Supporting True Asynchrony in JAX-WS RI

Kohsuke Kawaguchi
Jitendra Kotamraju

Sun microsystems

Background : Input from JBI team

- “We need 1000 concurrent server-to-server communication where each request processing takes 5 minutes"
- A client connect to a service via HTTP, send a request, server comes back 5 mins later with response. All in one HTTP connection.
- WS-Addressing. Request and response may use completely different transports and timing
- WS-ReliableMessaging. Abort, resend, etc.

Sun Confidential: Internal Only

Problems

- We can't afford to tie a thread with one request/response processing
- Current pipe architecture simply can't cope with this
- ... So we have to fix `Pipe`
- I'm very sorry

Sun Confidential: Internal Only

Proposal Overview

- Do things in continuation passing style
- Why CPS?
- Introducing **`Valve`** to replace **`Pipe`**
- How it works?
- Migration path

Sun Confidential: Internal Only

Continuation Passing Style

- During packet processing, we need to remember "the remaining work to be done" (=continuation)
  - Normally, Java uses call stack to capture this
  - CPS maintains this outside call stack
- Continuation conceptually takes a “parameter"
  - In Java, that "parameter" is really the return value
  - In JAX-WS, that "parameter" is always Packet
- Think of this as a virtual user-level thread
- We call it `Fiber`
- See Wikipedia

Sun Confidential: Internal Only

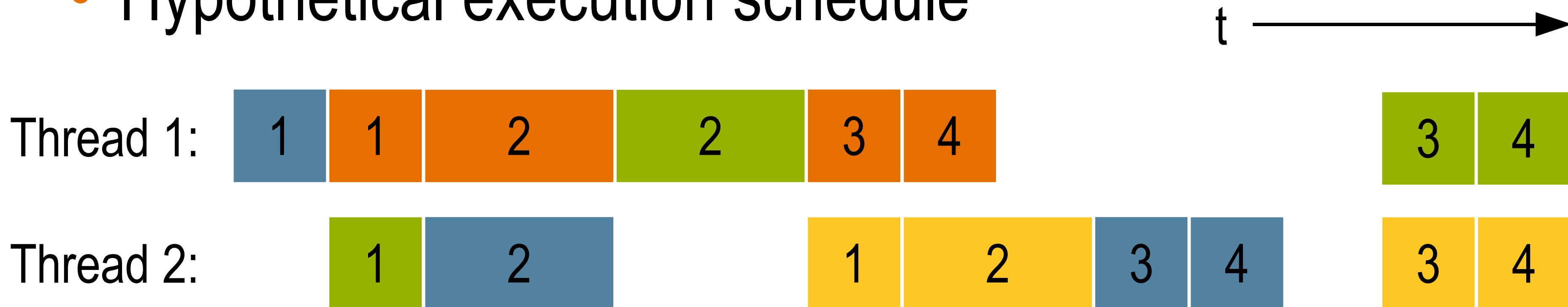
CPS: Example

- Consider the following scenario
  - App sends message A
  - Y puts A on hold and sends protocol message B
    - Continuation is "when response comes, run Z, Y, then sends A"
  - Y receives response C to B
  - Message A is sent
    - Continuation is "when response comes, run Z, Y, and X"
  - Server gives us response D

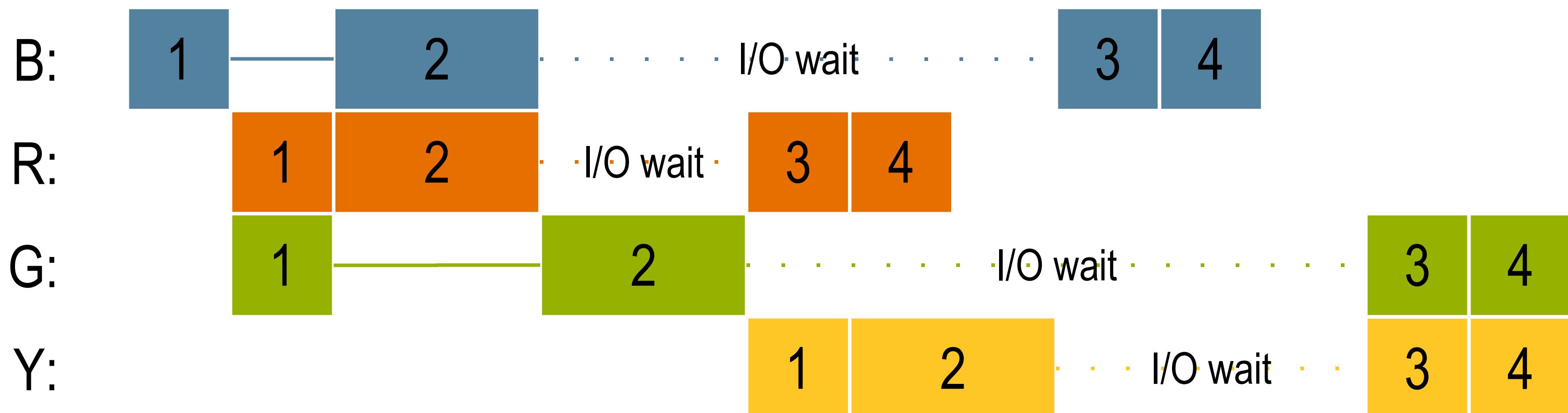
Sun Confidential: Internal Only

CPS: Example

- Hypothetical execution schedule



- How is each invocation served?



Sun Confidential: Internal Only

Why?

- One invocation might be served by multiple threads
  - > But we'll have synchronous execution support, too
- High thread utilization. No more blocking I/O
  - > On client-side, needs a suitable transport layer
- Large network latency is no longer a problem
- Slow network is **still** a problem
- Simple “send” and “response” style method chaining (like pre-rearch) won't work in the face of protocol messages

Sun Confidential: Internal Only

Introducing Fiber

- Think of this as a thread
  - > Except it takes "what to execute" in start, not ctor
  - > Keeps track of the continuation "do ... with packet P"

```
public class Fiber {
  void start(Valve,Packet);
  void resume(Packet);
  static Fiber current();
  static Fiber create();
}
```


Sun Confidential: Internal Only

Introducing Fiber

- Sometimes synchronous execution is needed
  - > For doPrivileged()
  - > For serving synchronous client invocation

```
public class Fiber {
    void start(Valve,Packet);
    void runSync(Valve,Packet);
    void resume(Packet);
    void suspend();
    static Fiber current();
    static Fiber create();
}
```


Sun Confidential: Internal Only

Introducing Valve

- Act on **`Packet`**, then decide what to do next

```
public interface Valve {
  NextAction processRequest(Packet p);
  NextAction processResponse(Packet p);
}
```


Sun Confidential: Internal Only

Introducing Valve

- Trivial filter example

```
class MyValve implements Valve {
   Valve next;
   NextAction processRequest(Packet p) {
      return NextAction.invoke(next,p);
   }
   NextAction processResponse(Packet p) {
      return NextAction.returnWith(p);
   }
}
```


Sun Confidential: Internal Only

Introducing Valve

- But creating NextAction every time could be expensive, so please do this instead

```
class MyValve extends AbstractValve {
  Valve next;
  NextAction processRequest(Packet p) {
    return doInvoke(next,p);
  }
  NextAction processResponse(Packet p) {
    return doReturnWith(p);
  }
}
```


Sun Confidential: Internal Only

Valve vs Pipe

- Pipe was proactive; each pipe calls the next one
- Valve is reactive to simulate CPS
  - You no longer invoke "next.process()"
  - Instead, tell the calling runtime system (JAX-WS) what to do, via NextAction
- Valve can be run as a pipe
- Pipe can be run as a (sloppy) valve

Sun Confidential: Internal Only

Converting Pipe to Valve

- This is your code today

```
class MyPipe implements Pipe {
  Pipe next;
  Packet process(Packet p) {
    op = p.getOperation();
    r=next.process(p);
    System.out.println(op);
    return r;
  }
}
```


Sun Confidential: Internal Only

Converting Pipe to Valve

- s/Pipe/Valve/ and extend from AbstractValve

```
class MyValve extends AbstractValve {
 Valve next;
 Packet process(Packet p) {
  op = p.getOperation();
  r=next.process(p);
  System.out.println(op);
  return r;
 }
}
```


Sun Confidential: Internal Only

Converting Pipe to Valve

- Split request and response

```
class MyValve extends AbstractValve {
 Valve next;
 Operation op;
 void processRequest(Packet p) {
  op = p.getOperation();
  r=next.process(p);
 }
 void processResponse(Packet r) {
  System.out.println(op);
 }
}
```


Sun Confidential: Internal Only

Converting Pipe to Valve

- Instead of next.process(), decide what to do next

```
class MyValve extends AbstractValve {
  Valve next;
  Operation op;
  NextAction processRequest(Packet p) {
    op = p.getOperation();
    return doInvoke(next,p);
  }
  NextAction processResponse(Packet p) {
    System.out.println(op);
    return doReturn(p);
  }
}
```


Sun Confidential: Internal Only

Possible Next Action

- Invoke(Valve n,Packet p)
  - > C'(x) := C(v.processResponse(x))
  - > Do n.processRequest(p) with C'
- Invoke and forget(Valve n,Packet p)
  - > Do n.processRequest(p) with C
- Return(Packet p)
  - > Do C(p)
- Suspend
  - > Wait for the fiber to be resumed with packet p
  - > Do C(p)

C: continuation

Sun Confidential: Internal Only

Migration Step 1: JAX-WS goes first

- JAX-WS will convert its pipes to valves
  - > Resulting valves can be still run as pipes
- JAX-WS will convert stubs to drive valves
- JAX-WS will implement server-side async transport
  - > We'd like TCP transport to be ported to async
- Pipeline starts running async, until it hits first pipe
- Then everything then on will be run as pipe

Sun Confidential: Internal Only

Migration Step 2: Tango goes next

- Individual component decides when to convert
  - For most filter pipes, we expect this to be easy
- A few components will be seriously impacted
  - RM and Addressing
  - Transports

Sun Confidential: Internal Only

Migration Step 3: Switch

- We need client-side async HTTP transport
  - > If we want to be scalable on server-to-server
- All the pipes are converted to valves
  - > We'll be automatically running completely in async
- Delete old pipe related code

Sun Confidential: Internal Only

More Resources

- Sandbox workspace used for prototyping
  - > http://kohsuke.sfbay/hudson/job/jaxws-cps-sandbox/
- Discussion
  - > dev@jax-ws.dev.java.net

Sun Confidential: Internal Only

Supporting True Asynchrony in JAX-WS RI

Kohsuke Kawaguchi
Jitendra Kotamraju

Sun microsystems