A Middleware for Transparent Group Communication of Globally Distributed Actors

Dominik Charousset, Sebastian Meiling, Thomas C. Schmidt
Hamburg University of Applied Sciences, Department Informatik

Matthias Wählisch
Freie Universität Berlin

{dominik.charousset,sebastian.meiling}@haw-hamburg.de {t.schmidt,waehlisch}@ieee.org

ABSTRACT

Actors have been designed for loosely coupled concurrent systems based on asynchronous message passing. This model is of particular relevance for Internet-wide distribution, but cannot unfold its full potential due to the lack of a globally available messaging service for groups. We present a message-oriented publish/subscribe middleware that enables global group communication at near-IP performance. Based on this transparent group layer, we build `libcppa`, an Actor library with modular support for group semantics that is compliant to the new C++ standard.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*Distributed Systems*; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Group Communication Middleware, Actor Multicast Model

1. INTRODUCTION

Social networks, multiplayer games, and Smart Grids, in fact numerous distributed applications of today implement some group semantic such as *Anycast* for load balanced or replicated services, *(selective) Broadcast* for rendezvous processes or contacting unknowns, *Con(verge)cast* for data aggregation or scalable many-to-one communication, *Multicast* for scalable (m)any-to-many communication.

The publish/subscribe paradigm is most widely used for group communication, but distinctly implemented on several layers and communication scopes. There are event-based software designs such as Signal & Slot implementations in `C++`, D-Bus for system-wide event handling, as well as broadcast and multicast in overlay networks and in IP. Each technique requires a specific code access, thus forcing developers to decide on deployment at coding time. The divergent state of deployment for IP and overlay multicast (OM) increases the barrier to stable, scalable programming even further.

The Actor model [3] is a formalism describing concurrent entities that communicate by asynchronous message passing without making a distinction of communication type or technology. In today's mainstream languages, the publish/subscribe paradigm provides no common interface. Each technology defines its own API. The Actor model could make a contribution to unify and simplify group communication due to its message oriented programming style.

In this paper we make the following two contributions. We extend the Actor model by a general group semantic (§ 2) and introduce the underlying general group communication middleware along with its brief evaluation (§ 3).

Middleware Posters'2011, December 12th, 2011, Lisbon, Portugal.

2. GROUP COMMUNICATION AND THE ACTOR MODEL

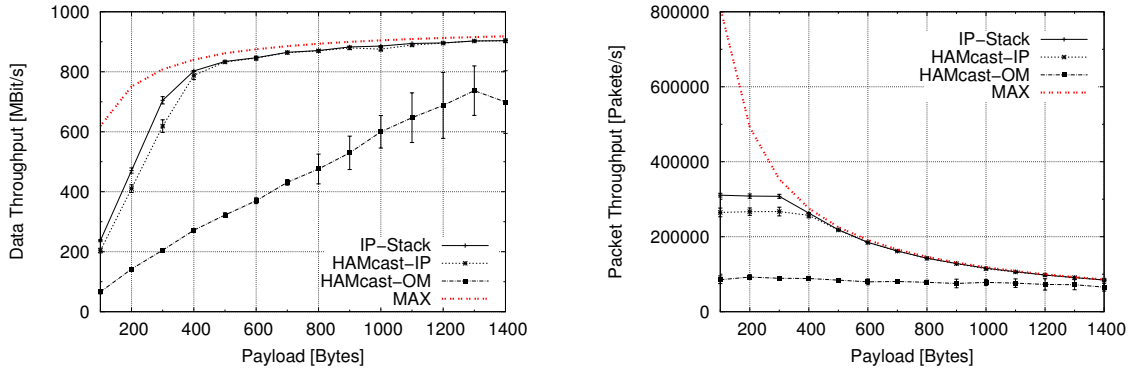
Today's most most widely used sequential programming languages such as `C++` and `Java` neither provide concurrency semantics nor a technology-transparent group communication interface. They grant low-level primitives such as threads and locks and interfaces to group technologies. However, scalable, concurrent applications are hard to build upon low-level threading primitives, because such software is difficult to verify and it is practically impossible to test whether it is free of race conditions and deadlocks [2].

The Actor model follows a complementary approach by replacing implicit communication through sharing with an explicit mechanism of messaging. Because Actors are isolated and do not share memory, they can be transparently distributed while race conditions are avoided by design. A lightweight Actor implementation that schedules all Actors in a properly pre-dimensioned thread pool can outperform equivalent thread-based approaches [1]. A full distribution of actors, though, relies on a transparent mapping of the message passing mechanism to the network layer.

Available implementations of the Actor model, such as Erlang, provide loose coupling based on a name service for Actors. However, distributed publish/subscribe event systems require loose coupling based on group names.

We extended the classical Actor model by a general group semantic and implemented this in a library called `libcppa` that is compliant to the new `C++` ISO standard. `libcppa` allows Actors to subscribe to groups and send (publish) messages to groups, as long as the sending Actor is a valid data source in the groups context.

In detail, the abstract class `group` provides an interface for the publish/subscribe paradigm. This class is implemented



(a) Data Throughput at Sender

(b) Packet Throughput at Receiver

Figure 1: Communication Performance of the HAMcast Middleware versus Native IP at 1 Gbit/s Link

in group communication modules, that allow access to a specific technology or communication scope. A static member function `get(module_name, group_id)` grants access to the module *module_name* and returns a singleton that manages access to the group identified by its *group_id*. For example, `group::get("local", "User Events")` returns a handle to a group for in-process communication that might be used for GUI relevant events.

Actors may receive messages from multiple sources simultaneously. A source can be a group that was previously joined, or another Actor. Such transparency in message handling reduces programming overhead and complexity of implementing distributed systems, while the unified group communication access allows developers to join groups with different communication scopes without additional effort.

However, distributed group communication equally relies on a proper distribution network. Facing the heterogeneous deployment of multicast service in today's networks, an implementation like `libcppa` alone cannot unfold its full capabilities, but requires additional system or network support.

3. THE COMMUNICATION MIDDLEWARE

Our middleware implements a universal publish/subscribe group communication service. It represents an abstraction layer between applications and various transport technologies, offering a globally available multicast service via a common API [4] that operates on transparent URI-based group identifiers like "`sip://news@cnn.com`". The abstraction layer enables an Loc/ID split, using mapping functions.

The HΛMcast architecture uses Inter-Domain Gateways (IMG) to connect multicast islands and span a global multicast service network. Basically, if an IMG was found in a local, multicast enabled IP network, HΛMcast uses the much more efficient native multicast service, otherwise a scalable overlay technology such as Scribe is used.

The middleware has a plug-in architecture for multicast technologies. In our prototype, we implemented IPv4, IPv6 and Scribe modules. During startup, the middleware performs a service discovery to detect network interfaces and multicast capabilities in the directly attached networks. After the startup phase, the service selection chooses the most efficient technology module.

Client applications based on libhamcast (a `C++` interface for the HΛMcast multicast service) use technology independent multicast socket stubs that forward method calls to the middleware on runtime via inter-process communication (IPC). Applications based on libhamcast are completely decoupled from any technology dependent API. Native sockets and group subscription for all known technologies are managed by the middleware modules.

3.1 Evaluation

Our brief evaluation of the HΛMcast middleware performance concentrates on the throughput in comparison to the native IP stack of the hosting node. Our set-up consists of a local network connecting nodes at homogeneous 1 Gbit/s links. A sender submits data at maximal capacity and receivers strive to process whatever arrives, passing it to the upper layer. We vary packet sizes, as the frequency in packet processing characterizes complexity.

Fig. 1(a) compares the data throughput at the sender, while Fig. 1(b) displays packet reception of listeners. In both cases, the HΛMcast IP processing approximates the native IP stack performance with minor flaws only for small packet sizes (< 400 Bytes). Overlay multicast (OM) distribution clearly falls short, on average by 40 %, due to overhead of additional packet headers and overlay routing performed in the middleware.

These results indicate negligible performance impact imposed by the HΛMcast middleware prototype. We conclude that this work may successfully serve as a proof of concept for establishing global communication patterns by a system-centric middleware.

4. REFERENCES

[1] P. Haller and M. Odersky. Scala actors: Unifying thread-based and event-based programming. *Theor. Comput. Sci.*, 410(2-3):202–220, 2009.

[2] P. B. Hansen. *Operating system principles.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1973.

[3] C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. In *IJCAI*, pages 235–245, 1973.

[4] M. Wählisch, T. C. Schmidt, and S. Venaas. A Common API for Transparent Hybrid Multicast. IRTF Internet Draft – work in progress 03, IRTF, July 2011.