RPC

Dave Eckhardt
de0u@andrew.cmu.edu

Synchronization

- No Lecture Friday
- Book report assignment
  - On Homework page, due mid-April
  - Book approval hotline: de0u+books@andrew
  - Paper collections fine
    - Send: titles, URLs, page counts
    - ***Avoid: "Some Exokernel papers"***
- Today: RPC
  - Text: 4.6 (far from exhaustive)

Overview

- RPC = Remote ***Procedure Call***
- Extends IPC in two ways
  - IPC = Inter-***Process*** Communication
    - OS-level: bytes, not objects
  - IPC restricted to single machine
- Marshalling
- Server location
- Call semantics

RPC Model

- Approach

```
d = computeNthDigit(CONST_PI, 3000);
```

  - Abstract away from “who computes it”
  - Should “work the same” when remote Cray does
- Issues
  - Must specify server ***somehow***
  - What “digit value” is “server down”?
    - Exceptions useful in “modern” languages

Marshalling

- Values must cross the network
- Machine formats differ
  - Integer byte order
    - www.scieng.com/ByteOrder.PDF
  - Floating point format
    - IEEE 754 or not
  - Memory packing/alignment issues

Marshalling

- Define a “network format”
  - ASN.1 - “self-describing” via in-line tags
  - XDR – not
- “Serialize” language-level object to byte stream
  - Rules typically recursive
    - Serialize a struct by serializing its fields in order
  - Implementation probably should ***not*** be

Marshalling

- Issues
  - Some types don't translate well
    - Ada has ranged integers, e.g., 44..59
    - Not everybody really likes 64-bit ints
    - Floating point formats are religious issues
  - Performance!
    - Memory speed ≅ network speed
  - The dreaded “pointer problem”

Marshalling

```
struct node {
  int value;
  struct node *neighbors[4];
}

n = occupancy(nodes, nnodes);
bn = best_neighbor(node);
i = value(node);
```

- Implications?

Marshalling

```
n = occupancy(nodes, nnodes);
```

- Marshall array – ok

```
bn = best_neighbor(node);
```

- Marshall graph structure – not so ok

```
i = value(node);
```

- ***Avoiding*** marshalling graph – not obvious

Server location

- Which machine?
  - Multiple AFS cells on the planet
  - Each has multiple file servers
- Approaches
  - Special hostnames: ***www***.cmu.edu
  - Machine lists
    - AFS CellSrvDB /usr/vice/etc/CellServDB
  - DNS SRV records (RFC 2782)

Server location

- Which port?
  - Must distinguish services on one machine
  - Fixed port assignment
    - AFS: fileserver UDP 7000, volume location 7003
    - /etc/services or www.iana.org/assignments/port-numbers
    - RFC 2468 www.rfc-editor.org/rfc/rfc2468.txt
  - Dynamic port assignment
    - Contact “courier” / “matchmaker” service via RPC
    - ...on a fixed port assignment!

Call semantics

- Typically, caller blocks
  - Matches procedure call semantics
- Blocking can be expensive
  - By a factor of a million!
- Asynchronous RPC
  - Transmit request, do other work, check for reply
  - Like programming language “futures”

Call Semantics

- Batch RPC
  - Send ***list*** of procedure calls
  - Later calls can use results of earlier calls
- Issues
  - Abort batch if one call fails?
    - Yet another programming language?
  - Typically wrecks “procedure call” abstraction
    - Must make N calls before 1st answer

Call Semantics

- Batch RPC Examples
  - NFS v4 (maybe), RFC 3010
  - Bloch, A Practical Approach to Replication of Abstract Data Objects

Call semantics

- Network failure
  - Retransmit
    - How long?
- Server reboot
  - Does client deal with RPC session restart?
  - Did the call “happen” or not?

Client Flow

- Client code calls ***stub*** routine
  - "Regular code" which encapsulates the magic
- Stub routine
  - Locates communication channel
    - Else: costly location/set-up/authentication
  - Marshals information
    - Including procedure #
  - Sends message, awaits reply
  - Unmarshals reply, returns

Server Flow

- Thread/process pool runnning ***skeleton*** code
- Skeleton code
  - Waits for request
  - Locates client state
    - Authentication/encryption context
  - Unmarshals parameters
  - Calls “real code”
  - Marshals reply
  - Sends reply

Deployment

- Define interface
  - Get it right, you'll live with it for a while
- Run stub generator
- Link stubs with client & server
- Run a server!

Java RMI

- ***R***emote ***M***ethod ***I***nvocation
- Serialization: programmer/language cooperation
  - ***Dangerously*** subtle!
    - Bloch, Effective Java
- RMI > RPC
  - Remote methods ≅ remote procedures
  - ***Parameters*** can be (differently) remote
    - Client on A can call method on B passing object on C (slowly)

Summary

- RPC is lots of fun
- So much fun that lots of things don't do it
  - SMTP
  - HTTP
- Read Effective Java
- No class Friday