FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Diplomarbeitspräsentation

compilers languages

Concurrent Programming with Actors and Microservices

Masterstudium:

Software Engineering & Internet Computing

Maximilian Irro

Technische Universität Wien
Institut für Information Systems Engineering
Arbeitsbereich: Compilers and Languages
Betreuer: Ao.Univ.Prof. Dipl.-Ing. Dr. Franz Puntigam

INTRODUCTION

Concurrent programming manages multiple concerns simultaneously. This work fills a gap in the literature [1] that emphasizes the connections between the **actor model** and the **microservice paradigm**. Both introduce concurrent computational units into software architectures. We investigate why actors and microservices qualify for programming concurrency and how they facilitate concurrent execution. We evaluate the expressive capabilities of actors and microservices and compare their performance relative to a non-trivial scenario of a concurrent system.

ACTOR MODEL

The actor model defines theoretically well-known constructs which support concurrent, parallel and distributed execution. Actors encapsulate state exclusively and communicate via asynchronous message passing. All actors live inside a runtime system that concurrently executes them. Actors are passive computational units, since they only react to messages. Exclusive state ownership and isolated message processing results in single-threaded semantics internally. Actors are therefore free of synchronization requirements [2].

MICROSERVICE PARADIGM

The microservice paradigm composes complex functionality through relatively small, independent, highly cohesive and loosely coupled executables. Every microservices is a dedicated operating system process. They communicate via lightweight, technology-neutral message passing channels. Microservices are executed by an operating system scheduler, which results in concurrency and parallelism. Network-based communication enables distribution. Microservices are active units and show behavior on their own accord [1].

»Actors and microservices have the same capabilities regarding concurrent programming concerns«

EXPRESSIVENESS & CAPABILITIES

Our evaluation shows that both models have the same expressive capabilities regarding concurrent programming concerns. Both models apply varying implementation strategies with different trade-offs. We focused our capability evaluations on four main concerns:

Encapsulation Actors and microservices conceptually encapsulate state exclusively. Shared-memory environments threaten the isolation of state. Programmers must not expose mutable state through reference sharing among actors. The microservice paradigm forbids shared memory sections. Operating systems enforce strict memory boundaries among system processes.

Communication Actors have an asynchronous messaging primitive. Additional communication abstractions build on top of this primitive, and are therefore at most semi-synchronous. Microservices are open for every technology-neutral, message passing communication channel in every interaction style.

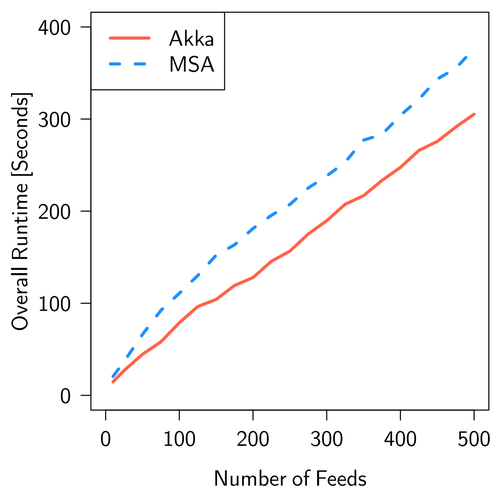
Concurrent Execution Actors and microservices encapsulate state exclusively and communicate solely through message passing semantics. These properties result in a temporal and spacial decoupling, which is the foundation for concurrent execution through an actor runtime or an operating system.

Scalability The ability to scale is a major argument for using actors and microservices. The concurrent and parallel execution capabilities as well as message passing interaction lead to good vertical scalability. The distribution capabilities additionally enable horizontal scalability.

EFFICIENCY & BENCHMARK

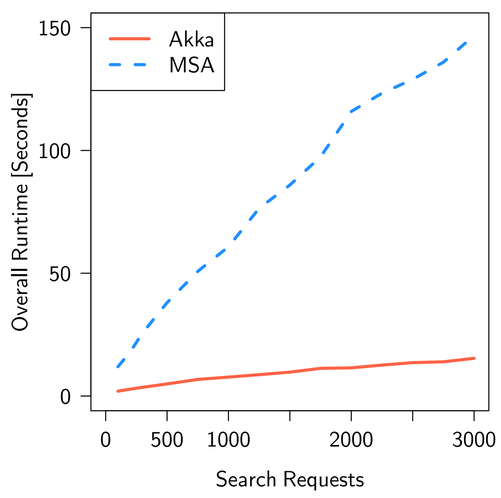
We evaluated the efficiency of the programming models with respect to a non-trivial scenario of a concurrent system. Our scenario is a domain-specific search engine, since the architecture provides many opportunities to facilitate concurrent processing.

The benchmark results below are for an actor-based and a microservices-based system implementation of this scenario. We used Akka as the actor system and built the microservices on the Spring framework.



Experiment 1

The indexing phase facilitates asynchronous communication. The results show that the execution modality of actors is more efficient. Microservices have a higher runtime overhead, since they are separate system processes.



Experiment 2

The time constraints of the retrieval phase favours synchronous interaction. Actors use a request/asynchronous response style. This semi-synchronous strategy shows better efficiency than strictly synchronous communication.

References

[1] Dragoni, Nicola, et al. M̈icroservices: yesterday, today, and tomorrow.P̈resent and Ulterior Software Engineering. Springer, Cham, 2017. 195-216.
[2] Agha, Gul A., and Wooyoung Kim. Ä̈ctors: A unifying model for parallel and distributed computing.J̈ournal of systems architecture 45.15 (1999): 1263-1277.

Kontakt: max@irro.at