Aggregating Information in Peer-to-Peer Systems for Improved Join and Leave[*]

Keno Albrecht, Ruedi Arnold, Michael Gähwiler, Roger Wattenhofer
Swiss Federal Institute of Technology
Department of Computer Science
8092 Zurich, Switzerland
{kenoa@inf, rarnold@inf, mgaehwil@student, wattenhofer@inf}.ethz.ch

Abstract

In this paper, we introduce the Distributed Approximative System Information Service (DASIS) as a useful scheme to aggregate approximative information on the state of a peer-to-peer system. We present how this service can be integrated into existing peer-to-peer systems, such as Kademlia and Chord. As a sample application, we show how DASIS can be employed for establishing an effective deterministic join algorithm. Through simulation, we demonstrate that the insertion of peers using DASIS information results in a well-balanced system. Moreover, our join algorithm gracefully resolves load imbalances in the system due to unfortunate biased leaves of peers.

1. Introduction

Peer-to-peer (P2P) systems connect ordinary desktop computers throughout the Internet, leveraging their united power beyond the sum of the individual parts. In a P2P system, peers share computer resources and services without a central coordinator. The most prevalent publicly available peer-to-peer systems, such as Kazaa or BitTorrent, focus on the task of sharing multimedia data. In the near future, however, we envision P2P systems designed for more elaborate tasks, such as online collaborations or transactional database systems.

The growing popularity of real-world P2P systems has spawned a thriving research community. The focus of most research is the development of an efficient lookup operation: given a search key, a peer responsible for the key must be identified. This operation is related to hashing and is therefore sometimes also known as distributed hashing in conjunction with a distributed hash table (DHT).

Following the seminal work of Plaxton et al. [9], an assortment of variants of P2P systems and distributed hashing algorithms have been proposed in the literature, such as CAN [11], Chord [13], PAST [4], or Tapestry [17].

For concreteness, we present our results for “tree” topology P2P systems, such as Kademlia [8]. As discussed in Section 5, we can support other topologies as well. For completeness, we give a quick overview on a Kademlia-like tree topology. Each peer is assigned a unique overlay identifier, a binary bit string. This ID specifies the “domain space” of the peer; a peer is responsible for storing all keys that are within its domain space. In particular, a key is stored by the peer whose bit string matches the longest prefix of the key. A peer p with the bit string $b_1 b_2 \ldots b_k$ keeps contact with k other peers—its “neighbors.” Neighbor p_i ($i = 1, \ldots, k$) of peer p features a similar bit string as peer p; in particular, all the first $(i-1)$ bits are the same as the bits of peer p, and the bit i itself is inverted. Note that various systems handle the remaining bits differently; this difference is not relevant in this paper.

The peers are connected in such a way that an efficient logarithmic (in the number of peers) search operation is guaranteed, and at the same time each peer only needs to connect to a logarithmic or even constant (such as with Viceroy [7] or Koorde [6]) number of peers.

In essence, the logarithmic degree (number of neighbors) and the logarithmic diameter (search time) of P2P systems are required by the dynamics and fluctuation of a P2P system. Peers are highly unreliable—most users owning a peer will join the P2P system only for the time they personally use it, e.g. for searching and downloading files, which is on average only one hour [12]. A P2P system does not feature a stable central server (or a group of servers) that manages the system. Instead, the P2P system is managed by the peers themselves. Having only a logarithmic number of neighbors helps in the case of a leaving peer, because only a logarithmic number of peers, that is, the neighbors of the peer that left, must search for a replacement.

[*] The work presented in this paper was supported (in part) by the Hasler Stiftung under grant number 1828 and the Swiss National Science Foundation under grant number 21-66768.01.

1.1. Joins and leaves

An important lingering problem, and the primary focus of this paper, is how the overlay ID is assigned to a peer. Since the P2P system is completely decentralized and highly dynamic, present solutions assign the overlay IDs randomly: a newly joining ("bootstrapping") peer connects to an arbitrary peer in the P2P system and chooses a random overlay ID, e.g. by hashing its own IP address. Similarly to a lookup operation, the newly joining peer is routed to its place—determined by the chosen overlay ID—in the P2P system and connects to its neighbors. A peer leaving the P2P system, which generally happens without notification, drops all stored keys at once.

It is often argued that random overlay ID association will balance the keys well. This is not quite true; in fact, a balls-into-bins analysis will reveal that there is a logarithmic imbalance factor [3]. In other words, with high probability a highly loaded peer stores a factor of $\Theta(\log n)$ more keys than a peer with average load.

There have been a number of recent randomized proposals on how to improve the imbalance. Byers et al. [3] applied the "power of two choices"-paradigm to reduce the logarithmic imbalance to $\Theta(\log n/\log\log n)$. Rao et al. [10] adopted hill-climbing techniques. In this paper, we propose a non-randomized join algorithm deploying a new abstract distributed approximative information service for P2P networks, which leads to well-balanced P2P systems. Even though we stress only join and leave and thus load balancing in this paper, our aggregation mechanism can be applied to several other applications, such as system monitoring, e.g. the "health" of the system [16], or publish/subscribe, e.g. how many peers are interested in some topic.

The paper is organized as follows: in Section 2 we introduce the Distributed Approximative System Information Service (DASIS) for P2P systems. Section 3 explains our join algorithm based on DASIS. We show simulation results and discuss some implementational issues of DASIS in Section 4. In Section 5, we consider how to integrate DASIS into existing P2P systems, such as Chord. Section 6 discusses related work and, finally, Section 7 concludes the paper.

2. Distributed Approximative System Information Service

The *Distributed Approximative System Information Service* (*DASIS*) is an abstract decentralized service which provides approximate[1] information about the P2P system.

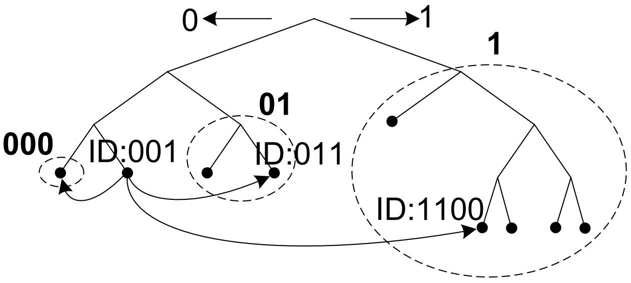

Figure 1. A sample illustration of sub domains. Dashed circles indicate the partitioning of the system into sub domains for peer 001.

DASIS is built on top of the regular P2P structure as sketched in the introduction. The basic idea is as follows: a peer p with the bit string $b_1b_2\dots b_k$ is considered to be an "expert" on all the sub domains of all the prefixes of its bit string, that is, for $b_1b_2\dots b_i$, $i = 0,\dots,k$. The expert knowledge is constructed inductively through information exchange with the neighbor peers. The peer p is by definition an expert about its own sub domain $b_1b_2\dots b_k$. Also, the peer p can deduce the state in sub domain $b_1b_2\dots b_i$ by aggregating its own knowledge on sub domain $b_1b_2\dots b_{i+1}$, which is available by induction, with the knowledge provided by neighbor peer p_{i+1} about sub domain $b_1b_2\dots\overline{b_{i+1}}$. In the end, peer p can deduce the state of the whole P2P system, which is equivalent to the sub domain of the empty prefix.

For illustration, we give an example: we use DASIS to learn the total number of peers in the P2P system. We assume to have a stable P2P system, as shown in Figure 1.

We describe our example from the perspective of peer p with the bit string 001 (see Figure 2). Peers (periodically) exchange sub domain information with their neighbors. In particular, peer p sends the information that there is one peer in sub domain 001 to neighbor peer p_3 (with ID 000), and in exchange learns that there is one peer in sub domain 000 from neighbor p_3. Literally summing up one and one, peer p deduces that there are 2 peers with prefix 00. Similarly, on the next higher level, peer p exchanges information with neighbor peer p_2 (ID 011) to learn that there are 2 peers with prefix 01. This sums up to a total of 4 peers with prefix 0. In a last step, peer p learns from neighbor peer p_1 (ID 1100) that there are 5 peers with prefix 1.

Since there are 4 peers with prefix 0 and 5 peers with prefix 1, peer p knows that there is a total of 9 peers in the P2P system. Note that DASIS runs simultaneously at every peer, and, therefore, provides information in a bottom-up aggregated manner at every peer.

[1] The exact up-to-date state of the whole system cannot be known. This would be equivalent to consensus in an asynchronous and dynamic distributed system, which is well known to be impossible.

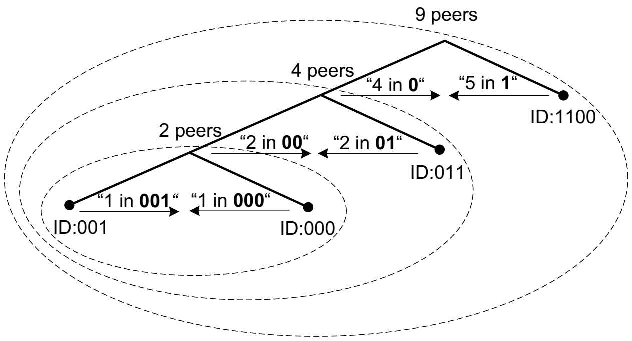

Figure 2. Illustration of the messages exchanged by peer p (ID 001) for the example given in Figure 1.

The accuracy of DASIS depends on the message propagation mechanisms of the implementation. In a static system, that is, without peers joining or leaving, DASIS provides exact information without message overhead. In a dynamic system, however, more accuracy requires more frequent message exchange between neighbors. We discuss some of these implementational issues of DASIS in more detail in Subsection 4.1.

Besides computing the total number of peers in the system, DASIS can deliver a wide range of information, such as the average up-time of peers, the total amount of bytes stored in the system, or, as we show in the next section, the minimal depth of a peer in the tree structure.

3. Join algorithm using DASIS

The insertion of new peers is an essential and challenging operation in a P2P system. In this section, we introduce a join algorithm as an example application using information provided by DASIS.

3.1. Random join (RJ)

Assignment of overlay IDs and insertion of new peers into the P2P system is typically done similarly in various P2P proposals. As stated in Section 1, joining peers are routed to their destination, which is determined by their randomly assigned overlay ID. We include this *random join algorithm (RJ)* for reference in our simulations.

3.2. Depth join (DJ)

For our join algorithm, we employ the *minimal depth service*. The *depth* of a peer is defined as the length of its bit string. Note that we use bit strings of variable length. If the bit strings would be of fixed length, the depth of a peer is the length of the so far assigned prefix of its bit string.

The minimal depth service of DASIS works as follows (we consider the example given in Figures 1 and 2 again): peer p with ID 001 wants to know in which sub domain a peer with minimal depth can be found. From its neighbor peer p_3 (ID 000) it knows that its minimal depth is 3, and so deduces that with prefix 00 the minimal depth is 3, since both the sub domain of p_3 and p have the same minimal depth. In the next inductive step, through information exchanged with neighbor p_2 (ID 01), peer p learns that the minimal depth in the sub domain of p_2 is 3 as well. In a last step, peer p gets to know from neighbor p_1 (ID 1100) that the minimal depth in its sub domain is 2. Thus, the overall minimal depth is 2, and DASIS provides peer p with this result.

Generally, using the *depth join algorithm (DJ)*, each peer can deduce the minimal depth of its sub domains. A new peer (*joiner*) is first routed through the P2P system to sub domains with the smallest minimal depth and then assigned an overlay ID. At every peer passed, one bit of the bit string of the joiner is fixed; this guarantees termination. If a peer (*inserter*) cannot route the joiner any further, it becomes responsible for inserting the new peer. The inserter assigns the joiner its own bit string plus a 1, and appends a 0 to its own bit string, thus splitting its domain space in half. In our example, the joiner would be routed to the peer with ID 10 which splits its current domain space in half, inserts the new peer with ID 101 into the system, and chooses the ID 100 as its own. Afterwards, the tree is balanced with a minimal depth of 3

It is worth mentioning that the number of peers in a certain sub domain is not a good criterion for inserting new peers. Consider Figure 1 again. A newly joining peer is inserted on the left half of the tree, that is with prefix 0, because the sub domain with prefix 0 is sparser. Since the most loaded peer (ID 10) remains at depth 2, this does not reduce the imbalance in the P2P system. Therefore, we chose the minimal depth as our criterion for inserting peers.

Note that our join approach can also adapt to other criteria than the depth of a node, such as the average number of requests per node, the available disk space or cpu time, or even combinations thereof. Moreover, it can be used with other load balancing strategies, such as load-stealing or load-shedding as described in [3].

As an additional feature, our join algorithm also works against attackers: a malicious adversary might attack a random join system by simply taking out all the peers of a sparse sub domain, making that sub domain even sparser, and raising the load of the remaining peers in the sub domain. Our non-randomized solution will constantly guide newly joining peers towards the sub domain with smallest minimal depth, filling the gaps of the peers that left.

4. Simulation

We conducted a series of experiments running fine-grained event-driven simulations of our algorithm—factoring in, for example, message delay. We ran several simulations in order to test the algorithm in different realistic situations. The size of the simulated P2P systems varies in the range from a couple of hundreds up to tens of thousands of peers. New peer arrivals are modeled as a Poisson process. In addition, each simulation consists of 10 runs in order to average all measures and have statistically stronger results.

We use two simple, but reasonable and handy criteria for evaluating the proposed algorithms. The first is the minimal depth D of a peer in the system. We have chosen this quality measure since a small depth denotes a high load. We assume a large and uniform distributed key population. Therefore, we do not assign keys to peers in our simulation. Note that in this case, decreasing the depth of a peer by 1 increases its load by a factor of 2. We desire a minimal depth to be as close as possible to the optimal minimal depth.

The second measure, the balance measure B, is more fine-grained and also takes the number of peers with small depths into account:

$$B = \sum_{i \in V} 2^{-2d_i} > 0$$

where V is the set of all peers in the system and d_i is the depth of peer i. This way we have a weighted measure in which peers with smaller depth contribute more than peers with larger depth. For expressiveness, we normalize the balance B_{Alg} of algorithm Alg with the optimal balance B_{Opt}:

$$\rho_{Alg} = \frac{B_{Opt}}{B_{Alg}} > 0$$

Note that in an optimal balanced P2P system all peers are at depth $\lfloor log_2(n) \rfloor$ and $\lceil log_2(n) \rceil$. By definition, an optimal algorithm Opt has a ρ_{Opt} of 1.0.

4.1. Implementational issues

In this subsection, we describe three different implementations of our join algorithm.

As a first approach, the description of DASIS in Section 2 mentioned that every peer *periodically* sends update messages to all of its neighbors. The shorter the update interval, the more accurate is the information available for joining peers. On the other hand, the shorter the interval, the more update messages must be transmitted. To be practical, the number of messages should be small and scale with the size of the system.

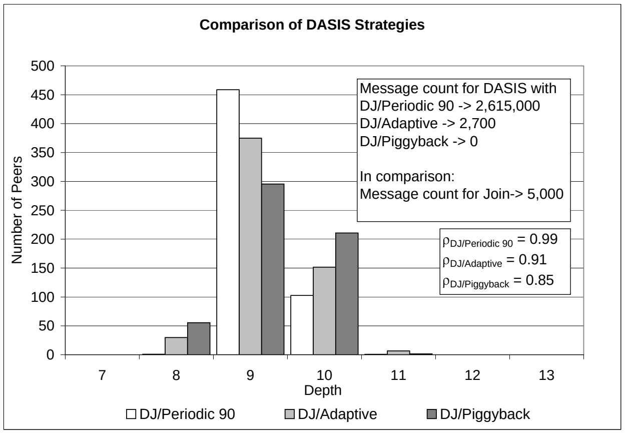

Figure 3. This graph shows the comparison of the minimal depth join algorithm using the three described DASIS approaches—*periodic DASIS* with an update interval of 90 time units, *adaptive DASIS*, and *piggyback DASIS*. The simulation started with an initial P2P system containing two peers and inserted 561 peers. This leads to an optimal depth of 9.

As an improvement, our second algorithm uses an *adaptive* technique. Messages are only sent if there is a change in the DASIS data set. For the DASIS service providing minimal depth information this means: a peer sends an update of the DASIS information to its neighbor if the peer detects a change in the minimal depth. Because changes of minimal depth in a P2P system only take place rarely, this clearly reduces the amount of messages sent. Using the adaptive technique, the total message count drops from millions to thousands, as can be seen in Figure 3. However, reducing the number of messages also reduces the quality of the join algorithm.

Our final and preferred approach to reduce the message overhead employs updating DASIS information while routing “regular” (e.g. lookup) messages through the P2P system. For our join application, we simply *piggyback* the minimal depth with the regular messages. This introduces no additional messages into the P2P system. As illustrated in Figure 3, the quality reduces gracefully.

Of course, piggybacking does not restrictively apply to the join application. Any application using a P2P system can employ piggybacking to spread information within its regular messages.

4.2. Scalability and imbalances

The advantage of the DJ algorithm is that it levels out imbalances. This can be seen in Figure 4, where the DJ/Piggyback algorithm levels out the initial imbalanced system much better than the RJ algorithm. Conse-

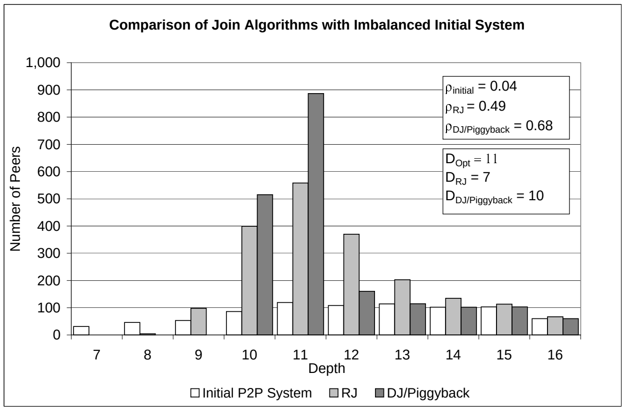

Figure 4. Initially the P2P system is populated with 995 peers in an imbalanced fashion. The graph shows the distribution of peers after inserting 1,113 peers to a total of 2,108 peers.

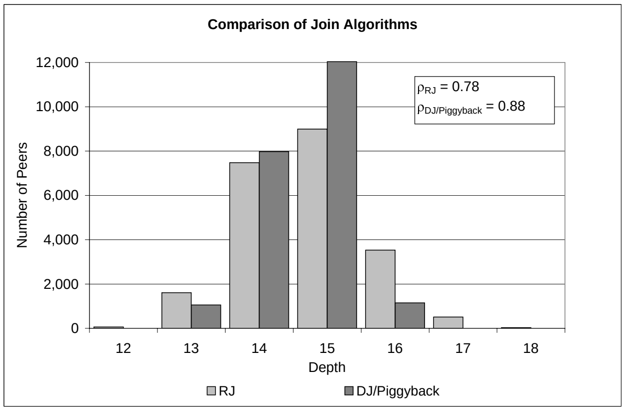

Figure 5. This graph shows the distribution of peers after insertion of 22,211 peers with an optimal depth of 14.

quently, the DJ algorithm is superior to the RJ algorithm in resolving load imbalances. Imbalances may for example arise from biased leaves of peers.

Simulations described so far use in the order of hundreds of peers. In order to evaluate the scalability of our approach, we simulate our proposed DJ algorithm with piggyback DASIS in a larger setting. Figure 5 shows that our solution performs better with respect to balancing in a P2P system with about 22,000 peers inserted, compared to the commonly used random join, at no additional message cost.

4.3. Steady state

In a last simulation, we use a realistic scenario where peers join, leave (fail), and perform lookup operations. The initial perfectly balanced P2P system consists of 1,024 peers. Peers join and leave at the same rate, that is, the expected number of peers stays at 1,024.

The lookup messages are used to spread piggybacked DASIS information. As argued in Subsection 4.1, regular routing traffic helps piggyback DASIS to approximate the system state, which in turn improves the join algorithm. For the simulation, we introduce a load parameter L: on average, a peer performs L lookups (helping DASIS) before it leaves the system (troubling DASIS).

Surprisingly, all algorithms (RJ as well as DJ/Piggyback with loads from 1 to 50) get into a steady-state depth distribution quite quickly. After a few thousand joins and leaves, the depth distribution as shown in Figure 6 remains more or less fixed, independently of the initial distribution. Since there are more messages which piggypack DASIS updates, a higher load L balances the system better. If every peer initiates on average $L = 50$ lookup messages before leaving, this leads to a well balanced system with $\rho \approx 0.88$.

More surprisingly, even when there are almost no lookups ($L = 1$), the steady-state balance is as good as RJ.

Note that with $L \to \infty$, $\rho_{DJ/L\infty}$ converges to 1. In this case, DASIS is accurate at any time and new joining peers are inserted at a "correct" position, that is, a peer with smallest minimal depth would split.

5. Integrating DASIS into existing P2P systems

The DASIS approach translates directly to several other (non-tree) P2P topologies. In this section, we exemplarily sketch how to integrate DASIS into ring topologies used, for example, by Chord [13]. Similarly, this description is also applicable to recently developed P2P systems based on the skip list data structure [1, 5]. On the other hand, P2P systems based on the De Bruijn graph [6] cannot easily be adapted to support DASIS because they have only a constant number of neighbors.

5.1. DASIS on Chord

In Chord, peers, respectively their IP addresses, and data keys are hashed into an identifier circle containing 2^m values, where m is the number of bits used for an identifier. For simplicity, we assume that all identifiers are in the interval $[0, 1)$. Every peer knows about $O(\log n)$ other peers (its *fingers*), where n denotes the number of peers in the system. To allow for routing a key in $O(\log n)$ hops toward its destination, the i^{th} finger ($1 \leq i \leq \log n$) spans about $1/2^i$ of

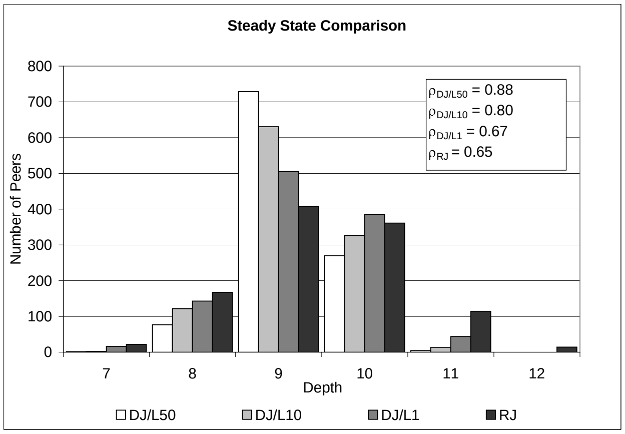

Figure 6. This simulation started with 1,024 perfectly balanced peers. It shows the depth of peers after peers leave and join at the same rate.

the circle so that every hop at least halves the distance to the destination.

Figure 7 shows the identifier circle of Chord and some finger information. For example, "`A.finger[3]`" denotes the 3^{rd} finger of peer A and thus references a peer whose identifier is at least $1/2^3$ *larger* than the identifier of peer A.

In the following, we sketch how to aggregate, for example, the number of peers in the Chord system. In this case, the request that a peer P with identifier `P.id` sends to its i^{th} finger is not "How many peers do you know with prefix length i?" as in DASIS, but "How many peers do you know in the interval `[P.finger[i].id,P.finger[i+1].id)`?", where $1 \leq i \leq \log n$ and `P.finger[(log n)+1]` denotes P itself. Thus, `P.finger[i]` is an "expert" on the interval `[P.finger[i].id,P.finger[i+1].id)`, and P can deduce the state of the whole P2P system by aggregating the information delivered by all of its fingers.

Our scenario shows an ideal Chord system where peers referenced by fingers have exactly the *desired* identifier, that is, the identifier of a peer referenced by `P.finger[i]` is exactly `P.id`$+1/2^i$. Unfortunately, in a real Chord system, finger interval lengths are not exactly powers of $1/2$; indeed, `P.finger[i]` often references a peer whose identifier is larger than the desired one. Thereby, it is likely that information delivered by different fingers is based on overlapping intervals. In our example of aggregating the number of peers, this means that the calculated values tend to be too large because some peers are counted twice. Thus, peer P needs to adjust the delivered values before summing them up. Since P is able to determine overlapping intervals, this is approximately possible.

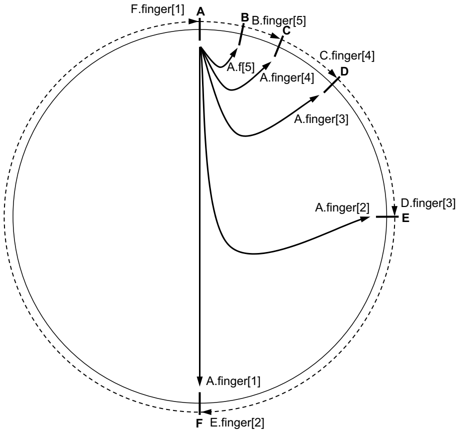

Figure 7. A Chord identifier circle and additional finger information. `P.finger[i]` denotes the i^{th} finger of peer P which spans about $1/2^i$ of the circle length.

Note that this problem is not relevant when computing the max and min functions because they are correct for overlapping intervals, too. Therefore, our proposed join algorithm can easily be adapted to work with Chord: the peers gather and aggregate information about the "maximum gap," that is, the largest interval between two neighboring peers on the identifier circle. A newly joining peer is then assigned an identifier within this interval.

6. Related work

In this section, we first present work related to DASIS as a general aggregation service. Thereafter, we compare our improved join algorithm to other approaches to load balancing in P2P systems.

6.1. Aggregation services

The Astrolabe system [14] is a distributed information management system, which the authors describe as a decentralized hierarchical database. Astrolabe employs an aggregation technique that is similar to the one presented here, although more powerful using a SQL-like query style. In comparison to our work, Astrolabe is presented as a new stand-alone system, which uses a gossip protocol to disseminate information, whereas DASIS is intented to be integrated into a P2P system, such as Chord [13] or Kademlia [8].

Improving a join algorithm or load balancing does not seem an obvious application of Astrolabe itself. Nevertheless, such an algorithm can leverage Astrolabe's aggregation mechanism. Just like SelectCast [2] sets up its own tree—maintained by Astrolabe information—of TCP connections for supporting publish/subscribe applications, our depth join algorithm can be employed in a separate DHT, again, using depth information provided by Astrolabe.

Astrolabe's follow up work Willow [15] also allows for SQL-like queries to aggregate information in a P2P system. Unlike Astrolabe, Willow integrates DHT functionality and directly supports publish/subscribe. Willow's aggregation mechanism works quite similar to that of DASIS. In fact, Willow could directly benefit from using our join algorithm.

Zhang et al. [16] introduced another infrastructure providing system meta information. In this approach, a "Self-Organized Meta Data Overlay" (SOMO) tree is built and maintained on top of an arbitrary DHT, such as CAN [11]. The tree grows and shrinks dynamically as the system size changes. All the information is aggregated bottom up along this tree, and disseminated down again.

Since SOMO implements a hierarchical approach, it can be used in a plug-in like fashion independently of the underlying (P2P) topology. Although this offers a variety of features, it can be criticized in a "pure P2P mindset," as done by the authors themselves. In some sense, DASIS provides SOMO functionality in a downright P2P style, with zero message overhead.

Furthermore, we consider the deterministic assignment of SOMO's root node a drawback. Although in the case of failures, another node automatically takes over the responsibility of the SOMO root node, with permanent—probably malicious—failures of the changing root node the SOMO service is at risk. Since DASIS operates on the regular P2P topology, it does not have a single point of failure, and, therefore, provides reliable information even in malicious environments.

6.2. Load Balancing

Assignment of IDs to joining peers using DASIS employs approximative global system information. At a high level, the idea of employing information about the system in order to assign IDs to joining peers can be found, in a local scope though, in CAN [11]. CAN proposes a join algorithm in which the joining peer chooses a random ID, and the peer responsible for this ID returns another ID that would split the most loaded peer among itself and all its neighbors.

Chord [13] maps multiple *virtual nodes* to each real peer to achieve load balancing. That is, each peer is inserted $O(\log n)$ times (n is the total number of peers in the system) with unrelated identifiers into the Chord ring. While this approach balances the number of key/data item pairs stored at each peer so that it differs (with high probability) only by an arbitrarily small constant factor ϵ, our approach could easily be adapted to match other criteria, such as the number of requests per peer, the total disk space available, or even combinations thereof. Furthermore, in Chord, a peer has to maintain a routing table for each virtual node, while the depth join algorithm can be employed without additional overhead. Finally, our join algorithm can even be used in conjunction with virtual nodes.

7. Conclusions

We introduced the Distributed Approximative System Information Service as a simple yet useful tool to aggregate information in a P2P system. As a sample application, we use minimal depth information provided by DASIS for our proposed join algorithm. We showed by simulations that a join with DASIS results in better balanced P2P systems than the standard assignment of random overlay IDs, especially in the case where the P2P system is imbalanced due to a high number of leaving peers. Finally, we sketched how to integrate DASIS into existing P2P systems.

Currently, we are planning to implement DASIS and our join algorithm in Java to perform real-world measurements. We are also integrating DASIS into the Chord join protocol. Additionally, we will provide a theoretical analysis of our join algorithm.

References

[1] J. Aspnes and G. Shah. Skip Graphs. In *Proceedings of Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[2] A. Bozdog, R. Van Renesse, and D. Dumitriu. Selectcast – a scalable and self-repairing multicast overlay routing facility. *Proceedings of the First ACM Workshop on Survivable and Self-Regenerative Systems*, 2003.

[3] J. Byers, J. Considine, and M. Mitzenmacher. Simple Load Balancing for Distributed Hash Tables. In *Proceedings of IPTPS*, Berkeley, CA, USA, February 2003.

[4] P. Druschel and A. Rowstron. PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility. In *HotOS VIII*, pages 75–80, Schloss Elmau, Germany, May 2001.

[5] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *Proceedings of USITS*, Seattle, WA, USA, March 2003.

[6] M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of IPTPS*, Berkeley, CA, USA, February 2003.

[7] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *Proceedings of PODC*, Monterey, CA, USA, July 2002.

[8] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *Proceedings of IPTPS*, Cambridge, MA, USA, March 2002.

[9] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.

[10] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Structured P2P Systems. In *Proceedings of IPTPS*, Berkeley, CA, USA, February 2003.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proceedings of ACM SIGCOMM*, 2001.

[12] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking*, San Jose, CA, USA, January 2002.

[13] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM*, 2001.

[14] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[15] R. van Renesse and A. Bozdog. Willow: DHT, Aggregation, and Publish/Subscribe in One Protocol. In *Proceedings of IPTPS*, San Diego, CA, USA, February 2004.

[16] Z. Zhang, S.-M. Shi, and J. Zhu. SOMO: Self-Organized Metadata Overlay for Resource Management in P2P DHT. In *Proceedings of IPTPS*, Berkeley, CA, USA, February 2003.

[17] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.