Concurrent Programming with Actors and Microservices

Maximilian Irro

Diplomprüfung
12.11.2018

Outline

Forms of Concurrent Execution

- **Pseudo-Simultaneous**: in alternation on a single CPU
- **Parallel**: truely simultaneous on several CPU cores
- **Distributed**: several host machines

Actor Model

- Theoretically well-known constructs
- Receive and process messages (asynchronous, passiv)
- One message at a time
- Encapsulate state exclusively
- Concurrent execution through runtime
- Single-threaded semantics

Microservices Paradigm

- Complex functionality: composition of several *services*
- Microservice: small, independent executable
- „Small“ size $\rightarrow$ in it's *scope of responsibility*
- Dedicated operating system processes
- Communicate via message passing channels

Research Questions

RQ1 Why do actors and microservices qualify for programming concurrency?

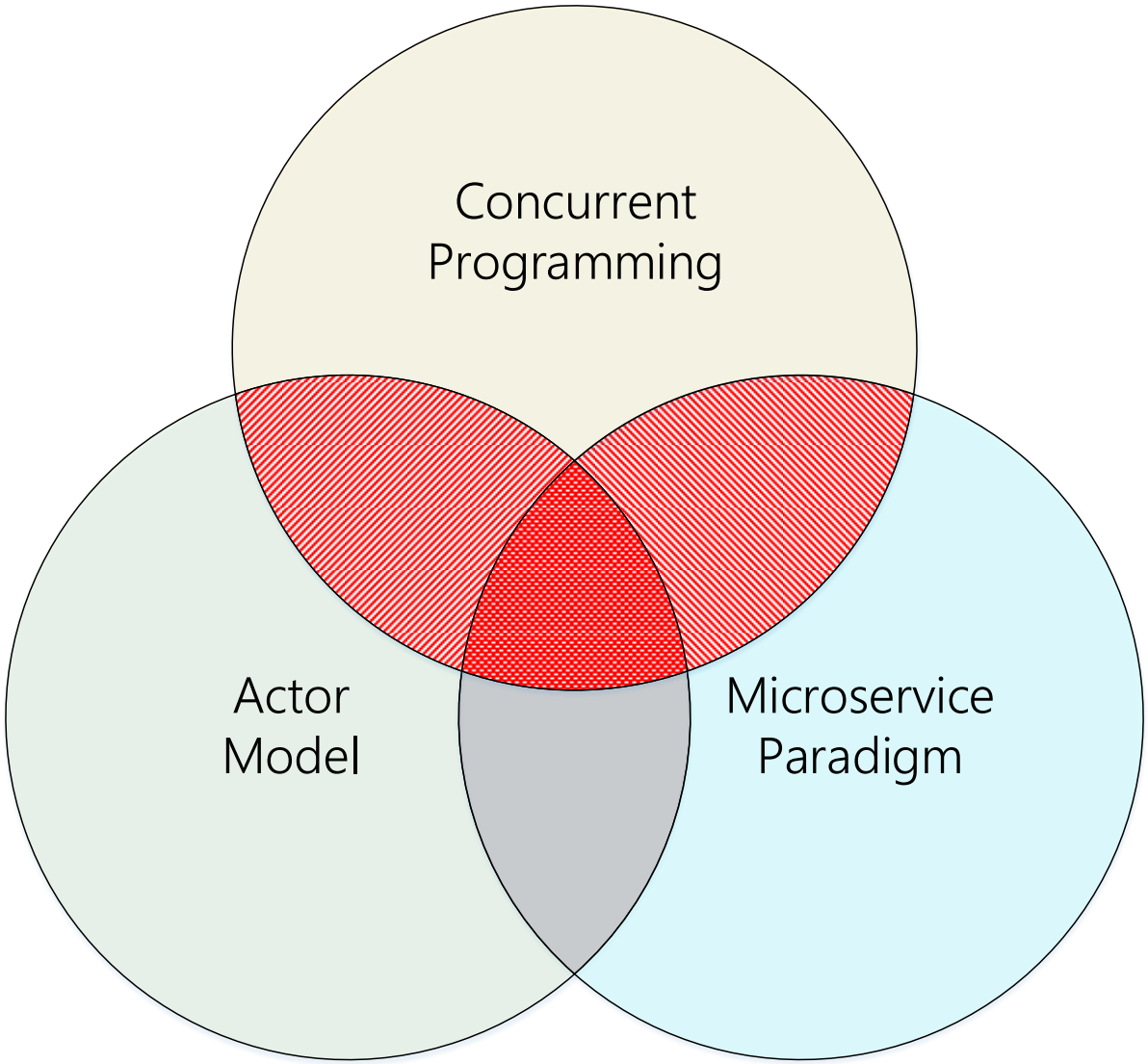
RQ2 How do the actor and the microservice model facilitate concurrent execution?

RQ3 What are the expressive capabilities of actors and microservices regarding concurrent programming concerns?

RQ4 How does the performance of actors and microservices compare in a multi-core environment relative to a concurrent system scenario?

Scope of Research Area

Scope of Research Area



Selected Capabilities

- **Encapsulation**
  Isolation (Shared/ Mutable State), Persistence/IO, Cohesion, Coupling, Independence
- **Communication**
  Communication Styles, Message Routing
- **Concurrent Execution**
  Conception of Conc. Exec., Distribution, Location Transparency, Fairness, Resource Consumption, Notion of Time
- **Scalability and Modularity**
  Forms (Vertical, Horizontal, Load), Dynamic Reconfiguration, Extensibility, Technology Diversity
- **Integration of Actors and Microservices**

Contributions

- Designed a non-trivial concurrent system scenario
  $\rightarrow$ Actor and microservice implementation
- Compared programming of concurrency
- Capability evaluation + efficiency benchmark
- Describe interrelations $\rightarrow$ filled a gap in the literature

Implementation: Domain-specific Search Engine

Implementation: Domain-specific Search Engine

- Gateway (G)
- CatalogStore (C)
- Updater (U)
- Web Crawler (W)
- Parser (P)
- IndexStore (I)
- Searcher (S)

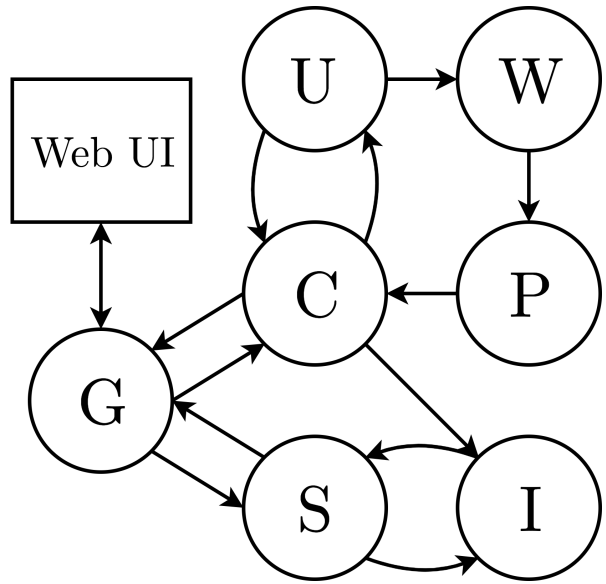


Figure 1: Interaction Model

Implementation: Subsystems & Processing Pipelines

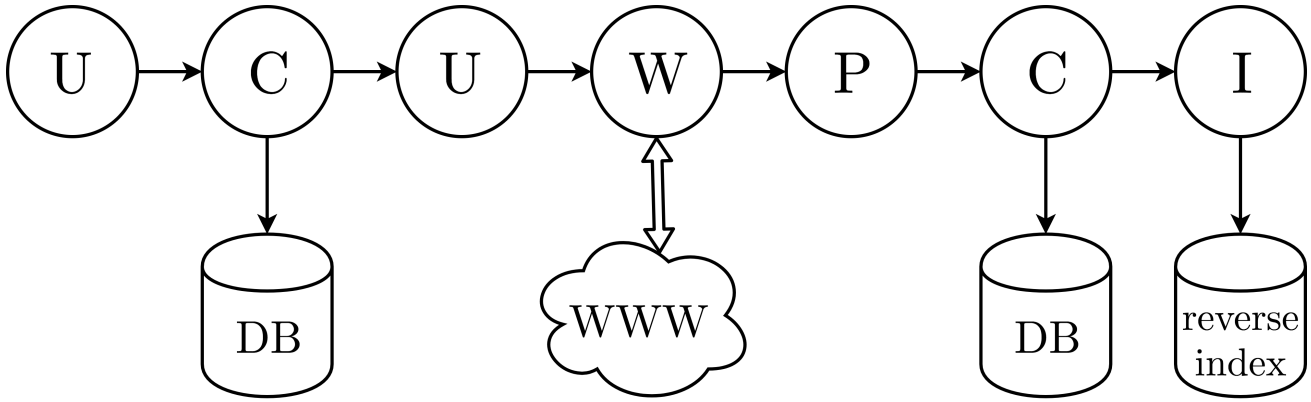


Figure 2: Indexing Pipeline

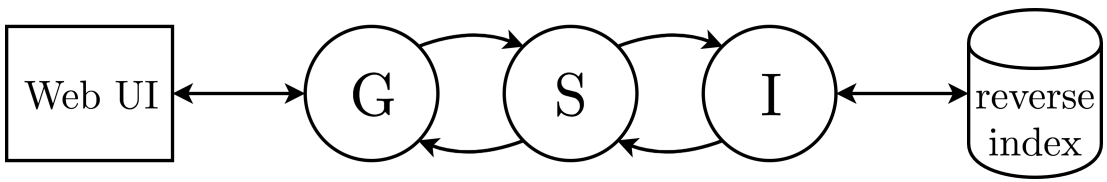


Figure 3: Retrieval Pipeline

Implementation: Technology Stacks

- JVM based
- Actor variant: Akka
- Microservice: Spring Boot + Spring Cloud

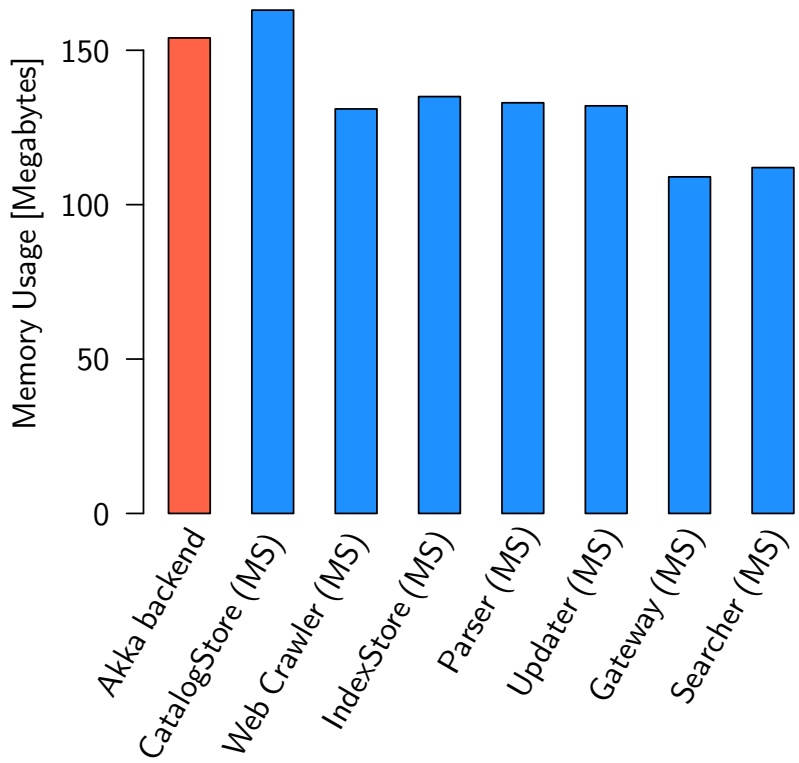
Software Artifact Analysis

Artifact	**LoC**	**sJAR (KB)**	**fJAR (KB)**	**Up (s)**
Akka monolith	4487	1004.3	76 775.1	5.5
CatalogStore (MS)	1838	56.1	89 225.8	14.6
IndexStore (MS)	724	23.8	83 518.2	8.8
Searcher (MS)	656	22.2	81 754.4	8.1
Web Crawler (MS)	716	23.5	83 517.9	9.2
Parser (MS)	703	24.2	83 519.1	8.6
Registry (MS)	334	9.9	90 699.7	9.4
Gateway (MS)	889	30.5	83 655.1	9.7
Updater (MS)	693	23.9	83 518.3	8.7

- $\sum \mathrm{LoC(MS)} = 6553$, about 46 % larger

Artifact Memory Consumption

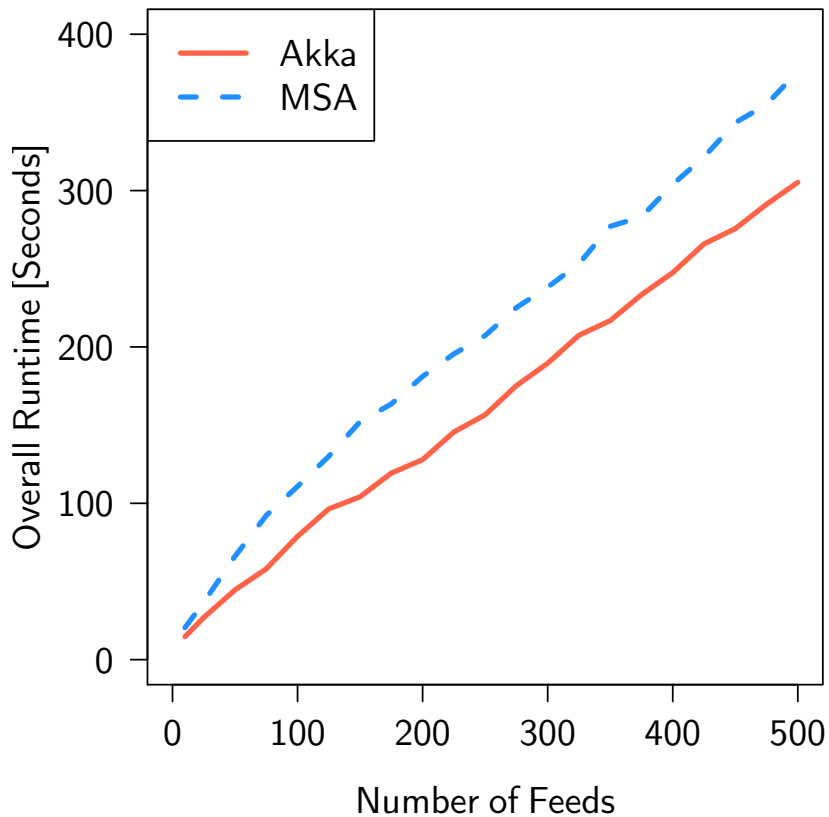
Memory consumption of the executable artifact VMs in the indexing phase:



- $\text{mem}(\text{Akka}) = 154\text{mb}$
- $\sum \text{mem}(\text{MS}) = 915\text{mb}$

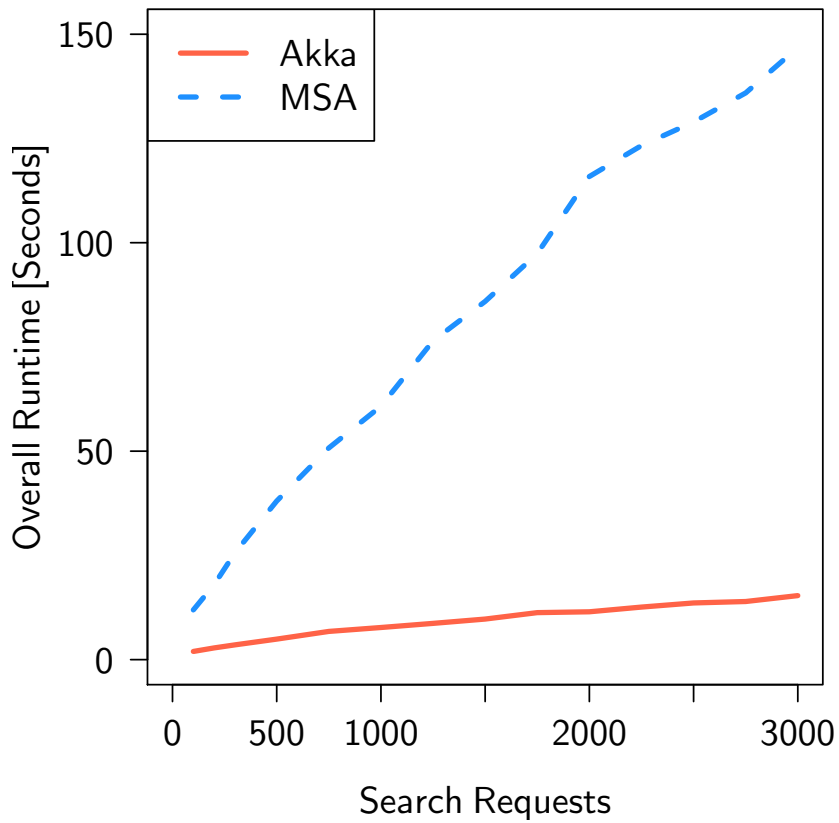
Overall Processing Time: Indexing Subsystem

Benchmark results for the overall processing time of the indexing subsystem:



Overall Processing Time: Retrieval Subsystem

Benchmark results of the overall processing time for the retrieval subsystem:



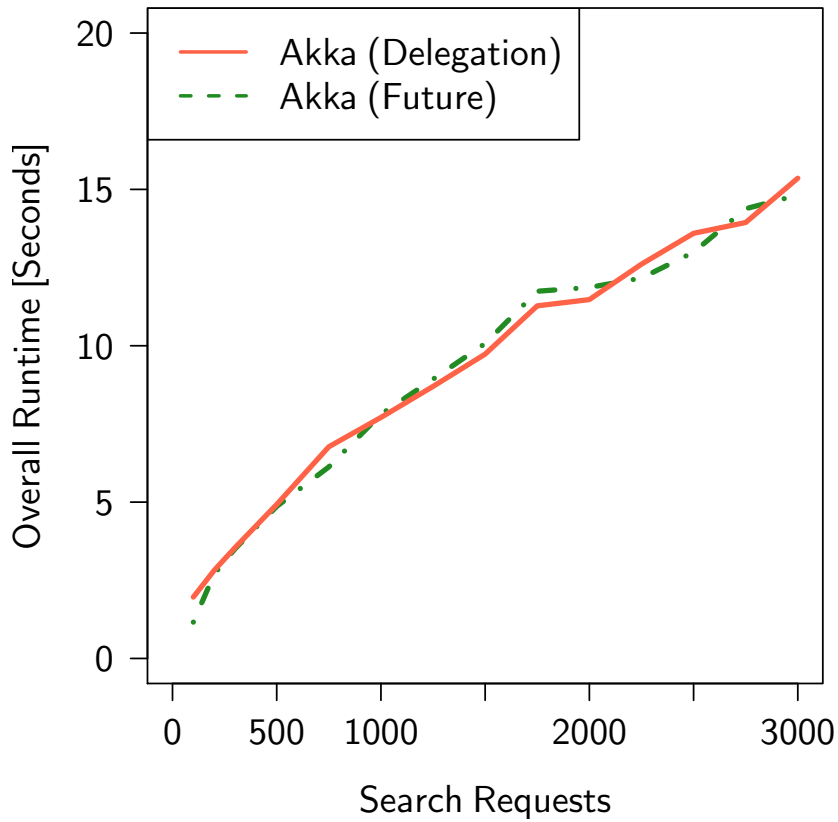
Conclusion & Implications

- **Same** capabilities regarding concurrent programming concerns
- **Different** strategies + trade-offs
- Actors: more efficient, less resource consumption
- Microservices: more independent, more interaction freedom
  $\rightarrow$ benefit from actor principles

</presentation>

Supplemental: Semi-Synchronous Communication in Akka

Comparison of the benchmark results for the retrieval subsystem using either delegation or futures for request/response communication in the Akka:



Supplemental: State Encapsulation vs. Isolation

- Microservice: process memory boundaries
- Actors (on the JVM):
  - Visibility + Accessibility $\rightarrow$ information hiding
  - Reference types + pass-by-value $\rightarrow$ immutability
  - Coding conventions required