.

We dealt with stale overlay links by sending small acknowledgment messages for every overlay hop of an application request. On a timeout, the corresponding link was removed and the request was tried to be forwarded using the remaining overlay links. As a result, every dropped request is dropped at the application level, not at the routing or network level. This provides a clear picture of our algorithm's benefit.

In our simulation the average peer lifetime is Pareto-distributed. We follow the previous work of Ledlie and Saltzer [9] and choose the shape parameter of the distribution to be $\alpha = 2$. The average peer lifetime is 60 minutes, unless noted otherwise, which implicitly defines the distribution's scale parameter to be $\beta = 0.5$.

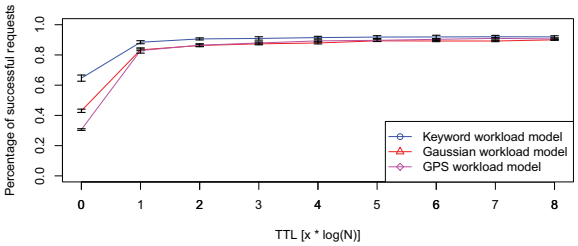

Figure 7. Impact of varying the TTL value with $N = 1024$ peers.

We define system load based on the capacity available in the P2P network, i.e., at a default system load of 90%, peers with a capacity of 10 requests/second issue 9 requests/s. Although these levels of load might seem unreasonably high at first glance, they highlight the critical regions for our ID management approach. Since our approach falls back to assigning random IDs if no overloaded peer has been discovered in the bootstrapping process, its performance in low load scenarios equals the unbalanced case.

In practice, the high load must not necessarily be generated by the peer itself. Instead, each peer could act as a proxy node [32] to several other nodes (potentially several hundred) that are not themselves part of the DHT network but use the DHT network as infrastructure to perform requests (e.g. name resolution requests as stated in the introductory example) via these proxy nodes. If not stated otherwise the TTL for ID Requests is $4 \cdot R$ where R is derived from the routing complexity of the overlay routing protocol. As we use Chord, R is $\log_2 N$ here. In all simulations, we set the number of possible forwarding links s that the method PROCESSIDREQUEST chooses from to 3. The number of entries k in the Destination Table DT is set to 100.

B. Results

Varying TTL: At first, we discuss the impact of the TTL parameter which defines the number of hops an ID request makes in the P2P network to find a suitable ID for joining. Fig. 7 illustrates the percentage of successful requests depending on the TTL. The case $TTL = 0 \cdot \log_2 N$ thereby corresponds to the simulation without ID management. The plot contains confidence intervals at a confidence level of 90% generated from 30 individual runs. Since we observed comparably low deviations for our other experiments as well, we decided to omit the confidence intervals in the remaining figures for the sake of legibility.

For $TTL = \log_2 N$, the increase of successfully answered requests is already about 36% in the keyword workload model, 93% in the Gaussian workload model and more than 170% for the GPS workload model. In all three considered workload models, the percentage

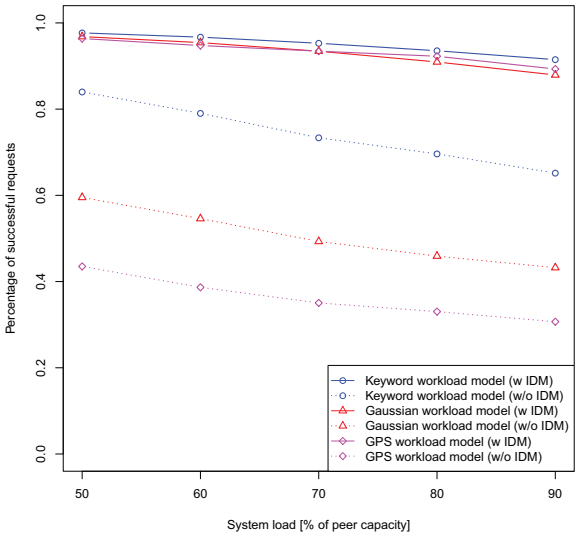

Figure 8. Impact of varying system load with $N = 1024$ peers.

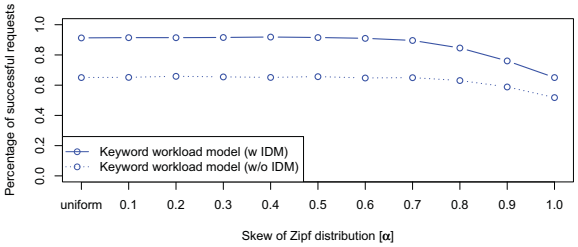

Figure 9. Impact of varying execution skew with $N = 1024$ peers.

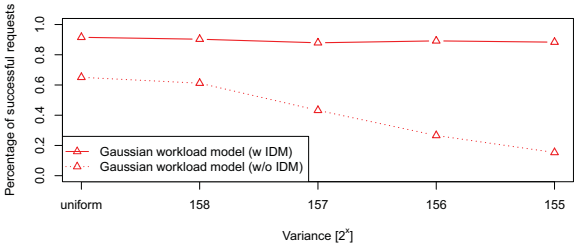

Figure 10. Impact of varying data skew with $N = 1024$ peers.

of peers joining at a random position, i.e., where our algorithm cannot suggest a suitable ID, is less than 4% and decreases further with increased TTL values. Moreover, in this scenario with $N = 1024$ peers, more than 90% of the issued ID requests already found a suitable overloaded peer after less than 7 hops regardless of the used workload model, illustrating that our algorithm only needs a relatively small number of extra messages to be successful. Increasing the TTL beyond $\log_2 N$ only leads to small improvements with respect to the rate of successful requests. Please note that this also implies that knowing the correct number of nodes N in the network is not critical. If N is chosen too large, this only results in a slightly larger number of forwarding steps for a message. In practice, one can use a generously chosen upper bound for N without adding significant additional messaging overhead.

Varying system load: Fig. 8 shows the percentage of successful requests depending on the system load for all three workload models, with ID management (*w IDM*) and without (*w/o IDM*). As expected, the success rate decreases with increasing system load. This figure illustrates the significant advantages of using our ID management approach: Without ID management, the system only achieves a success rate of 43% at a system load of 50% in the GPS workload model. At a system load of 90%, the success rate even drops to 30%. With ID management enabled, the success rate is increased almost by factor 3. The system achieves a success rate of almost 97%, gracefully droping to 89% at a system load of 90%. The performance of our approach becomes even more apparent considering that no retransmissions are done in the simulation. Any dropped request that may result from the sudden departure of a busy peer and its temporary overloaded neighbor is negatively reflected in the success rate.

For the Gaussian and keyword workload model, our ID management algorithm shows a similar success rate, but the gap to the unbalanced case is smaller.

Varying variance/skew In this experiment, we vary the parameters of our keyword and Gaussian workload model, respectively. Fig. 9 depicts the performance of our algorithm for the keyword workload model in comparison to the unbalanced case when varying α, the skew of the Zipf distribution. The performance gain through ID management is constantly about 40% starting from a uniform popularity distribution (no execution skew) to $\alpha = 0.7$ (high execution skew). The results are reasonable since the data items are distributed uniformly in the ID space D. Even with $N = 1024$ peers, the probability that a single peer is responsible for two popular data items is small. For higher levels of skew ($\alpha \geq 0.8$), the popularity of a single data item is sufficient to overload a peer. Hence, the success rate for both the balanced and unbalanced case begins to decrease, since ID management approaches in general cannot guard against this problem.

For the Gaussian workload model (Fig. 10), the performance improvement from our ID management is also about 40% at the beginning (uniform data distribution). However, it steadily increases with increased data skew. At $v = 155$ (high data skew), the sucess rate for the unbalanced case is as low as 15% whereas the ID management algorithm can handle the data skew with a success rate of 88%.

An interesting property of the P2P network in this context is the average hop count and the *in degree* of peers, i.e., the number of overlay links pointing to a particular peer. While the average hop count is mainly important for the routing latency, the in degree of a peer

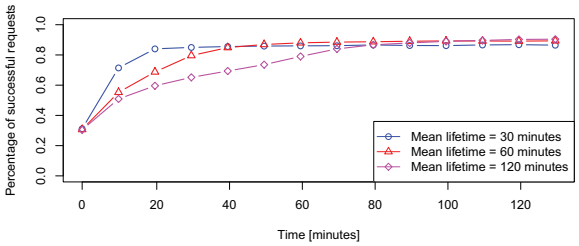

Figure 11. Impact of varying churn with $N = 1024$ peers.

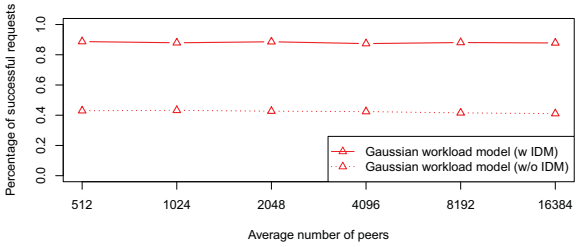

Figure 12. Impact of varying the number of peers.

is an important metric for the stability and distribution of overlay network's routing paths. For the keyword workload model, the average hop count and the in degree distribution among the peers does not change by using our ID management algorithm. For the Gaussian workload model, the average hop count remains almost unaffected by our ID management as well. However, the in degrees of particular peers can rise significantly, i.e., the variance of the in degree distribution is increased. This is not problematic in scenarios where the bottleneck is the number of processed messages per time unit as evaluated here. In other scenarios where the network is also considered to be a bottleneck, this drawback can be circumvented by using overlay routing protocols that construct their links in the node space rather than the ID space [33].

Varying churn: Fig. 11 depicts the response time of our algorithm for different levels of churn using the GPS workload model. At the time $x = 0$, no ID management has been performed yet. Since the algorithm requires natural churn to position peers, the success rate for the shortest average lifetime (30 minutes) begins to rise the fastest. After 80 minutes, all three considered cases have approached a success rate of almost 90%. After that point in time, the success rate for the peers with the highest average lifetime (120 minutes) begins to slightly exceed the others as fewer peers depart.

For the keyword workload model, the differences in slope are less distinctive. In all three workload models, the load balancing achieves their steady state after at most 100 minutes. This makes our approach suitable even for applications with high drifts in the workload distribution.

Varying number of peers: The results of our scalability tests are illustrated in Fig. 12. Here, the success rate

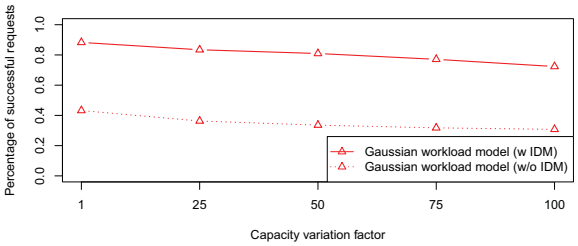

Figure 13. Impact of varying the peers' capacity with $N = 1024$ peers.

is depicted depending on the average number of peers in the P2P network for the Gaussian workload model. All simulations have been conducted with a TTL of $4 \cdot \log_2 N$. Again, the algorithm provides a success rate of about 90% for all considered numbers of peers. This represents a constant improvement of about 100% to the unbalanced case.

Varying peer capacity: To conclude our evaluation, Fig. 13 shows the performance of our algorithm when the capacity of the participating peers is no longer equal. For this experiment, we defined a capacity variation factor which denotes the ratio between highest and lowest capacity of a peer in the system. Upon initialization, peers choose their capacity uniformly randomly from the interval $[C_{lowest}, C_{highest}]$. The rate of successful requests drops with increasing capacity variation factor. This is to be expected since the assumption that highly frequented regions of the ID space also point to overloaded peers becomes less appropriate. However, the decrease is modest, illustrating the usability of our algorithm even for highly heterogeneous networks. Even with a capacity variation factor of 100, a factor far beyond most practical applications, the success rate is still at 72% with load balancing whereas the unbalanced system can only answer 30% of the issued application requests successfully.

V. Conclusion

In this paper we introduced a novel load balancing algorithm for structured P2P networks based on ID management. Its key idea is to observe message flows during regular network operation and use the collected statistics to direct joining peers towards highly-frequented regions of the ID space.

Unlike previous approaches, we focus on applications where the number of requests a peer can process at the application level is the bottleneck. We examined our algorithm for different workload models, churn rates, and capacity distributions. The performance results lets us conclude that it provides good load balancing properties, even in scenarios with frequent peer arrival/departure, a highly skewed workload distribution and heterogeneous peers. In particular, our work presents the first ID management approach that is able to respond to data skew.

With the emphasis on processing application requests we think our algorithm closes an important gap in the

set of existing load balancing schemes and provides a valuable contribution to the design of flexible and robust distributed systems.

REFERENCES

[1] G. S. Manku, "Balanced binary trees for ID management and load balance in distributed hash tables," in *PODC '04: Proc. Twenty-Third Annual ACM Symposium on Principles of Distributed Computing*. New York, NY, USA: ACM, 2004, pp. 197–205.

[2] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for Internet applications," *IEEE Transactions on Networking*, vol. 11, no. 1, pp. 17–32, February 2003.

[3] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*. London, UK: Springer-Verlag, 2001, pp. 329–350.

[4] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao, "Distributed object location in a dynamic network," in *SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM, 2002, pp. 41–52.

[5] G. Giakkoupis and V. Hadzilacos, "A scheme for load balancing in heterogenous distributed hash tables," in *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 2005, pp. 302–311.

[6] P. Ganesan, M. Bawa, and H. Garcia-Molina, "Online balancing of range-partitioned data with applications to peer-to-peer systems," in *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, 2004, pp. 444–455.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *Proc. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99)*, vol. 1, March 1999, pp. 126–134.

[8] S. Zhao, D. Stutzbach, and R. Rejaie, "Characterizing files in the modern gnutella network: A measurement study," in *In Proceedings of SPIE/ACM Multimedia Computing and Networking*, vol. 6071, 2006.

[9] J. Ledlie and M. Seltzer, "Distributed, secure load balancing with skew, heterogeneity and churn," in *Proc. 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM05)*, vol. 2, March 2005, pp. 1419–1430.

[10] S. Bianchi, S. Serbu, P. Felber, and P. Kropf, "Adaptive load balancing for DHT lookups," in *Proc. 15th International Conference on Computer Communications and Networks (ICCCN 2006)*, Oct. 2006, pp. 411–418.

[11] B. Ahlgren, M. D'Ambrosio, C. Dannewitz, M. Marchisio, I. Marsh, B. Ohlman, K. Pentikousis, R. Rembarz, O. Strandberg, and V. Vercellone, "Design considerations for a network of information," in *Proceedings of the First International Workshop on Re-Architecting the Internet (ReArch2008)*, Madrid, Spain, December 2008.

[12] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *SIGCOMM '07: Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: ACM, 2007, pp. 181–192.

[13] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, kc claffy, D. Krioukov, D. Massey, C. Papadopoulos, T. Abdelzaher, L. Wang, P. Crowley, and E. Yeh, "Named data networking (NDN) project," PARC, TechReport NDN-0001, October 2010. [Online]. Available: http://www.named-data.net/ndn-proj.pdf

[14] C. Dannewitz, J. Golic, B. Ohlman, and B. Ahlgren, "Secure naming for a network of information," in *Proc. 13th IEEE Global Internet Symposium 2010*, San Diego, USA, March 2010.

[15] M. Kindler, A. Leschinsky, and D. A. Quellmalz, "WM-streaming," *c't magazin für computer technik*, vol. 16, pp. 136–139, July 2010.

[16] M. Cai, M. Frank, J. Chen, and P. Szekely, "MAAN: A multi-attribute addressable network for grid information services," in *GRID '03: Proc. Fourth International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2003, p. 184.

[17] T. Biermann, C. Dannewitz, and H. Karl, "An adaptive resource/performance trade-off for resolving complex queries in P2P networks," in *Proc. IEEE International Conference on Communications (ICC)*, Dresden, Germany, June 2009.

[18] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: A public DHT service and its uses," in *SIGCOMM '05: Proc. 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: ACM, 2005, pp. 73–84.

[19] D. R. Karger and M. Ruhl, "Simple efficient load-balancing algorithms for peer-to-peer systems," *Theor. Comp. Sys.*, vol. 39, no. 6, pp. 787–804, 2006.

[20] D. Warneke and C. Dannewitz, "Statistics-based ID management for load balancing in structured P2P networks," in *Proc. 34rd IEEE Conference on Local Computer Networks (LCN 2009)*. Piscataway, NJ, USA: IEEE Computer Society, October 2009, pp. 273–276.

[21] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems," in *Proc. 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, vol. 4, 2004, pp. 2253–2262.

[22] Y. M. Zhu and Y. Hu, "Efficient, Proximity-aware Load Balancing for DHT-based P2P Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 4, pp. 349–361, April 2005.

[23] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area Cooperative Storage with CFS," *SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 202–215, 2001.

[24] M. Naor and U. Wieder, "Novel architectures for P2P applications: The continuous-discrete approach," *ACM Trans. Algorithms*, vol. 3, no. 3, p. 34, 2007.

[25] I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, and E. Pavlov, "A generic scheme for building overlay networks in adversarial scenarios," *Proc. Parallel and Distributed Processing Symposium*, April 2003.

[26] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM, 1997, pp. 311–320.

[27] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: a resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41 – 53, 2004.

[28] I. Baumgart, B. Heep, and S. Krause, "OverSim: A flexible overlay network simulation framework," in *Proc. 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007*, May 2007, pp. 79–84. [Online]. Available: http://doc.tm.uka.de/2007/OverSim_2007.pdf

[29] A. Klemm, C. Lindemann, M. K. Vernon, and O. P. Waldhorst, "Characterizing the query behavior in peer-to-peer file sharing systems," in *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2004, pp. 55–67.

[30] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 314–329, December 2003.

[31] National Aeronautics and Space Administration, "Visible earth: Population density," http://visibleearth.nasa.gov/view_rec.php?id=116.

[32] T. H. ting Hu, B. Thai, and A. Seneviratne, "Supporting mobile devices in gnutella file sharing network with mobile agents," *Proc. Eighth IEEE International Symposium on Computers and Communication (ISCC 2003)*, vol. 2, pp. 1035–1040, July 2003.

[33] T. Schütt, F. Schintke, and A. Reinefeld, "Range queries on structured overlay networks," *Computer Communications*, vol. 31, no. 2, pp. 280–291, February 2008.

Daniel Warneke studied computer science at the University of Paderborn, Germany. Currently, he is a research assistant at the Complex and Distributed IT Systems (CIT) group at Technische Universität Berlin, Germany, working on massively-parallel, fault-tolerant data processing frameworks on Infrastructure-as-a-Service platforms.

His research interests center around the areas of Cloud Computing and large-scale Distributed Systems.

Christian Dannewitz studied computer science and electrical engineering at the University of Paderborn, Paderborn, Germany, and the Carleton University, Ottawa, Canada. He is currently driving the research of ubiquitous and information-centric networking at the Research Group Computer Networks of the University of Paderborn. Prior to his work at the University of Paderborn, he led the research & development unit of an international software company, focusing on information management.

His research interests include Future Internet technologies with a focus on network architecture and information-centric networking.