Formal Specification and Model Checking of the Distributed Consensus Protocol Raft

2030405 Takanori Ishibashi

In this master's research project report, we report on a case study in which Raft is formally specified in Maude and model checking experiments are conducted based on the formal specification. The model checking experiments say that Raft enjoys properties that it is expected to guarantee. Raft is widely known as one of the distributed consensus protocols and is used to build highly available and strongly consistent services. Raft divides a distributed consensus problem into two independent sub-problems: leader election and log replication. In the leader election, Raft chooses at most one leader in each logical time called a term. There is one and only one leader in a Raft cluster in regular operations and all the other servers are then followers. In the log replication, the leader accepts requests from clients, saves such requests in its log, and forwards them to all the other servers. On receipt of such requests, each server saves them in its log. When the leader receives positive replies for a client request from the majority of servers, it commits (or consents to) the request. Each server has a state machine in which clients' requests are processed. When a follower receives a message saying that a client request has been committed, the follower commits the clients' request up to the client request (inclusive).

In this master's research project report, we concentrate on the leader election and the log replication, which are basic mechanisms in Raft. We formally specify the leader election and the log replication in Raft using Maude, which is a rewriting logic-based specification/programming language. In the leader election, we model check with Maude that Raft enjoys the Election Safety Property. In the log replication, we model check with Maude that Raft enjoys the Log Matching Property and the State Machine Safety Property. The Election Safety Property is that at most one leader can be elected in each logical time. The Log Matching Property is that if two logs contain an entry with the same index and term, then the logs are identical in all entries up through the given index. The State Machine Safety Property is that if any two servers have applied two entries to their state machines at a same index, the two entries must always be the same. The first property is expressed as an invariant property of the state transition system formalizing the leader election, and the last two properties are expressed as invariant properties of the state transition system formalizing the log replication. Maude is equipped with a linear temporal logic (LTL) model checker and a reachability analyzer (called the search command) as model checking facilities. The

search command can be used as an invariant model checker. Because the three properties are invariant properties, we use the search command for the model checking experiments. In the leader election, our model checking experiments show that the protocol enjoys the Election Safety Property under the condition that we limit the logical time and the number of servers. In the log replication, our model checking experiments show that the protocol enjoys the Log Matching Property and the State Machine Safety Property under the condition that we limit the length of the server's log and the number of servers.

We assume that a server in a Raft cluster conducts unexpected operations, which is different from the log replication in Raft. A server failure can result not only in a simple shutdown, but also in incorrect behavior. It is preferable to be able to handle the latter as well. Our model checking experiments also show that servers except for a server that conducts unexpected operations enjoy the the Log Matching Property and the State Machine Safety Property.

keyword: distributed consensus protocols, invariant properties, Maude, model checking, Raft, search command, state transition systems