An Investigation of the Chandy-Lamport Distributed Snapshot Algorithm and its Model Checking

Wenjie Zhang (1310043)

School of Information Science,
Japan Advanced Institute of Science and Technology

February 12, 2015

Keywords: distributed snapshot, distributed system, state machine, model checking, reachability property.

Concurrent and distributed systems have attained remarkable achievements in the past decades and are being widely used in our real life. Telecommunication networks such as wireless sensor networks, network applications such as world wide web and banking systems, real-time process control systems such as aircraft control systems and industrial control systems, and cloud computing systems are all distributed systems. Since the design of such distributed systems is generally complex, with a high possibility that subtle errors will cause erroneous behavior, such systems may crash, and we need to recover them if that is the case.

Many problems in distributed systems such as stable property detection and checkpointing can be cast in terms of the problem of detecting global states. A stable property is one that persists: once a stable property becomes true it remains true thereafter. Examples of stable properties are "computation has terminated," "the system is deadlocked" and "all tokens in a token ring have disappeared."

The global state of a distributed system consists of the states of every process and every channel in the system, where the state of a process is characterized by the state of its local memory and depends upon the context, and the state of a channel is characterized by the sequence of messages *"in-transit"*, those that have been sent on that channel, but not yet received by its destination process.

For a global state to be meaningful, the states of all the components of the underlying distributed system ($\mathcal{UDS}$) must be recorded at exactly the same instant. This will be possible if the local clocks at processes were perfectly synchronized or there was a global system clock that could be instantaneously read by the processes. However, given the fact that distributed systems are asynchronous and processes in the system do not share common clocks or memory, each process cannot record its local state at exactly the same time, namely that such global states can never be instantaneously done, and it leaves open the possibility of inconsistent global states. Moreover, the variability in message delays could lead to these separate processes constructing different global states for the same computation. In a word, due to the asynchrony of distributed systems, the lack of globally shared memory, global clock and unpredictable message delays make recording such consistent global states non-trivial. The global states we obtained may be inconsistent if we record the state of each component (process or channel) in the system whenever we want.

However, it turns out that even if the state of all the components in a $\mathcal{UDS}$ has not been recorded at the same instant, such a state will be meaningful provided every message that is recorded as received is also recorded as sent. Basic idea is that an effect should not be present without its cause. A message cannot be received if it was not sent; that is, the state should not violate causality. Such states are called *consistent global states* and are meaningful global states. Inconsistent global states are not meaningful in sense that a $\mathcal{UDS}$ can never be in an inconsistent global state.

Therefore, it is necessary to apply some algorithms when we want to obtain consistent global states of a distributed system. One of those algorithms is known as the Chandy-Lamport Distributed Snapshot Algorithm ($\mathcal{CLDSA}$), which was proposed by Chandy and Lamport in 1985.

The $\mathcal{CLDSA}$ can be used to determine consistent global states of a distributed system during its computation. Since it is very important and also non-trivial, it deserves to be formally specified and verified with respect to (w.r.t.) some significant properties. Let s_1 (called the start state) be the state when the $\mathcal{CLDSA}$ starts (a distributed snapshot starts being taken), s_* be the snapshot, and s_2 (called the finish state) be the state when the $\mathcal{CLDSA}$ terminates (the snapshot completes being taken). The $\mathcal{CLDSA}$ should enjoy the property that s_* is always reachable from s_1 and s_2 is always reachable from s_*. The property is called the Distributed Snapshot Reachability ($\mathcal{DSR}$) property, which guarantees the $\mathcal{CLDSA}$ takes consistent global states of a distributed system.

To formally verify the $\mathcal{DSR}$ property, model checking, which is an automatic verification technique for finite-state concurrent systems, can be used. It has been practically used in hardware industry, while many studies have been actively conducted on model checking so that model checking can be effectively and practically used for software.

The process of model checking comprises three main tasks: modeling, specification and verification.

Modeling is to convert a system that is to be reasoned about into a formalism accepted by a model checking tool. State machine can be used to model distributed systems. It consists of a set of states and a set of state transitions (i.e., a binary relation over the states). In such a model, the system is in one of the possible states, and the transition relation describes how the system moves from one state to another.

Specification is to state the properties that the system must satisfy before verification. These are written in a specification language, usually defined in a logic-based formalism. Completeness is one of the important issues in specification. Model checking provides means for checking that a model of the system satisfies a given specification, but it is impossible to determine whether the given specification covers all the properties that the system should satisfy.

Verification is to check the validity of the properties that have been stated previously. Ideally it is completely automatic. However, in practice it often involves human assistance such as the analysis of the verification results. In case of a negative result, the user is often provided with an error trace. This can be used as a counterexample for the checked property and can help the designer in tracking down where the error occurred.

Model checking refers to the following problem: Given a model of a system, exhaustively and automatically check whether this model meets a given specification. The main challenge in model checking is dealing with the *state explosion* problem caused by the fact that the state machine represents the *state space* of the system under investigation, and thus it is of size *exponential* in the size of the system description. Therefore, even for systems of relatively modest size, it is often impossible to compute their state machines.

There have been several major advances in addressing the *state explosion* problem. One of the first major advances was *symbolic model checking with binary decision diagrams* (BDDs). In this approach, a set of states is represented by a BDD instead of by listing each state individually. The BDD representation is often exponentially smaller in practice. Model checking with BDDs is performed using a *fixed point* algorithm. Another major advance is the *partial order reduction*, which exploits independence of actions in a system with asynchronous composition of processes. A third major advance is *counterexample-guided abstraction refinement*, which adaptively tries to find an appropriate level refinement, precise enough to verify the property of interest yet not burdened with irrelevant detail that slows down verification. Finally, *bounded model checking* exploits fast Boolean satisfiability (SAT) solvers to search for counterexamples of bounded length.

Maude, an algebraic specification language originated from OBJ family, is based on rewriting logic that includes as a sub-logic membership equational logic (an extension of order-sorted equational logic). Maude supports rewriting modulo equational theories such as associativity (`assoc`), commutativity (`comm`), and identity (`id`). Basic units of Maude specifications are modules such as `BOOL` and `NAT` used for boolean values and natural numbers. State machines (or transition systems) are specified in rewriting logic, and their specifications are called system specifications. Data used in state machines are specified in membership equational logic. States of state machines are expressed as tuples and associative-commutative collections (called soups), and state transitions are described in rewrite rules.

The $\mathcal{CLDSA}$ is a non-trivial distributed algorithm that deserves to be formally specified and verified w.r.t. the $\mathcal{DSR}$ property.

As far as we have investigated, to formalize the $\mathcal{DSR}$ property, we have to consider two kinds of states, (1) the states of a $\mathcal{UDS}$, and (2) the states of the $\mathcal{UDS}$ superimposed by the $\mathcal{CLDSA}$. In existing temporal logics such as LTL and CTL, only one kind of states are considered when they are used to formalize system properties. Thus, it is not straightforward to express the $\mathcal{DSR}$ property in LTL and CTL.

Moreover, there is an existing study in which a distributed system superimposed by the $\mathcal{CLDSA}$ has been formally specified in Maude and model checked w.r.t. the $\mathcal{DSR}$ property with the Maude search command. We do not, however, think that the existing study provides the sufficiently good foundation backing up that the $\mathcal{CLDSA}$ is surely model checked w.r.t. the $\mathcal{DSR}$ property, because the authors did not discuss whether the property is faithfully expressed or not. And then the $\mathcal{DSR}$ property encoded in the Maude search command are neither readable nor comprehensible. To make it executable in Maude, moreover, the system superimposed by the $\mathcal{CLDSA}$ has been specified in a very concrete way, in which the state of each process only depends on the tokens owned by the process itself. We do think that it is necessary to make sure that the property is faithfully expressed to claim that the property is model checked for the $\mathcal{CLDSA}$.

This research aims to investigate the $\mathcal{CLDSA}$, its formal specification in Maude and its model checking with the Maude search command, and to conduct some model checking experiments with several different underlying distributed systems. Moreover, to complement the existing study, we have considered how to surely model check the $\mathcal{DSR}$ property. To this end, we have already found a way to faithfully express the $\mathcal{DSR}$ property. Our way to express the property relies on two state machines, although the two state machines are closely related. And the property used in the existing study relies on only one state machine. Our way to express the $\mathcal{DSR}$ property has been affected by the Chandy-Misra's.