Reconsidering Reliability in Distributed Actor Systems

Raphael Hiesgen Dominik Charousset Thomas C. Schmidt
Dept. Computer Science, Hamburg University of Applied Sciences, Germany
{raphael.hiesgen,dominik.charousset,t.schmidt}@haw-hamburg.de

Abstract

Frameworks inspired by the actor model have recently attracted much attention. Actor systems promise transparent concurrency and distribution by combining message passing with a strong failure model. In this work, we re-examine distribution transparency and find that reliability breaks the promise in several dimensions. Solutions for regaining awareness of failures are briefly discussed.

Categories and Subject Descriptors C.2.4 [*Distributed Systems*]: Distributed applications

Keywords Actor model, message passing, distribution transparency, failure detection

1. Introduction

The actor model [1] seamlessly integrates concurrency and distribution. Its 'actors' solely communicate via *network transparent* message passing using a strong failure model. In reaction to a message, an actor can send messages, create new actors, or set its future behavior.

The C++ Actor Framework (CAF) [2] is a lightweight implementation of the actor model in modern C++ that combines a high level API with an efficient message passing layer. Like in most actor systems its behavior under distribution differs from local concurrency. For example, message passing to remote actors is unreliable while messages are reliably delivered to local actors. In general, reliability in distributed systems is a major source of complexity and much harder to achieve than in local regimes.

In this work, we reconsider reliability aspects of distributed actor systems. In search for enhanced transparency, we question whether a lightweight system can provide the same behavior for distributed as for concurrent scenarios, or whether the runtime system can at least detect when these behaviors deviate.

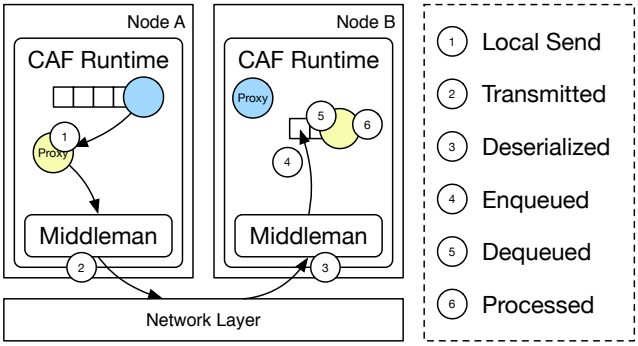

Figure 1. The steps that affect the reliability of messaging

Distributed message passing in CAF proceeds in several steps, each prone to specific errors (see Figure 1). The first step to pass a message is a synchronous local operation (1), which can only fail due to limited memory resources. For messages to remote actors, a proxy transparently forwards the messages to the middleman (MM). After serialization (2), the MM resolves the address of the receiver and transmits the message. Messages can get lost during transport, be duplicated, or change order. The MM on node B deserializes the message (3) if enough memory is available. Then, the message is enqueued into the mailbox of the receiver (4) provided it has enough space. When the receiver is scheduled and its mailbox contains no messages that arrived previously or have a higher priority, it dequeues the message (5) and processes it (6).

The following section discusses the core constituents of reliable message passing: message delivery, order preservation, and the detection of failing actors. We analyze the discrepancies of distributed scenarios compared to concurrent ones. Finally, § 3 concludes with an outlook.

2. Reliable Message Passing

2.1 Message Delivery

Message delivery in concurrent scenarios depends on a single synchronous enqueue operation into the mailbox of the receiving actor. Mailboxes are lock-free many-writer-single-reader queues. Sending a message in a concurrent scenario guarantees that it is enqueued into the mailbox of the receiver. Moreover, actors are free to skip or drop messages they receive. When the receiver fails, it remains unclear

whether the message has already been processed as explicit feedback is implementation dependent.

The same reliability for remote message passing requires messages to reliably reach the mailbox of the remote actor—step (4) in Figure 1. As sending a message is an asynchronous operation, the runtime environments (REs) are responsible for message handling. Simply relying on transport guarantees such as from TCP is thus insufficient. Beyond message transport, runtime failures such as memory allocation errors can occur on the receiving node. Systems that allow message routing via intermediate nodes additionally have to take failures after the first hop into account.

A straight-forward solution to reliable message passing is granted by an acknowledgement and retransmission mechanism. However, this is an expensive operation, as the sender needs to maintain a timer state per remote peer and has to buffer each message until reception is acknowledged by the remote actor. A lighter way to transparency is implicitly monitoring the liveliness of the remote nodes, which return (and forward) error messages in case of runtime failures.

2.2 Message Ordering

Ordering carries information about the relationship of messages, supports reasoning about the program flow, and eases debugging. For example, if messages sent by sequential statements are delivered in order, reading code and considering side effects is easier. A reliable order can also be used as a basis for more complex algorithms.

The synchronous characteristic of local sends does not only create a FIFO ordering between pairs of actors but extends to causal ordering. The logical relationship of causal ordering is described by the "happened before" relation [3]. Since actors are free to manipulate the order in their mailbox, ordering remains only valid until messages are enqueue into a mailbox.

The default delivery order for distributed scenarios is non-deterministic. Deploying causal order is very expensive and imposes either latency, using synchronous communication or fixed routing topologies, or additional message overhead by vector timestamps and the number of transitive dependencies [4].

Including a single vector timestamp in each message is sufficient to detect violation of causality. However, including the time vector that contains all actors in messages is not a feasible solution. As causality is guaranteed in concurrent scenarios, it might suffice to use a time vector with the logical clocks of nodes for this purpose. Still, synchronization impact imposed by the clock access remains.

While a failing delivery can be signaled back to the sender, a violation of order occurs without responsible party. It could only be signaled as a global error. The cost of causality leads some actor implementations to remain with FIFO ordering, which can be implemented comparably cheap, although the local ordering might be stronger.

2.3 Failure Detection

Actors that exit abnormally on a local node are detected by the RE that sends an `EXIT` message to all linked actors and a `DOWN` message to monitoring ones.

The extension to distributed scenarios requires the RE to track a list of remote links and monitors for notification. The reliable receipt of these failure messages depends on the reliability of the message delivery. There are two additional failure cases to track: link and node failures. To a remote observer, both cases are indistinguishable from each other, and from a very slowly responding node. The RE tracks these failures separately, for example through heartbeat messages.

The physical deployment at runtime should be transparent to actors. The RE supports this by mapping partial failures of the system to individual actor failures. Thus, actors do not have to handle a new category of errors. In this model, nodes are considered transparent containers for a set of actors. Actors might move between nodes depending on the implementation, in which case the lifetime becomes independent from the lifetime of the original node.

3. Conclusion and Outlook

Reliability assurances in current actor systems diverge between concurrent and distributed scenarios. This discrepancy motivates a reconsideration of reliability aspects. In this work, we examined message delivery, message ordering and failure detection as all three have weaker guarantees in distributed scenarios. While the reliability guarantees for delivery and failure detection can be aligned at reasonable cost, causal ordering is more expensive to achieve and introduces stronger coupling. We identified further research directions on how these aspects combine at low performance impact.

Our work is part of a redesign of CAF's network layer. Further aspects to consider are reachability, rendezvous, scalability, and security. These dimensions pose heterogeneous demands and we face the challenge of identifying a unified solution space. Our goal is an efficient network layer that recuperates transparency, scales up to Internet-wide distribution and down to low-power networks of things.

References

[1] Hewitt et al. A Universal Modular ACTOR Formalism for Artificial Intelligence. *Proc. of the 3rd IJCAI*, pp. 235–245, San Francisco, CA, USA, 1973.

[2] Charousset et al. Native Actors – A Scalable Software Platform for Distributed, Heterogeneous Environments. *Proc. of the 4rd ACM SIGPLAN Conf. SPLASH*, Workshop AGERE!, ACM, New York, NY, USA, 2013, pp. 87–96.

[3] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[4] F. Adelstein and M. Singhal. Real-time Causal Message Ordering in Multimedia Systems. *15th Intern. Conf. on Distr. Computing Systems*, ICDCS '95. Washington, DC, pp. 36–43.