Scalable Verification of Distributed Systems Implementations via Messaging Abstraction*

Can Arda Muftuoglu†, Habib Saissi‡, Péter Bokor††, and Neeraj Suri
Technische Universität Darmstadt
{arda, habib, pbokor, suri}@cs.tu-darmstadt.de

1 Approach Overview

Motivation A number of verification approaches attempt proving liveness and safety properties in distributed system models and the more complex target of ensuring these in actual implementations. A typical approach is to verify a simplified model of the system and argue, mostly informally, that this result applies to the implementation, i.e. the unmodified system. Hence, the implementation of the system is not verified and can contain unrevealed bugs.

One major challenge of verifying a distributed system is to capture its (global) state, which is hard, or impossible, in general. Current approaches apply snapshot algorithms to capture slices of the system state [7]. As this considerably complicates the design of the verifier, the complexity of distributed snapshot can be mitigated by mapping processes running on different (and physically remote) machines into processes running within a single machine [5].

Even if capturing the global state can be done efficiently, the verification of distributed systems suffers from state space explosion due to concurrency and possible faults. Also, the unmodified system containing implementation details further increases complexity. As a result, the verification of unmodified distributed systems has so far been limited to debugging [7] or to small systems [3].

Proposing messaging abstraction We propose to decouple the distributed system operations into (a) (system specific) process-level operations and (b) (system independent) communication elements. As a result, the separated communication layer can be substituted with abstractions that are more amenable to verification. In the context of message-passing systems, we refer to this approach as messaging abstraction. Messaging abstraction also enables fast prototyping new systems with different communication models. For example, the messaging abstraction can be easily changed from reliable FIFO channels to lossy channels with possible out-of-order message delivery. Such changes can be tedious with real implementations.

Results overview In the following case study, we substantiate our approach by model checking implementations of complex state machine replication (SMR) protocols, namely Paxos [6] and Zab [4]. We implement the concept of messaging abstraction within the MP-Basset model checker [8]. Our experiments show that, thanks to messaging abstraction, MP-Basset is able to quickly disprove false properties of both protocols. In particular, our approach was able to identify the failure of Paxos SMR to preserve the order of user operations as submitted to the states machine, and also Zab to ensure liveness.

2 A Case Study

MP-Basset is a model checker for message-passing systems [8]. Based on the Java Pathfinder model checker, MP-Basset allows the local program of a process to be written in Java. Messaging abstraction in MP-Basset is simply a queue of messages where each message is a triple of the sender, the recipient, and the content of the message. Processes use two primitives for sending and receiving messages through the messaging abstraction layer. Firstly, a process can call the predefined method $send(\ldots)$ to send a message. Secondly, if a message is available in the queue, the recipient process can specify a condition (or guard) as of when the message is consumed. The condition depends on the content of the message and the local state of the process.

System examples: Paxos SMR & Zab We consider two state machine replication (SMR) protocols and model check them using messaging abstraction. We choose SMR for its complexity and generality, given that SMR can be used to replicate an arbitrary service. The first SMR protocol is based on the Paxos consensus algorithm [6]. The second protocol is using the Zab atomic broadcast algorithm [4]. Zab is part of the Zookeeper protocol suite.

*Demo with MP-Basset [8] is available at the conference.
†Student.
‡Presenting author.
which is a general coordination service used at Yahoo. We use our prototype Java-based implementation of Paxos SMR and Zab. Although a Java-based implementation of Zab exists, this implementation is part of a Java-based Zookeeper implementation\(^1\) and the code of Zab could not be trivially extracted.\(^2\) Our implementation of both protocols is available online\(^3\).

SMR is specified via the usual agreement and liveness properties. Agreement requires that every replica executes the same sequence of user operations. SMR is live if every user operation will be executed. Both protocols should satisfy agreement assuming that faulty processes fail by crashing. None of the protocols guarantees liveness if a special process, called the leader, crashes. Zab specifies an additional property, which we call FIFO order: Operations must be executed in the same order as they are issued by the user. It is known that Paxos SMR can violate the FIFO order even if FIFO channels are used [4].

**Model checking results** Our model checking results are depicted in Table 1. Each row corresponds to model checking a protocol and a property. Every experiment measures the number of states visited by the model checker and the time of model checking. The result of the experiment is CE if a counterexample of the property is returned or timeout if model checking runs out of resources (No CE found). The experiments run on DETERlab machines\(^4\) with Dual Xeon processors and 2 GB of memory.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Property</th>
<th>Result</th>
<th>States</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxos SMR</td>
<td>Agreement</td>
<td>No CE found</td>
<td>10,845,128</td>
<td>&gt;96h</td>
</tr>
<tr>
<td></td>
<td>FIFO order</td>
<td>CE found</td>
<td>1,216</td>
<td>1m26s</td>
</tr>
<tr>
<td>Zab</td>
<td>Agreement</td>
<td>No CE found</td>
<td>7,864,131</td>
<td>&gt;96h</td>
</tr>
<tr>
<td></td>
<td>FIFO order</td>
<td>CE found</td>
<td>3,861</td>
<td>1m42s</td>
</tr>
</tbody>
</table>

Table 1: Our messaging abstraction results. *With the assumption that agreement and FIFO order hold.*

We summarize our results as follows:

- **Efficient debugging** Messaging abstraction enables efficient debugging. This is justified by MP-Basset disproving false properties for both protocols within a few minutes, without user intervention.

- **Counterexamples** When used with model checking, messaging abstraction enables finding compact counterexamples of false properties. The purpose of the FIFO order experiment for Paxos SMR was also to reproduce the counterexample from [4], which contains 4 user operations. MP-Basset returns a smaller counterexample with 3 operations.

Although the current version of MP-Basset only supports safety properties such as agreement or FIFO order, we can use MP-Basset to prove that Zab violates liveness: MP-Basset finds a run with three subsequent operations \(R_1, R_2, R_3\) issued by the user where the sequence \(R_1, R_3\) is executed by the state machine because the leader crashes after proposing \(R_2\). Under the assumption that Zab satisfies agreement and FIFO order, this run is a counterexample of liveness. Otherwise, either \(R_2\) is executed after \(R_3\) (FIFO order violation) or there is a replica executing \(R_1, R_2, R_3\) (agreement violation).

### 3 Work-in-Progress

Motivated by the promising debugging results, we plan to expand messaging abstraction to enable exhaustive state space exploration of complex distributed systems. To measure the efficiency of messaging abstraction, we do not use further optimizations in our experiments. For example, symmetry and partial-order reductions are known to be efficient for distributed systems [1, 2, 7]. In addition, given the component-based architecture of these systems, forms of modular verification, for instance recent results of interface reductions [3], are also promising optimizations that are applicable with messaging abstraction. Given its flexibility, we expect that messaging abstraction gives rise to highly efficient applications of the above reductions. However, the trade-offs and achieved efficiency of these reductions are yet to be explored.

### References


