Introduction to Akka

Lenko Donchev

August 7, 2013

Introduction

- Akka overview

Examples

- Example Akka actors

About

What is Akka?

- Event-driven middleware framework for building high performance and reliable distributed applications in Scala and Java.
- Open Source and available under the Apache 2 License.

Why Akka?

- Scalability
- Concurrency
- Fault-Tolerance

Simple Concurrency Distribution

- Asynchronous and Distributed by design.
- High-level abstractions like Actors, Futures and STM

Resilient by Design

- Write systems that self-heal.
- Remote and/or local supervisor hierarchies.

High Performance

- 50 million msg/sec on a single machine.
- Small memory footprint; 2.5 million actors per GB of heap..

Elastic Decentralized

- Adaptive load balancing, routing, partitioning and configuration-driven remoting.

Extensible

- Use Akka Extensions to adapt Akka to fit your needs.

STM - Software Transactional Memory

- Turns the JVM heap into a transactional data set
- Provides begin/commit/rollback semantics
- Implements the first three letters of ACID atomicity - all or none consistency - data is left in a consistent state isolation - only participants see changes during transaction
- Not Durable because STM is in-memory
- Modeled After Clojure's STM
- Transactions are automatically retried on collisions

Concurrency with Actors

- Provides a high-level abstraction for concurrent and distributed system development
- Asynchronous message processing using event-driven receive loop
- Removes the burden of explicit thread and lock management to make concurrent programming easier
- Pattern matching against messages is a convenient way to express an actor's behavior.
- Very lightweight
- Helps you to focus on the message workflow instead of low level primitives like threads, locks and socket IO

The origin of Actors

- Defined in a 1973 paper by Carl Hewitt
- Popularized by Erlang
- originally developed at Ericsson
- designed for distributed, fault-tolerant, non-stop systems
- 9-nine's reliability or down-time of 31 ms/year
- direct support for actor concurrency model in the language
- supports hot-swapping of code

An Actor...

- Encapsulates state and behavior into a lightweight ”process”
- Supports Hot Swapping of the Actors message loop (e.g. its implementation) at runtime.
- Shares nothing with other actors
- Communicates with other actors through messages
- Communicates asynchronously
- Has a message queue or ”mailbox”
- Has support for durable mailboxes
- Non-blocking

Advantages of the Actor Model

- Is easier to reason about
- Raises the level of abstraction
- Makes it easier to avoid:
  - race conditions
  - deadlocks
  - starvation
  - live locks

Remote Actors

- Remote Actors provide a way to scale "out"
- Actors are excellent for distributed computing
- Remote Actors are implemented with NIO on top of
  - JBoss Netty; an NIO client server framework
  - Google Protcol Buffers; structured data encoding format

Fault Tolerance

- The "let it crash" approach
- Designed for concurrent and distributed systems
- Notification of failures
- Supervision and repair of failed nodes

Accept Failure as a fact of life and manage it

Built in Fault-tolerant design from the ground up

- Supervisor Hierarchies have a different view of failure
- Components are monitored by a "linked" supervisor When the supervisor detects failure, nodes are:
  - reset to a stable state
  - restarted

Restart Strategies

Akka supports two restart strategies

- OneForOne - restarts the component that crashed
- AllForOne - restarts all managed components if one crashed

OneForOne Strategy

- Failure of one actor forces restart of only that actor
  - Actor Fails, throwing exception
  - Exception thrown by actor is propagated to Supervisor
  - Supervising Actor restarts failed actor
- Restart initializes actor to a well-known state

AllForOne Strategy

- Failure of one actor forces restart of all supervised actors
    - Actor Fails, throwing exception
    - Exception thrown by actor is propagated to Supervisor
    - Supervising Actor restarts all supervised actors
- Restart initializes actors to a well-known state

Restart Callbacks

- Actors have 2 different restart callbacks
  - pre restart
  - post restart
- Used to clean up and reinitialize state upon restart

Add-On Modules

- Persistence
- STM integration with NoSQL DBs for the 'D' in ACID Cassandra, MongoDB, Redis, CouchDB, Amazon SimpleDB, and others
- AMQP - based on the RabbitMQ client
- JTA - allows STM to participate in a JTA transaction
- Spring Integration
- Guice Integration

Example Actor in Scala

```
class HelloWorldActor extends Actor {
  def receive = {
    case ”hello” => println(”Hello World!”)
    case _ => println(”hi”)
  }
}
```

Example Actor in Java

```
public class HelloWorldActor extends UntypedAct
  public void onReceive(Object message) throws
    if (message instanceof String)
     System.out.println(”Hello world...”);
    else
     System.out.println(”hi”);
  }
}
```

Asynchronous Messaging in Java

```
helloWorldActor.sendOneWay(”Hello”);
```

Synchronous Messaging in Java

```
// send and receive
try {
  Object result =
  helloWorldActor.sendRequestReply(”Hello”, get
} catch(ActorTimeoutException ate) {
  // handle timeout
}
```

Replying to Messages

```
self.reply("reply")

// Scala
self.sender.get ! "reply"
```

Asynchronous Messaging in Scala

```
// fire-and-forget syntax (asynchronous)
helloWorldActor ! "Hello"
// returns immediately
```

Synchronous Messaging in Scala

```
// Send-and-Receive syntax (synchronous)
helloWorldActor !! "Hello"
```

Example: Building a simple Distributed Search Engine

https://github.com/lenko-d/distributed_search_engine

About

Lenko Donchev

@Twitter: Lenko_HD