Declarative distributed concurrency in Scala

Sergei Winitzki

Scale by the Bay 2018

November 16, 2018

Talk summary

How I learned to forget semaphores and to love concurrency

`Chymyst` = an implementation of the Chemical Machine (CM) paradigm

- CM $\approx$ Actors made purely functional and auto-parallelized
- Intuitions about why CM works better than other concurrency models
  - Comparison with related work: ING Baker, BPMN (workflow)
- New extension for distributed programming: DCM
- Code examples and demos

Not in this talk: academic theory

- Petri nets, π-calculus, join calculus, joinads, mobile agent calculus...
- DCM formulated within some theory of distributed programming

Concurrent & parallel programming: How we cope

Imperative concurrency & parallelism is difficult to reason about:

- low-level API: callbacks, threads, semaphores, mutex locks
- hard to reason about mutable state and running processes
- hard to test – non-deterministic runtime behavior!
  - race conditions, deadlocks, livelocks

Known declarative approaches to avoid these problems:

Kind of concurrency	**Formal structure**	**Scala implementation**
synchronous parallelism	applicative functor	Spark, `.par.map()`
asynchronous streaming DAG	monadic functor	`Future`, `async`/`await`, `RxJava`, Akka Streams
unrestricted streaming	recursive monad+	Flink, fs2, ZIO
unrestricted concurrency	?	Akka, `Chymyst`

For distributed computing: challenges remain

- coordination and consensus, persistence and fault tolerance
- cluster configuration and discovery
  - distributed coordination as a service: Apache ZooKeeper, `etcd`