Distributed Systems Made Simple

Pascal Felber, Lorenzo Leonini, Etienne Rivière, Valerio Schiavoni, José Valerio

Lua Workshop, 8 September 2011
Frick, Switzerland
About Me

• 2010-now : PhD student at the University of Neuchatel, Switzerland (and Lua user since then)

• Topic: large-scale distributed systems, cloud computing. Daily job:

  1. Invent new protocols for cloud applications

  2. Do experiments, write papers, go to 1

• 2007-2009: Research engineer at INRIA, France

• BSc and MSc in Computer Engineering at the Università degli Studi Roma Tre, Italy
Motivations
Motivations

• Developing, testing and tuning distributed applications is **hard**

• In Computer Science research, fixing the gap of simplicity between pseudocode description and implementation is **hard**

• Using worldwide testbeds is **hard**
What is PLANETLAB
What is PLANETLAB
What is PLANETLAB

• Machines contributed by universities, companies, etc.
  • 1098 nodes at 531 sites (02/09/2011)
  • Shared resources, no privileged access

• University-quality Internet links

• High resource contention

• Faults, churn, packet-loss is the norm
  • Challenging conditions
Daily Job With Distributed Systems
Daily Job With Distributed Systems

- Write (testbed specific) code
- Local tests, in-house cluster, PlanetLab...
Daily Job With Distributed Systems

- Debug (in this context, a nightmare)
Daily Job With Distributed Systems

- Debug (in this context, a nightmare)
Daily Job With Distributed Systems

- Deploy, with testbed specific scripts
Daily Job With Distributed Systems

- **code** → **debug** → **deploy**

3. Deploy, with testbed specific scripts
Daily Job With Distributed Systems

1. code
2. debug
3. deploy
4. get logs

- Get logs, with testbed specific scripts
Daily Job With Distributed Systems

- Get logs, with testbed specific scripts
Daily Job With Distributed Systems

1. code
2. debug
3. deploy
4. get logs
5. plots

• Produce plots, hopefully
Daily Job With Distributed Systems

code ➔ debug ➔ deploy ➔ get logs ➔ plots

5. Produce plots, hopefully
**SPLAy** At Glance

- Supports the **development**, **evaluation**, **testing**, and **tuning** of distributed applications on any testbed:
  - In-house cluster, shared testbeds, emulated environments
  - Provides an **easy-to-use** pseudocode-like language implemented in Lua
SPLAy  At Glance

- Supports the development, evaluation, testing, and tuning of distributed applications on any testbed:
  - In-house cluster, shared testbeds, emulated environments
- Provides an easy-to-use pseudocode-like language implemented in Lua
At Glance

• Supports the development, evaluation, testing, and tuning of distributed applications on any testbed:
  • In-house cluster, shared testbeds, emulated environments
  • Provides an easy-to-use pseudocode-like language implemented in Lua

SPLA4

gnuplot is your friend
The Big Picture

Splay
Controller
SQL DB

require"splay.base"
The Big Picture

require"splay.base"
The Big Picture
Why Lua?
Why?

- Light & Fast
- (Very) Close to equivalent code in C
Why Lua?

- Light & Fast
- (Very) Close to equivalent code in C
- Concise
- Allow developers to focus on ideas more than implementation details
- Key for researchers
Why LUA?

• Light & Fast
• (Very) Close to equivalent code in C
• Concise
• Allow developers to focus on ideas more than implementation details
• Key for researchers
• **Sandbox** thanks to the possibility of easily redefine (even built-in) functions
Concise
Concise Pseudo code as published on original paper

Executable code using SPLAY libraries
Sandbox: Motivations

• Experiments should access only their own resources

• Required for non-dedicated testbeds
  • In universities, companies
  • Totally available at night/holiday time

• Memory-allocation, filesystem, network resources are restricted
Sandboxing LuaSocket

- Same API as plain LuaSocket
- On-demand sandboxed sockets
  - Very easy thanks to Lua’s metatable
- Limits chosen by the SPLAYd deployer
- Both TCP and UDP
- Example: UDP’s socket.send

```lua
if sock.send then
  new_socket.send = function(self, data)
    local len = #data
    if total_sent + len > max_send then
      io:warn("Send restricted (total: ", total_sent, ")")
      return nil, "restricted"
    end
    local n, status
    if math.random(1000) > udp_drop_ratio then
      n, status = sock:send(data)
    else
      n = len
    end
    if n then
      total_sent = total_sent + len
    end
    return n, status
  end
end
```
Sandboxing LuaSocket

```lua
if sock.send then
    new_sock.send = function(self, data)
        local len = #data
        if total_sent + len > max_send then
            l_o:warn("Send restricted (total: "..total_sent..")")
            return nil, "restricted"
        end
        local n, status
        if math.random(1000) > udp_drop_ratio then
            n, status = sock:send(data)
        else
            n = len
        end
        if n then
            total_sent = total_sent + len
        end
        return n, status
    end
end
```
for _,module in pairs(splay)
for _,module in pairs(splay)
for _,module in pairs(splay)

Modules sandboxed to prevent access to sensible resources
splay.events

Modules sandboxed to prevent access to sensible resources
splay.events
splay.events

- Distributed protocols can use message-passing paradigm to communicate
- Nodes react to events
- Local, incoming, outgoing messages
- The core of the Splay runtime
- \texttt{splay.socket, splay.io} designed to provide a non-blocking operational mode
- Based on Lua’s coroutines
splay.rpc

luasocket  events  io

crypto    llenc/benc  json

misc      log      rpc
splay.rpc
splay.rpc

- Default mechanism to communicate between nodes
- Support for UDP/TCP
- Efficient BitTorrent-like encoding
- Experimental binary encoding
Life Before **SPLAY**

- Time spent on developing testbed specific protocols
- Or fallback on simulations...
- The focus should be on ideas
- Researchers usually have no time to produce industrial-quality code
public void declareNeighbourInactive(final DHTNode node) {
    // DO NOTHING HERE:
}

public void addToLeafSet(final DHTNode node) {
    // DO NOTHING
}

/**
 * Empty UDP message is 64 bytes, to which we must add 1 bytes for the header,
 * 1 bytes for the messegID, 28 bytes (IP, port, Bamboo ID) for message src, 28
 * bytes for message dest + bytes for the specific data carried in the
 * message. NOTE: the size of informations to ack a message is 10 bytes.
 */

@Override
public long calculateMessageBytes(final Message msg) {
    long result = 122; // 64 + 1 + 1 + 28 + 28;

    if (msg.header == MessageType.PING) {
        /* nothing to add */
        return result;
    }

    if (msg.header == MessageType.PONG) {
        /* some bytes for the informations about which message is being acked */
        result += ACK_INFORMATION_SIZE;
        return result;
    }

    if (msg.header == MessageType.BAMBOO_JOIN_LOOKUP_REQUEST) {
        /* msg.src */
        pong.ackedMsg = msg;
        simulator.send(pong);
    }

    Message pong = new Message(MessageType.PONG, msg.messageId, this,
            /* msg.src */);

    if (msg.header == MessageType.PONG) {
        /""
        * try to send the given message for a total number of times: if an ACK or a
        * response arrived out of time).
    }

    @Override
    protected void dispatchResponseToAppropriateHandler(final Message msg) {
}

    public final THashMap<Long, ResponseArrivedCallback>    sentLookupRequestsMessagesCallbacks = new THashMap<Long, ResponseArrivedCallback>();

    /**
     * When a response containing a given acked message arrives, the corresponding
     * handler is notified.
     * Singleton callback which does nothing on response received.
     */

    public static final THashMap<Long, ResponseArrivedCallback>    emptyResponseArrivedCallbackMap = new THashMap<Long, ResponseArrivedCallback>();

    @Override
    protected long nextMessageId() {
        return ++nextMessageId;
    }

    public final static String bigIntegerToBinaryString160(final BigInteger k) {
}

    public final static BigInteger distanceBetween(final BigInteger id1, final BigInteger id2) {
        final BigInteger second = distance(o2);
        final BigInteger diffId2Id1 = id2.subtract(id1).mod(MOD);
        return diffId2Id1;
    }

    public final static long distance(final Message m) {
        final long messageId = m.messageId;
        final LookupInfo lookupInfo = (LookupInfo) m.content;
        long result = 122; // 64 + 1 + 1 + 28 + 28;

        if (msg.header == MessageType.PING) {
            /* nothing to add */
            return result;
        }

        if (msg.header == MessageType.PONG) {
            /* some bytes for the informations about which message is being acked */
            result += ACK_INFORMATION_SIZE;
            return result;
        }

        if (msg.header == MessageType.BAMBOO_JOIN_LOOKUP_REQUEST) {
            /* msg.src */
            pong.ackedMsg = msg;
            simulator.send(pong);
        }

        Message pong = new Message(MessageType.PONG, msg.messageId, this,
                /* msg.src */);

        if (msg.header == MessageType.PONG) {
            /* try to send the given message for a total number of times: if an ACK or a
            * response arrived out of time).
        }

        @Override
        protected void dispatchResponseToAppropriateHandler(final Message msg) {
}

    }
Life Before SPLAY

- Time spent on developing testbed specific protocols
- Or fallback on simulations...
- The focus should be on ideas
- Researchers usually have no time to produce industrial-quality code

there is more :-(

Distributed Systems Made Simple - V. Schiavoni - Lua Workshop 2011, Frick, Switzerland
Lines of pseudocode $\sim$ Lines of executable code
Live Demo

www.splay-project.org
• Distributed systems raise a number of issues related to their evaluation

• Their implementation, debug, deployment and tuning is hard

• **SPLAY** leverages Lua to produce an easy to use yet powerful working environment to solve these issues
Backup Slides
The Big Picture
The Big Picture

Splay Controller

SQL DB
The Big Picture
The Big Picture

Shell | Web

REST API

Splay Controller

SQL DB
The Big Picture
The Big Picture

Shell

Web

REST API

Splay Controller

SQL DB

jobs

churn

blacklist
The Big Picture

Shell

REST API

Splay Controller

SQL DB

Web

require"splay.base"

alpha

churn

jobs

blacklist

jobs

blacklist
The Big Picture

Shell | Web
REST API
Splay Controller
SQL DB

require"splay.base"
...
The Big Picture
The Big Picture

Shell | Web
---|---
REST API

Splay Controller

SQL DB

ctrl

log

log

log

require"splay.base"

alpha

churn

jobs

blacklist
The Big Picture

Shell

REST API

Splay Controller

SQL DB

Web

jobs

churn

blacklist

require"splay.base"

alpha

Lua

Splayd

Ctrl

Log

Log

Log