Model Checking the Pastry Routing Protocol

Tianxiang Lu, Stephan Merz, Christoph Weidenbach

Electronic Communications of the EASST
Volume X (2010)

Proceedings of the 10th International Workshop on Automated Verification of Critical Systems (AVoCS 2010)

Model Checking the Pastry Routing Protocol

Tianxiang Lu, Stephan Merz, and Christoph Weidenbach

3 pages

Guest Editors: Jens Bendisposto, Michael Leuschel, Markus Roggenbach
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Model Checking the Pastry Routing Protocol

Tianxiang Lu[1,2], **Stephan Merz**[1], **and Christoph Weidenbach**[2]

[1] INRIA, Nancy, France
[2]Max-Planck-Institute for Informatics, Saarbrücken, Germany

Abstract: Pastry is an algorithm for implementing a scalable distributed hash table over an underlying P2P network, an active area of research in distributed systems. Several implementations of Pastry are available and have been applied in practice, but no attempt has so far been made to formally describe the algorithm or to verify its properties. Since Pastry combines rather complex data structures, asynchronous communication, concurrency, resilience to *churn* and fault tolerance, it makes an interesting target for verification. We have modeled Pastry's core routing algorithms in the specification language TLA$^+$ and used its model checker TLC to analyze qualitative properties of Pastry such as *correctness* and *consistency*.

Keywords: model checking, specification, verification of software

1 Pastry Routing Protocol

Pastry implements a distributed hash table (DHT) as a structured peer-to-peer overlay network. Mapping *object keys* (such as file identifiers) to *network nodes*, it provides a substrate for building distributed applications [CCR04]. At its core it offers a *lookup* primitive to route a message to the node responsible for a key. More precisely, keys are numbers from an ID space, arranged clockwise as a *virtual ring*. A node is assigned an ID from the same space and maintains a *routing table* and a *leaf set* to route lookup messages. The leaf set of a node contains the l closest neighbors of the nodes on either side (the *right* and *left set*), where l is a parameter of the protocol. The routing table supports routing to more distant nodes. When a new node joins, it constructs its leaf set based on those of its neighbors in the ring according to its ID and then obtains the ownership of a range of keys from its immediate neighbors. In order to cope with spontaneous departures of nodes, nodes periodically *probe* all nodes in their leaf set, exchanging the contents of their leaf sets and thus maintaining consistent local views of the network. Pastry is resilient to churn, i.e. the arrival and departure of nodes. Even under high churn rates, the protocol ensures that a message will eventually be answered by the owner of the key (*correctness*) and thus maintains a *consistent* DHT. Pastry relies on timeouts to achieve good routing performance and is resilient to intermittent failures by resending and rerouting messages if necessary.

2 TLA$^+$ model

We modeled Pastry in TLA$^+$ [Lam02] as a (potentially infinite-state) transition system. Our model is available on the Web[1]. It is structured as follows:

[1] http://www.mpi-inf.mpg.de/~tianlu/softwares/MSPastry-vE-2010-09-03.zip

- Several modules introduce the data structures used in Pastry. The module *Ring* declares basic parameters, such as the size of the leaf set l, and defines operations such as computing the distance between two nodes on the ring. The module *LS* models the leaf set as a tuple consisting of a node ID and two sets (representing the left and right set). The module *RT* models the routing table of a node as a mapping from *positions* fixed by its row and column in the table to the entry values. Both modules *LS* and *RT* define the fundamental operations over the data structures they represent, such as updating and removing entries. The module *Msg* introduces different types of messages that are used in the Pastry algorithm, such as *Lookup* containing a key and *JoinRequest* containing the node Id to join.

- The module *InitialStates* introduces the state variables representing the algorithm. In particular, these include the communication network, the status of nodes (e.g. *ready* or *dead*), their routing tables, leaf sets, as well as their local views of the liveness of neighbors. The communication network is represented as a set of messages in transit, corresponding to asynchronous message delivery where messages can be delayed and reordered.

- Module *Actions* contains the definitions for modeling the dynamics of the protocol. An action corresponds to an atomic transition step performed by some node in the network. It is defined by its enabling conditions and its effects on the local state of the node as well as the communication network (message sending and receiving). For example, the application request of looking up a key is modeled as an action without preconditions but with the effect of adding a *Lookup* message to the network. The spontaneous departure of a node is modeled by an action that sets that node's status to *dead* and reinitializes all its relevant variables.

- Module *MSPastry* contains the overall specification of Pastry based on the definitions of the initial state and the actions introduced before. It also defines linear temporal logical formulas representing the properties to be verified, including *correctness* and *consistency*.

- Module *MCMSPastry* explicitly adds actions corresponding to faults such as concurrent joining of two nodes and spontaneous departure of nodes as possible next steps to the specification for model checking. Currently, we do not consider link or message loss and abstract from the real-time aspects of the protocol.

3 Challenges

The first challenge in modeling Pastry was to determine an appropriate level of abstraction. For example, we decided to represent timing-dependent actions as occurring non-deterministically, sometimes guarded by non-local preconditions that represent assumptions about the relations between timing parameters. Thus, it is assumed that a node cannot remove its neighbor from its leaf set before the neighbor has stopped to receive messages. The second challenge was to fill in unstated details in the description of the Pastry algorithm, based on the counter-examples revealed by model-checking. For instance, it was not explicitly stated what it means for a leaf set to be "complete" or if an overlap between the right set and left set is allowed. We made explicit assumptions on how corner cases should be handled, sometimes based on an exploration

of the source code of the FreePastry implementation. Thus, we allow an overlap only if there are less than $2l$ nodes present in the entire network, and we allow an incomplete leaf set only if there are less than l nodes. In other cases, we established different models implementing alternatives based on different assumptions that appeared reasonable. A further challenge was to formulate the correctness properties themselves; in fact, these are not stated at all in [CCR04]. We introduced a notion of nodes being *stable* to distinguish nodes with a consistent local view of the DHT from those without. More precisely, a *stable* node has a complete leaf set, and we assume that only stable nodes handle lookup messages. The main correctness property that we are interested in is that the lookup message for a particular key is always answered by the numerically closest stable node.

4 Results and Future Work

We used TLC, the explicit-state model checker for TLA$^+$, to verify the correctness properties against our different versions of the Pastry model, instantiated to four nodes. After it had helped us in finding several modeling errors and revealing necessary details of the model, we ran the model checker against one of our models of Pastry, attempting to verify all properties we had defined. We stopped TLC after running it for days on two CPUs without finding any more counterexamples. It computed 251,189,726 distinct reachable states using breadth-first search to a depth of 21 (the number of actions applied subsequently starting from initial states) and left a queue of 150,865,719 states, whose successors were not yet computed.

The use of TLC helped us to quickly explore different alternatives and ask "what-if" questions, and in this way we produced different models corresponding to possible implementation choices, and formalize assumptions that ensure the consistency of the hash table (at least for the instances that we analyzed). As a next step, we expect to further validate our model in discussions with the designers of Pastry, and then produce formal proofs of the correctness of the protocol. A first step will be to prove the type invariant in the interactive TLA$^+$ proof system [**?**].

Bibliography

[CCR04] M. Castro, M. Costa, A. I. T. Rowstron. Performance and Dependability of Structured Peer-to-Peer Overlays. In *Intl. Conf. Dependable Systems and Networks (DSN 2004)*, pp. 9-18. Florence, Italy. IEEE Computer Society, 2004.

[CDL10] K. Chaudhuri, D. Doligez, L. Lamport, S. Merz. Verifying Safety Properties with the TLA$^+$ Proof System. In: J. Giesl, R. Hähnle, eds.: *5th Intl. Joint Conf. Automated Reasoning (IJCAR 2010)*. Springer LNCS 6173, pp. 142-148. Edinburgh, UK, 2010.

[Lam02] L. Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.