The Bootstrapping Service

Márk Jelasity†
University of Bologna, Italy
jelasity@cs.unibo.it

Alberto Montresor
University of Trento, Italy
alberto.montresor@dit.unitn.it

Ozalp Babaoglu
University of Bologna, Italy
babaoglu@cs.unibo.it

Abstract

We outline a lightweight architecture to support novel application scenarios for P2P systems. These scenarios include merging and splitting of large networks, or multiplexing relatively short-lived applications over a shared communication infrastructure. In such scenarios, the architecture needs to be quickly and efficiently (re)generated frequently, often from scratch. We propose the bootstrapping service abstraction as a solution to this problem. We present an instance of the service that can jump-start any prefix-table based routing substrate quickly, cheaply and reliably from scratch. We experimentally analyze the proposed bootstrapping service, demonstrating its scalability and its tolerance to message delivery failures.

1. Introduction

Structured overlay networks are increasingly seen as a key layer (or service) in peer-to-peer (P2P) systems, supporting a wide variety of applications. Index-based lookup is generally considered to be a “bottom” layer (e.g., [2, 12]), based on the assumption that the life cycle of the supported systems is similar to grassroots file sharing networks: there is at least one network that always exists and functions, while membership can change due to churn and the size of the network can also fluctuate but relatively smoothly. Join operations are assumed to be uncorrelated. Simulation and analytical studies also reflect these assumptions, typically modeling the environment after observed traces of existing file sharing networks.

While this scenario may be appropriate for many important applications, we believe that overlay networks can be important design abstractions in radically different scenarios that have not been considered by the P2P research community so far. In particular, massive joins to a large overlay network are not supported by known protocols very well, and many protocols have trouble dealing with massive departures as well. Other important, yet under- emphasized scenarios include bootstrapping a large network from scratch, merging two or more large networks, splitting a large network into several pieces, and recovering from catastrophic failure as close relatives of the first two.

If these scenarios were to be supported efficiently, we could build a fully open and flexible computing infrastructure that points well beyond current applications. In this paper we envision scenarios that involve (virtual) organizations with (possibly) large pools of resources organized in overlay networks. We want to allow these overlay networks to freely and flexibly merge and split on demand from networks of other organizations, and we want to admit assignment (or sale) of pools of resources for relatively short periods to users who could then build their own infrastructures on demand and abandon them when the time is up. This vision is in line with current efforts to enhance the flexibility of Grid infrastructures using P2P technology [4].

To support the above vision, we propose a P2P infrastructure that includes a dedicated bootstrapping service. To provide a reliable platform in the face of massive joins and departures, we propose the peer sampling service [6] as a lightweight bottom-most layer of our P2P architecture. Merging sev-
eral large networks or starting an application from scratch within its time-slice are unusual and radical events that many existing P2P protocols are not designed to cope with. In an architecture that supports bootstrapping, large collections of resources can be readily aggregated into global structured overlays rapidly and efficiently. This then allows the use of existing, well-tuned protocols [8, 11, 13, 17] without modification to maintain the overlays once they have been formed. As a concrete example of bootstrapping service, we present a novel protocol that can efficiently build prefix-based overlay routing substrates (and in some cases a ring) such as Pastry [13], Kademlia [8], Tapestry [17] and Bamboo [11] from scratch.

The outline of the paper is as follows. Section 2 presents the architecture to support the scenarios mentioned above. Section 3 briefly describes bottom layer: the peer sampling service. Section 4 describes the protocol implementing the bootstrapping of routing substrates, while Section 5 presents experimental results. Finally, Sections 6 and 7 compare our contribution to related work and conclude the paper.

2. The Architecture

Our ultimate goal is to design a P2P architecture that allows for large pools of resources to behave almost like a liquid substance: it should be possible to merge large pools, or split existing pools into several pieces easily. Furthermore, it should be possible to bootstrap potentially complex architectures on top of these liquid pools of resources quickly on demand.

Complex P2P architectures often involve structured overlay networks [2, 12], however, these overlays are not “liquid”: joining a large number of nodes, and especially jump-starting new overlays is very expensive. We have previously proposed the peer sampling service that fulfills the “liquidity” requirement [6], however, this service is rather basic: it provides nodes with random samples from the set of participating peers (see Section 3). Besides, the peer sampling service cannot directly support a number of important functions, such as efficient search.

In our proposed architecture (see Figure 1) we aim at combining the flexibility of peer sampling with the richness of applications offered by structured networks. To this end, we introduce two layers below structured overlays. The extremely robust lowest layer is the peer sampling service, which is responsible for dealing with scenarios where the set of participating nodes changes dramatically. Higher layers will simply start to get samples from the new set following the change, but other than that, the network will remain operational. On top of peer sampling, we include the bootstrapping service that is responsible for restoring, or creating from scratch, structured networks when necessary. The bootstrapping service should support existing protocols that manage structured overlays under normal circumstances.

This idea is feasible only if we can design an efficient and lightweight bootstrapping protocol that can construct a sufficiently rich set of structured overlays on demand. In this paper we propose such a protocol.

As shown in Figure 1, the architecture supports components other than structured overlays as well. For example, a number of components rely only on random samples, like probabilistic broadcast (gossip) or aggregation [3, 7]. The architecture can also support other overlays, such as proximity based ones [5, 15].

3. The Peer Sampling Service

The bottom layer of the proposed P2P architecture is the peer sampling service [6]. The purpose of this layer is to provide random peers from the set of participating nodes through the GETPEER method. In addition, the layer implicitly defines membership as being the pool from which the samples are drawn. Our previous work confirms that the layer is a source of high quality (i.e., sufficiently
random) samples, even immediately after massive joins and departures. This is essential for dealing with the scenarios described in the Introduction and allows us to focus on bootstrapping over a network on which the sampling service is already functional.

In this paper we implement peer sampling through the NEWSCAST protocol [6]. The basic idea of NEWSCAST is that each node periodically sends a small, locally available random set of node addresses to a member of this random set. After receiving such a message, the node keeps a fixed number of freshest addresses (based on timestamps). In the following we briefly summarize the most important properties of the protocol.

Cost: Each node sends one message to another node during a fixed time interval. Implementations exist in which these messages are small UDP messages containing approximately 30 IP addresses, along with the ports, timestamps, and descriptors such as node IDs. The time interval is typically long, in the range of 10 seconds. The cost is therefore small, similar to that of heartbeat messages in many distributed architectures.

Self-healing: The protocol can withstand catastrophic failures (up to 70% nodes fail) and massive churn. The protocol can recover from a wide range of initial states extremely quickly, in a few gossip rounds.

Due to its low cost, extreme robustness and minimal assumptions, this protocol is an ideal bottom layer that makes the bootstrap service possible. The sampling service is also useful (and, in fact, sufficient) for virtually all gossip-based protocols that are based on sending information periodically to random peers.

4. The Bootstrapping Service

On top of the sampling service we develop a protocol for building a prefix-based routing table from scratch. The key idea is to build a sorted ring, and during the process, collect entries to fill the prefix tables at all nodes.

The prefix table is defined as follows. We assume that all nodes have a unique numeric ID. An ID is represented as a sequence of digits in base $2^b$ — each digit is encoded as a $b$-bit number. The prefix table contains up to $k$ entries for all pairs $(i, j)$, where $i$ is the length (in digits) of the longest common prefix, and $j$ is the first different digit. The entries may be less than $k$ if there are not enough nodes with the desired prefix and digit. Many overlay routing substrates are based on this prefix table: for example Pastry [13], Kademlia [8], Tapestry [17] and Bamboo [11]. Using the generated prefix tables and the leaf sets that define the sorted ring, the routing tables of all these networks can be bootstrapped.

The protocol executed at all nodes is shown in Figure 2. Each node has a prefix table to fill and a leaf set, that is being evolved to contain the nearest neighbors in the sorted ring of node IDs. The size of the leaf set is $c$. The components of the protocol work as follows.

Method UPDATELEAFSET takes a set of node descriptors (addresses and corresponding IDs) and tries to improve the leaf set using these descriptors. First, it merges the set given as a parameter, and the current leaf set, and then sorts this set according to distance from the node’s own ID in the ring of all possible IDs. Note that all IDs can be classified as successors and predecessors: if an ID is closer in the increasing direction, it is a successor, otherwise it is a predecessor. Then, in an effort to collect an equal amount of successors and predecessors, the method attempts to keep an equal number $(c/2)$ of closest successors and predecessors. If there are not enough successors or predecessors, then the leaf set

```plaintext
1: for each $\Delta$ time units do
2:  $q \leftarrow$ SELECTPEER()
3:  $m_p \leftarrow$ CREATEMESSAGE($q$)
4:  send $m_p$ to $q$
5:  $m_q \leftarrow$ receive($q$)
6:  UPDATELEAFSET($m_q$)
7:  UPDATEPREFIXTABLE($m_q$)
(a) active thread

1: loop
2:  $m_q \leftarrow$ receive($*$)
3:  $m_p \leftarrow$ CREATEMESSAGE($q$)
4:  send $m_p$ to sender($m_q$)
5:  UPDATELEAFSET($m_q$)
6:  UPDATEPREFIXTABLE($m_q$)
(b) passive thread
```

Figure 2. Bootstrapping protocol at node $p$. 
is filled with the closest elements in the other direction.

Method \texttt{selectPeer()} also sorts the leaf set according to distance from the node’s own ID in the ring of all possible IDs, and then picks a random element from the first half of the sorted list.

These components of the protocol are similar to the application of T-MAN for building a sorted ring, as described in [5]. The rest of the components are responsible for building the prefix table. The basic idea is that the gradually improving prefix table is fed back into the ring building process, so that the two components mutually boost each other. The simplest method is \texttt{updatePrefixTable} that takes a set of node descriptors and fills in any missing table entries from this set.

The key component of the algorithm is \texttt{createMessage}, which is responsible for generating a set of node descriptors to be sent to the peer node. Knowing the ID of the peer, the method optimizes the information to be sent as follows. First it takes the union of the leaf set, \( c_r \) random samples taken from the sampling service, the current prefix table, and its own descriptor (in other words, all locally available information). It orders this set according to distance from the peer node, and keeps the first \( c_r \) entries. In addition, it adds to the message all node descriptors that are potentially useful for the peer for its prefix table (i.e., have a common prefix with the peer). The size of this additional part is not fixed but is bounded by the size of the full prefix table, and usually is smaller in practice.

Finally, the protocol needs to be started in a loosely synchronized manner, that is, the nodes have to start the execution of the protocol with a relatively small time difference, for example, within an interval of \( \Delta \) time units, which defines the frequency of communication, for instance, within an interval of \( \Delta \) time units, which defines the frequency of communication. Finally, \( c_r \) is the number of random samples used for optimizing the messages to be sent. Note, that these samples are “free” (if \( c_r \) is not too large) since the generic peer sampling layer is assumed to function independently.

5. Simulation Results

Both the sampling service and the bootstrapping service were implemented for the \texttt{PeerSim} simulator developed by us [10]. We focus on two aspects of the protocol: scalability and fault tolerance. To this end, we fix all the parameters of the protocol, except the network size. In our simulations, the id is a 64 bit integer. Although typical definitions of the id space start from 128 bit integers, using only 64 bit for simulation is not restrictive, because the length of the largest common prefix is much less than 64 bits for all pairs of nodes in a network of any practical size. The extra bits play no role in this protocol.

The parameters of the prefix table were chosen to match common settings: \( b = 4 \) and \( k = 3 \). For networks that do not require multiple alternatives of a given table entry, setting \( k > 1 \) is still useful because it allows for optimizing the routes according to proximity. The leaf set size was \( c = 20 \) and the parameter \( c_r \) was set to be 30.

The set of network sizes (\( N \)) we experimented with is \( 2^{14}, 2^{16} \) and \( 2^{18} \). The scenario of an experiment was as follows. We assumed that we are given a network on which the sampling service is already functional. We start the bootstrapping protocol at each node at a different random time within a time interval of length \( \Delta \). For convenience, we call the successive intervals of length \( \Delta \) cycles. The end of the first interval is the end of cycle 0.

The protocol is then run until the perfect leaf sets and prefix tables are found at all nodes, based on the actual set of id-s in the network. This cannot be decided locally, and indeed, the protocol has no stopping criterion. However, since it is cheap and needs only a small number of iterations, in practice it can simply be run until a fixed number of cycles after the initialization, that is sufficient for convergence.

To test scalability, in the first set of experiments (shown in Figure 3) all messages are delivered reliably. For the network sizes \( 2^{14}, 2^{16} \) and \( 2^{18} \) we performed 50, 10 and 4 independent experiments,
respectively. The plots show each individual experiment, ending when perfection is reached.

The most important observation we can make is that the speed of the collection of the leaf sets clearly shows only a constant increase when the network size is multiplied by the same factor of four. This is a strong indication that the time needed for convergence is logarithmic in the network size. Apart from being logarithmic, the actual convergence times are rather small as well. Besides, the convergence clearly shows an exponential character.

The convergence of the prefix tables is rather surprising. There is clearly a more complex relationship between the convergence time and the parameters of the protocol: the network of $2^{18}$ nodes converges faster in the final phase than the network that is four times smaller, with the same parameters. Note, that in this final phase the vast majority of the entries are already available (less than 1 out of 1000 links are missing), and this slight difference has to do with the scarcity of suitable id-s for the remaining positions to fill.

In the second set of experiments we tested the fault tolerance of the protocol by dropping messages with a uniform probability (Figure 4). This kind of failure is important to study because we designed the protocol with a cheap, unreliable transport layer in mind (UDP). The drop probability was chosen to be 20%, which is unrealistically large. Since the protocol is based on message-answer pairs, if the first message is dropped, then the answer is not sent either. Simple calculation shows that the expected overall loss of messages is 28%.

The main conclusion of these experiments is that the behavior of the protocol is very similar to the case when there is no failure, only convergence is slowed down proportionally.
6. Related Work

Massive joins to already running overlays have been addressed previously (e.g., [16] and [12]) proposing a form of periodic repair mechanism for maintaining the leaf set, not unlike the one presented here. Very recently the bootstrapping problem has been addressed as well focusing on specific overlays [1,9,14]. Our contribution with respect to related work is twofold. First, we propose an architecture that can support a protocol that jump-starts an entire overlay from scratch. Our protocol is independent of the protocol that manages the routing substrate: we singled out the abstract bootstrapping service as an important architectural component. Second, we present a protocol, that is efficient and lightweight, and that supports overlays based on prefix-tables and leaf sets.

7. Conclusions

The proposed architecture is motivated by our research into the peer sampling service, and applications and services that rely on it. Although the functionality of the sampling service is basic, its implementation is more robust and flexible than those of currently available structured overlays. The architecture we presented here, and in particular, the bootstrapping service, bridges the robustness and flexibility of the sampling service and the functionality of structured overlays.

According to the simulation results, the proposed bootstrapping protocol can build a perfect prefix table and leaf set at all nodes, in a logarithmic number of cycles, even in the presence of message delivery failures. This performance, in combination with the support of the sampling layer, enables the on-demand deployment of complex (multi-layered) P2P applications in short time-slices over large pools of resources, and allows for large pools of resources to be temporarily merged or separated as well.

References


