Join and leave in peer-to-peer systems

The DASIS approach

Join and Leave in Peer-to-Peer Systems: The DASIS Approach

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

November 2003

Abstract

In this paper we introduce the distributed approximative system information service (DASIS) as a useful means to collect approximate information about a peer-to-peer system. As an example application we show how this service can be employed for establishing an effective deterministic join algorithm. Through simulation we demonstrate that insertion of peers using the service 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 Gnutella or Kazaa 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.

An assortment of variants of P2P systems and distributed hashing algorithms have been proposed in the literature, such as Plaxton et al. [8], CAN [10], Chord [11], PAST [4], or Tapestry [13].

For concreteness, we present our results for "tree" topology P2P systems, such as Kademlia [7]. As discussed in Section 5, we can support other topologies as well. For completeness, we give a quick overview on this 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 bit string $b_1 b_2 \dots b_k$ keeps contact with k other peers—its "neighbors." Neighbor p_i $(i = 1, \dots, k)$ 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.[1]

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 peer connects to an arbitrary peer in the P2P system[2], and chooses a random overlay ID. Similarly to a lookup operation, the newly joining peer is routed to its place (determined by the randomly chosen overlay ID) in the P2P system, and connects to the neighbors. A peer leaving the P2P system (generally without notice) drops all stored keys at once.[3]

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 [2]. 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. [2] applied the "power of two choices"-paradigm to reduce the logarithmic imbalance to $\Theta(\log n / \log \log n)$. Rao et al. [9] adopted hill-climbing techniques. In this paper, we propose a non-randomized join algorithm deploying a new abstract distributed information service for P2P systems, which leads to well-balanced P2P systems.

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. Section 5 concludes the paper.

[1] Various systems handle the remaining bits differently; this difference is not of relevance in this paper.

[2] This process is known as "bootstrapping."

[3] In order to allow for fault tolerance, P2P systems store keys at several peers; if one of the peers goes down, there are still enough replicas available.

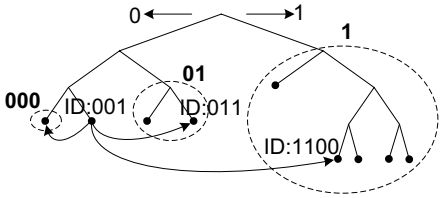

Figure 1: **A sample illustration of sub domains. Dashed circles indicate the partitioning of the system into sub domains for peer 001.**

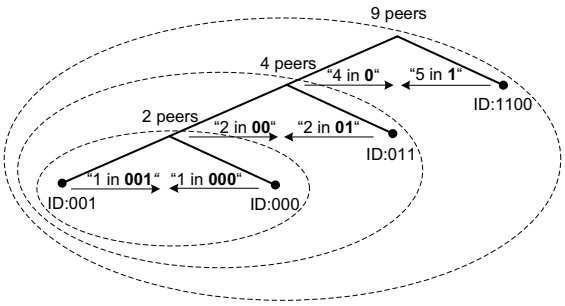

Figure 2: **Illustration of the messages exchanged by peer $p = 001$ for the example given in Figure 1.**

2 Distributed Approximative System Information Service (*DASIS*)

The distributed approximative system information service (*DASIS*) is an abstract decentralized service which provides approximate[4] information about the P2P system (e.g. the "health" of the system [12]).

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 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 $b_1b_2 \dots \overline{b_{i+1}}$ (peers periodically exchange sub domain information with their neighbors). 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 in Figure 1. We describe our example from the perspective of peer p with 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 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.

The accuracy of *DASIS* depends on the message propagation mechanisms of the implementation. In a stable system, *DASIS* provides exact information without message overhead. In a dynamic system, more accuracy requires more frequent message exchanges 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, or the total amount of bytes stored in the system.

The idea of having an infrastructure providing system meta information was introduced by Zhang et al. [12]. In this approach, a "Self-Organized Meta Data Overlay" (SOMO) tree is built and maintained on top of an arbitrary distributed hashtable (such as CAN [10]). 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 almost zero 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.

[4]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.

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 employing *DASIS* 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 (RJ)* algorithm for reference in our simulations.

3.2 Depth Join (DJ)

For our join algorithm we employ the *DASIS* minimal depth service. The *depth* of a peer is defined as the length of its bit string.[5]

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) it 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. The overall minimal depth is 2 and *DASIS* provides peer p with this result.

The minimal depth (*Depth Join*, short *DJ*) algorithm works as follows. Through the *DASIS* minimal depth service, 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 passing peer, 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 adds a 0 to its own bit string, thus splitting its domain space in half.

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. Newly joining peers are inserted on the left half of the tree, that is with prefix 0, since the sub domain with prefix 0 is sparser. This does not reduce the imbalance in the P2P system, since the most loaded peer (ID 10) remains at depth 2. We therefore chose the minimal depth as our criterion for inserting peers.

Note that our join approach can be combined with other load balancing strategies, such as load-stealing or load-shedding as described in [2].

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.

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 nodes can be found, in a local scope though, in CAN [10]. CAN proposes a joining algorithm in which the joining node choses a random ID, and the node responsible for this ID returns another ID that would split the most loaded node among itself and all its neighbors.

4 Simulation

We conducted a series of experiments running fine-grained event-driven simulations (factoring in, for example, message delay) of our algorithm. 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 stands for 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 depth 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

[5] Note that we use bit strings of variable length. If the bit strings are of fixed length, the depth of a peer is the length of the so far assigned prefix of its bit string.

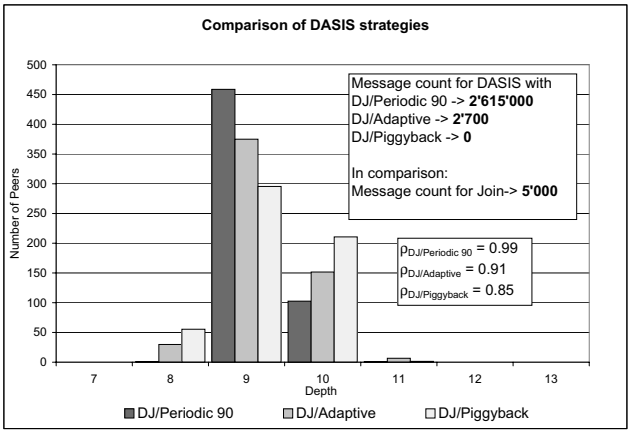

Figure 3: **This graph shows the three described approaches—*Periodic DASIS* with an update interval of 90 time units, *Adaptive DASIS*, and *Piggyback DASIS*—deployed for propagating DASIS information through the P2P system. The simulation started with an initial P2P system containing two peers and inserted 561 peers. This leads to an optimal depth of 9.**

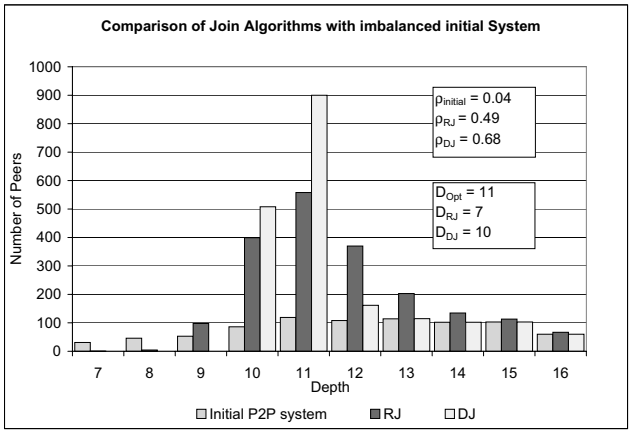

Figure 4: **Initially the P2P system is populated with 995 peers in an imbalanced fashion. The graph shows the distribution of peers after inserting 1113 peers to a total of 2108 peers.**

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}[6]:

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

(By definition, an optimal algorithm Opt has a ρ_{Opt} of 1.0.)

4.1 Implementational Issues

In the implementation of *DASIS* described in Section 3, every peer *periodically* sends update messages to 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.

As an improvement, our second approach 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 an 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 messages also reduces the quality of the join algorithm.

[6]Note that in an optimal balanced P2P system all peers are at depth $\lfloor log_2(n) \rfloor$ and $\lceil log_2(n) \rceil$.

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* algorithm levels out the initial imbalanced system much better than the *RJ* algorithm. Consequently, 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 P2P system consists of 1024 perfectly balanced peers. Peers join and leave at the same rate, that is, the expected number of peers stays at 1024.

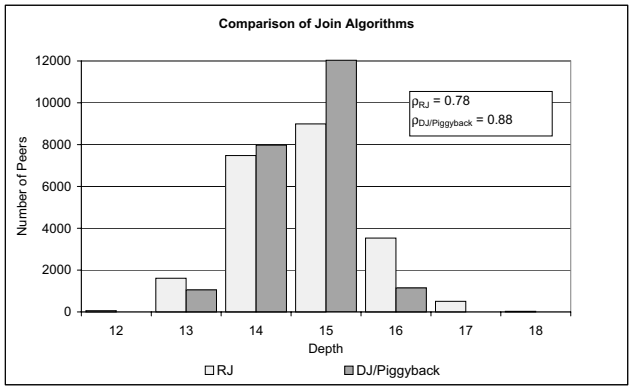

Figure 5: **This graph shows the distribution of peers after insertion of 22'211 peers with an optimal depth of 14.**

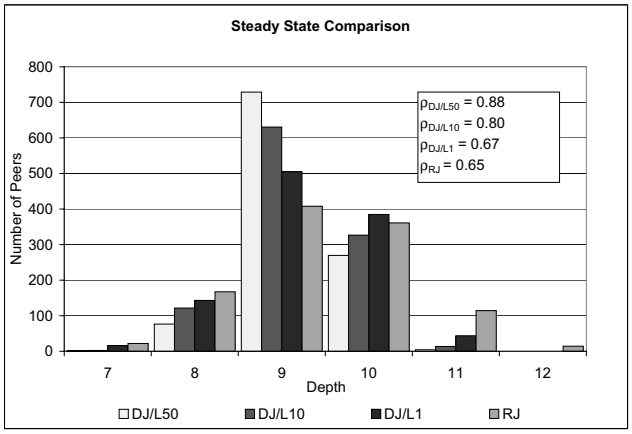

Figure 6: **This simulation started with 1024 perfectly balanced peers. It shows the depth of peers after peers leave and join at the same rate.**

The lookup messages are used to spread *DASIS* information (*Piggyback DASIS*). As argued in Subsection 4.1, regular routing traffic helps *Piggyback DASIS* to approximate the system state better, 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 *Piggyback DASIS* 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, independent of the starting distribution. Not surprisingly, a higher load L balances the system better. The reason being that *DASIS* information used for routing is more accurate, since there are more messages which piggypack *DASIS* updates. If every peer initiates on average L=50 lookup messages before leaving, this leads to a well balanced system with $\rho \approx 0.88$.[7] More surprisingly, even when there are almost no lookups ($L = 1$), the steady-state balance is as good as RJ.

5 Conclusions

We introduced the distributed approximative system information service as a simple yet useful tool for P2P systems. As a sample application, we showed how *DASIS* is employed for join operations. Our proposed join algorithm is based on minimal depth information provided by *DASIS*. We showed by simulations that join with DASIS results in better balanced P2P systems than the standard assignment of random overlay IDs, especially in the case where the leaves leave behind an imbalanced P2P system.

The DASIS approach translates directly to several other (non-tree) topologies, such as the recently developed P2P systems based on the skip list data structure [1, 5] or the Butterfly-based Viceroy system [6]. For some of the P2P systems mentioned in the introduction [11, 10], DASIS cannot be adopted directly – more research is necessary.

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

References

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

[2] J. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. In *IPTPS '03*, 2003.

[3] J. R. Douceur and R. P. Wattenhofer. Optimizing file availability in a secure serverless distributed file system. In *Proceedings of 20th IEEE SRDS*, 2001.

[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 USITS '03*, 2003.

[6] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings PODC'02*, 2002.

[7] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of IPTPS02, Cambridge, USA*, March 2002.

[8] 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.

[9] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. In *Proceedings of IPTPS'03*, Berkeley, CA, February 2003.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, 2001.

[11] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *SIGCOMM*, 2001.

[12] Z. Zhang, S.-M. Shi, and J. Zhu. Somo: Self-organized metadata overlay for resource management in p2p dht. In *Proceedings of IPTPS'03*, 2003.

[13] 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.