Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A machine-checked correctness proof for Pastry

Noran Azmy [a,b], Stephan Merz [b,*], Christoph Weidenbach [a]

[a] Max Planck Institute for Informatics, Saarbrücken, Germany
[b] University of Lorraine, CNRS, Inria, LORIA, Nancy, France

ARTICLE INFO

Article history:
Received 7 October 2016
Received in revised form 10 August 2017
Accepted 11 August 2017
Available online xxxx

Keywords:
Formal verification
Machine-checked proof
Distributed algorithm
Peer-to-peer protocol
Distributed hash table

ABSTRACT

Protocols implemented on overlay networks in a peer-to-peer (P2P) setting promise flexibility, performance, and scalability due to the possibility for nodes to join and leave the network while the protocol is running. These protocols must ensure that all nodes maintain a consistent view of the network, in the absence of centralized control, so that requests can be routed to the intended destination. This aspect represents an interesting target for formal verification. In previous work, Lu studied the Pastry algorithm for implementing a distributed hash table (DHT) over a P2P network and identified problems in published versions of the algorithm. He suggested a variant of the algorithm, together with a machine-checked proof in the TLA$^+$ Proof System (TLAPS), assuming the absence of node failures. We identify and correct problems in Lu's proof that are due to unchecked assumptions concerning modulus arithmetic and underlying data structures. We introduce higher-level abstractions into the specifications and proofs that are intended for improving the degree of automation achieved by the proof backends. These abstractions are instrumental for presenting the first complete formal proof. Finally, we formally prove that an even simpler version of Lu's algorithm, in which the final phase of the join protocol is omitted, is still correct, again assuming that nodes do not fail.

1. Introduction

In a peer-to-peer (P2P) network, individual nodes, or *peers*, communicate directly with each other and act as both suppliers and users of a given service. P2P networks are motivated by their self-organization, scalability and robustness, since there is no central server representing a single point of failure or a performance bottleneck.

A key problem with P2P networks—particularly large-scale ones—is how to efficiently manage the available resources. In completely unstructured P2P networks where no topology is imposed on the nodes, functions like search typically resort to flooding the network via broadcasting a search request until the request reaches a peer that has the required data item. Flooding causes a very high amount of unnecessary network traffic, as well as CPU and memory usage [1].

Distributed hash tables (DHTs) are a way of structuring P2P networks so that the available resources are organized, and communication among peers is reliable and efficient. A DHT implements a hash table where key-value pairs are stored at different nodes on the network. Nodes are assigned unique identifiers, and messages from one node to another—instead of being flooded through the network—are routed through a number of intermediate nodes, the route being determined by node identifiers. Distributed hash tables tap the advantages of both P2P communication and hash tables, with a simple

* Corresponding author.
E-mail address: stephan.merz@loria.fr (S. Merz).

http://dx.doi.org/10.1016/j.scico.2017.08.003
0167-6423/© 2017 Elsevier B.V. All rights reserved.

and elegant design that enables locating a required piece of data with high efficiency, and without the need for global information. As in a classic hash table, the main function of a DHT is key lookup. Due to the lack of a central server with a global view of the network, nodes in a DHT must collaborate to decide on their respective key storage range, and to route lookup requests to the appropriate node. Pastry, Chord, Kademlia, CAN and $\mathcal{DKS}$ are among the most popular published DHT protocols [2–6]. These protocols are similar in that they focus on the efficient management of data stored in a distributed fashion over a large number of nodes, but they differ in some characteristics such as the topology of the overlay network, the distance function between nodes on the network, and the routing model.

In most practical applications, P2P/DHT networks are subject to a certain level of *churn*; nodes are continuously joining and leaving the network, and they may fail abruptly without giving notice to other nodes. The DHT implementation should handle this turbulence efficiently and smoothly and ensure that the network always recovers to a stable state where connectivity among the live nodes is maintained and there is no confusion among the nodes about the key space. This aspect presents an interesting target for formal verification.

In this paper, we present the results of our formal verification of Pastry. We verify correct delivery of lookups for two variants of Pastry by giving two complete proofs of correctness written in the interactive proof assistant TLAPS [7]. It has already been shown in [8] that published variants of the protocol violate this correctness property: not only may node departures and failures may cause the network to separate irreversibly, but more surprisingly, the Pastry ring may even disorganize when new nodes join the network. Here, we show that a version of Pastry suggested by Lu [9] is correct with respect to delivery of lookup messages in a pure-join model, i.e., where no nodes may leave the network or fail. We also show that in the pure-join model, correctness still holds using a join protocol that is simpler than that used by Lu and originally proposed in [10].

1.1. Related work

Bakhshi et al. [11] describe an abstract model for structured P2P networks with a ring topology in the π-calculus, and use this model for verifying the stabilization algorithm of Chord by establishing weak bisimulation between the specification of Chord as a ring network and the implementation of the stabilization algorithm. This is a pure-join model in which node failure is not taken into account, and features such as finger (routing) tables and node successor lists are not modeled. Using Alloy to formally model and verify Chord, Zave [12] shows that the pure-join Chord protocol is correct, but that the full version of the protocol may not maintain the claimed invariants. In subsequent work [13] she presents a full version of Chord (where both node arrivals and departures are modeled) with a partly mechanized proof of correctness. The correctness of this version of Chord relies on the assumption that there is a *stable base* of $r+1$ permanent network members, where r is the size of the successor list maintained by each node. The authors of the protocol $\mathcal{DKS}$ conduct some experiments using simulation to observe how lookup efficiency is affected by churn [6]. In his Ph.D. thesis, Ghodsi [14] discusses several issues such as concurrent joins and node failure, and claims that it is impossible to guarantee correctness when node failure is possible, due to the possibility of network separation. Borgström et al. [15] use CCS for the formal verification of lookups in the static case of the protocol, i.e., without taking node joins or failure into account.

The work that is most relevant to this paper was done by Lu on Pastry [8,9,16]. Lu models Pastry in TLA$^+$, and uses the TLC model checker and the TLAPS proof assistant to formally verify *correct delivery* of lookups: *at any point in time, there is at most one node that answers a lookup request for a key, and this node must be the closest live node to that key*. As in the case of Chord, Lu discovers several problems in the original Pastry protocol. He also shows that the improvements proposed in later publications on Pastry, in particular by Haeberlen et al. [10], still do not guarantee correct delivery, even in the absence of node failures. Finally, he presents a pure-join variant of Pastry, which he calls LuPastry, for which he verifies correct delivery. Notably, Lu's Pastry variant restricts the protocol described in [10] by enforcing that a live node may only facilitate the joining of one newly arriving node at a time. Lu's proof reduces correct delivery to a set of around 50 claimed invariants, which are proved with the help of TLAPS. As such, LuPastry represents a major effort in the area of computer-aided formal verification of distributed algorithms. Due to the sheer size of the proof, however, as well as the lack of maturity of the tools at the time, Lu's proof relies on many unproved assumptions relating to arithmetic and to protocol-specific data structures. Upon examining Lu's proof, we discovered counterexamples to several of the underlying assumptions. While we were able to prove weaker variants of many assumptions, this was not possible for others. In fact, we were able to find a counterexample to one of Lu's claimed invariants, for which the TLA$^+$ proof was only possible because of incorrect assumptions. This led us to redesign the overall proof of correctness for Pastry. In the process, we introduced higher-level abstractions in the specification and the proof that help make the TLA$^+$ specification of the protocol more understandable and, importantly, also help improve the degree of automation of the proof.

Our improved specifications and the outline of the new proof were published in [17], and the present article is an extended version of that conference paper that contains a more detailed presentation of our contributions. Moreover, we observe that the node join process of the protocol can be simplified substantially, without impacting correctness: the invariants used for the proof reveal that the final "lease exchange" step, a handshaking step between a new node and its neighbor nodes before it becomes an active participant, is not necessary for correctness in the join-only scenario. In fact, this step was not part of the original Pastry protocol published in [2], but was introduced by Haeberlen et al. [10], among other improvements to the protocol. Although the reasons for adding the lease exchange step are not stated explicitly, one may suspect that it was introduced in order to improve the accuracy of the leaf sets and, consequently, the consistency of