Reasonable RPC with Remotely

Rúnar Bjarnason
Tim Perrett

ScalaDays, 18th March 2015
Existing solutions

No panaceas. Just tradeoffs.
Monolith

- All functionality in a single binary
- Doesn’t scale
- Difficult to evolve and maintain
Broker pattern

- Message bus or broker mediates between services
- Single point of failure/integration
- If broker goes down, the whole system is down
Message-passing

- Lightweight and inexpensive
- Reliable and scales well
- Discovery is a pain
Message passing solutions

- Finagle
- Thrift
- Akka
- HTTP*(JSON+XML)/REST
Finagle

- Imperative API
- No reuse
- Too powerful
- No higher-order constructs
Thrift

- All of the problems with Finagle
- Tough tooling
- Documentation is lacking
- No formal spec for wire format
Akka

- Too powerful
- Untyped
- Actor system API bleeds to the outside
- Behavior of complex systems is unreasonable
HTTP*(JSON+XML)/REST

- Unproductive, developer time wasted
  - Marshalling/unmarshalling
  - Defining HTTP endpoints and using HTTP clients
- Resources wasted on HTTP layer and parsing
- Does what we want, but at tremendous cost
Our Vision
Productivity, Safety, Reuse
Productivity

Don’t want to think about:

- HTTP protocol specifics
- Every possible failure mode
- Serialization format
- Compatibility
Safety

- Ad-hoc evolution doesn’t scale
- Incompatibilities should be compile-time failures
- Fail as early as possible
Reuse

- Use an actual programming language
- Build protocols from modular primitives
- Compose large protocols from small ones
Remotely
RPC for reasonable people
Remotely
RPC for reasonable people
BETA
Remotely

- Pilot for Verizon open-source
- Taking the time to build exactly what we want to use
- You can use it too!
Remote call

```scala
val x = factorial(9)
```
Remote reference

factorial: Remote[Int => Int]
Applying a remote function

val x: Remote[Int] = factorial(9)
Getting a response

```scala
val x: Response[Int] = factorial(9).run(endpoint)
```
Response monad

type Demo[A] = Kleisli[Response, Endpoint, A]

implicit def remoteToK[A](r: Remote[A]): Demo[A] = Kleisli.ask[Response, Endpoint].flatMapK(r.run)

def call: Demo[(Movie,List[Actor])] = for {
  a <- MovieClient.getMovie("m2")
  b <- MovieClient.getActors(a)
} yield (a,b)
val address = new InetSocketAddress("10.1.0.1",8080)

val exe: Task[(Movie,List[Actor])] = for {
    transport <- NettyTransport.single(address)
    endpoint = Endpoint.single(transport)
    output <- call(endpoint)(Context.empty)
    _ <- transport.shutdown
} yield output

val (movie, actors) = exe.run
Demo
Feature Pageant
Binary Codecs

- Fast, lightweight binary serialization
- Built on `scodec`
- Ships with codecs for standard Scala types
- Easy to make codecs for your own types

Future goal:
  Fully automate codec-creation
Pluggable transports

- Uses Netty 4 out of the box
- Akka I/O
- Netty 3
- ØMQ
Pluggable transports

type Handler = 
  Process[Task, BitVector] => 
  Process[Task, BitVector]
Endpoints

case class Endpoint(
  connections: Process[Task, Handler])
Endpoint combinators

```python
def circuitBroken:
    (Duration, Int, Endpoint) => Endpoint

def failoverChain:
    (Duration, Process[Task, Endpoint]) => Endpoint
```
Context-passing

- Context provides arbitrary container for experiment propagation.
- Every request tagged with a UUID.
- Contains stack of request UUIDs.
- highlights the “path” through the system request graph when debugging.
Pluggable monitoring

- Report round-trip latency timing.
- Trace every request uniquely.
- Trace the request stack entirely.
- Bring your own by implementing the `Monitoring` trait.
- Uses `println` out of the box.
Capability checking

- Protocol versioning over time inevitable
- Determine client/server compatibility before establishing connection
How fast is it?

- Went from 56 ms to 32 ms (production payload)
- In benchmarks on modest hardware
  - Min: \(\sim 400 \, \mu s\)
  - Mean: \(\sim 1.5 \, ms\)
- Throughput is an issue currently
Future work

- Prettier API
- Drastically improve throughput
- Extensible protocol language
- More built-ins for e.g. server-side map and flatMap
- Remote polymorphic functions
Summary

- Remotely is productive
- It’s safe
- It promotes code reuse
- You can contribute!
EOF

@runarorama
@timperrett

github.com/oncue/remotely

github.com/oncue/remotely-demo