Unlocking Performance
Reactive Systems Design Principles

Balaji Srinivasaraghavan
What are Reactive Systems?
Reactive Systems

- Responsive ⇔ usability, simplified error handling
- Resilient ⇔ isolated components with external monitoring, supervisors
- Scalable ⇔ detect input rate and allocate resources, backpressure
- Event-driven ⇔ asynchronous message-passing, non-blocking
Reactive Programming

• Stream based APIs
• Framework provides failure recovery
• Message driven
  – Non blocking (no direct method calls)
  – Time wise isolation – concurrency
  – Space wise isolation – distributability
• Backpressure enabled
Stream Example

messageQueueSource

  .map(message -> parse(message))

  .mapAsync(5, tweet -> detectBot(tweet.get())) //concurrency hint

  .runForeach(this::writeFraudReport, materializer);

* from https://github.com/ktoso/akka-streams-alpakka-talk-demos-2016
Choosing a Toolkit (JVM)
Akka vs RxJava

- Akka
  - Platform play; typically a rewrite of existing apps
  - Vendor lock-in
  - Performant
  - Lower build time due to comprehensive tooling - HTTP, DB

- RxJava 2
  - Incremental adoption; easier to add to existing code
  -Verbose non-intuitive API does not encourage clean code
  - Not as performant as Akka’s actor model based implementation

- Spring Reactor
  - Promising new kid on the block
Granular Parallelism

- Granular parallelism is at the heart of reactive system design
- Enables
  - Better resource utilization
  - Error bulk heading

Amdahl's law - parallel computing with many processors is useful only for highly parallelizable programs
Stream Decomposition

• Direct Akka Streams
  – High performance, low latency
  – Simple business logic

• Actor hosted streams
  – Structuring complex business logic
  – Check pointing – efficient recovery & replay on failures
  – Need to maintain state – statistics, threshold counters
Model Concurrency with Stream Primitives

- Design and implement concurrency using reactive toolkit primitives only

**Jax-RS Async (Avoid)**

```java
    .async().get();
```

**RxJava 2 (Adopt)**

```java
source.parallel(2)
    .map(this::parallelWorkToRun)
    .sequential()
```
Reactive Development
Build a Pattern Library

• Building a library of patterns institutionalizes thinking and vocabulary
  – Select between two flows based on a predicate
  – Select one of many flows based on a decider function
  – Bypass running a flow on errors
  – Retry an operation based on a decider
  – Akka provides some patterns as well – akka-stream-contrib

Flow<M, N, NotUsed> selectByIndex(
  Function<M, Integer> numericPartitioner,
  Flow<M, N, NotUsed>... innerFlows
)

©2017 PayPal Inc. Confidential and proprietary.
Use Rich Message Types

• Stream element types are an important design tool
• Provide for orthogonal features like error handling, retry
• Propagate upstream state including failures within the stream envelope

```java
public abstract class BaseExceptionAwareMessage {
    final public Optional<Exception> exception;
    public boolean hasException() { exception.isPresent(); }
}

httpSource
    .map(message -> parse (message))
    .via(passThroughErrors(this::additionalWork))
    .runForeach(this::logMessageCompletion, materializer);
```
Use Immutables for State

- Final fields are special - automatically visible to all threads
- Immutable state ≠ unchangeable
- Every change results in a new object instance
- Akka assumes this model for reactive streams
- JMM (JSR-133) guarantee enables correctness & performance

```java
class Foo {
    public final Person customer;
    public Foo(Person customer) {
        this.customer = customer;
    }
}
```

©2017 PayPal Inc. Confidential and proprietary.
Q & A
Thank You