Today's Objectives

- Remote Procedure Calls

Web Server Reminders

- I can only see what's in the GitHub repo, not what's in your local repo
- Include your tests (web pages, image files, scripts, etc.) in your repo
- Include your writeup as a PDF in your repo
- GitHub makes a snapshot of your repo at the deadline.

Review

- Services: What is harvest vs yield?

Harvest and Yield

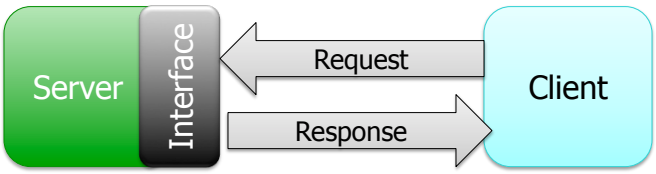
- *yield = queries completed/queries offered*
  - In some sense more interesting than availability because it focuses on client perceptions rather than server perceptions
  - If a service fails when no one was accessing it…
- *harvest = data available/complete data*
  - How much of the database is reflected in each query?
- Should faults affect yield, harvest, or both?

Distributed Systems Programming

- Distributed programming is challenging
  - Sockets: connect; read/write; disconnect
- Need common ***primitives/abstractions*** to hide complexity
- Communication: Message Passing Interface (MPI)
  - `send` and `receive` calls with the data
  - Explicit communication between sender, receiver

Goal: Make distributed computing look like local computing

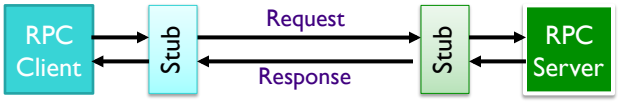
Remote Procedure Call (RPC)



- Server advertises *interface*, i.e., procedures that client can execute
- To client, looks like a *local* procedure call
- Execution on client stops while server executes the procedure and returns the result

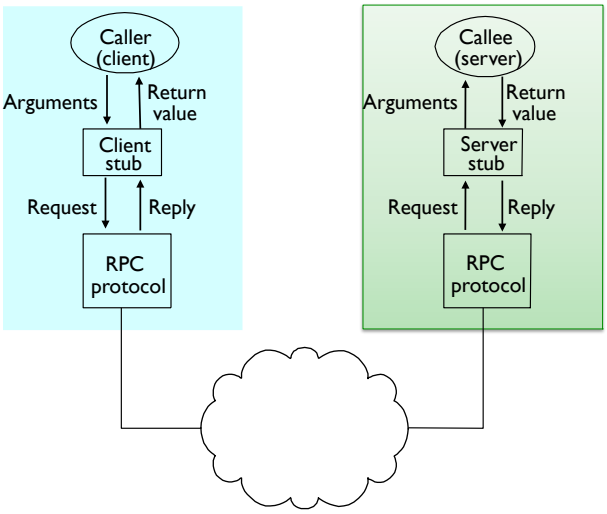
What does a local procedure call look like?

Remote Procedure Call



- Stubs translate from local call to remote call (or vice versa)
- Client stub indicates which procedure should run at server
  - Packs arguments into a message, called *marshalling* arguments
- Server stub unmarshals arguments
  - Big switch statement to determine local procedure to run

RPC Components



RPC Mechanics

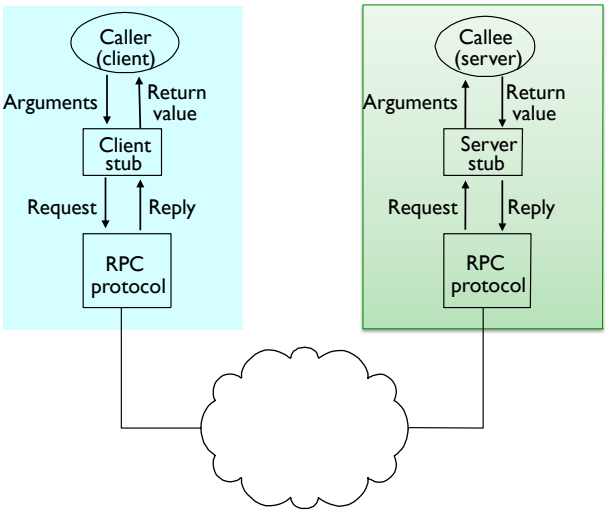
- Client issues request by calling stub procedure
  - Stubs can be automatically generated with compiler support
- Stub procedure marshals arguments, transmits requests, blocks waiting for response
  - RPC layer deals with network issues (e.g., TCP vs. UDP)
- Server stub
  - Unmarshals arguments
  - Determines correct local procedure to invoke, computes
  - Marshals results, transmits to client
  - Can also be automatically generated with language support

RPC Benefits

- Procedure call interface
- Writing application is simplified
  - Hides all network code into stubs
  - App programmers don't have to worry about sockets, headers, formatting, etc.

RPC Components

What are some places that we may run into trouble?



Remote Procedure Call Issues

- Underlying protocol
  - UDP vs. TCP
  - Advantages/Disadvantages?
- Semantics
  - What happens on network/host failure?
  - No *fate sharing* (compared to local machine case)
- Transparency
  - Hide network communication/failure from programmer
  - With language support, can make remote procedure call "look just like" local procedure call

Remote Procedure Call Issues

- Idempotent operations
  - Idempotency means that executing the same procedure does not have visible side effects
  - Can you re-execute operations without harmful side effects?
  - Examples?
- Timeout value
  - How long to wait before re-transmitting request?

RPC Semantics

- Local procedure calls: *exactly once*

Delivery Guarantees			RPC Call Semantics
Retry Request	Duplicate Filtering	Retransmit Response	
No	N/A	N/A	Maybe
Yes	No	Re-execute Procedure	At-least once
Yes	Yes	Retransmit reply	At-most once

Can't tell if remote method executed

Causes problems if not idempotent

Most RPC systems offer at-least once or at-most once

Presentation Formatting

- **Marshaling** (encoding) application data into messages
- **Unmarshaling** (decoding) messages into application data

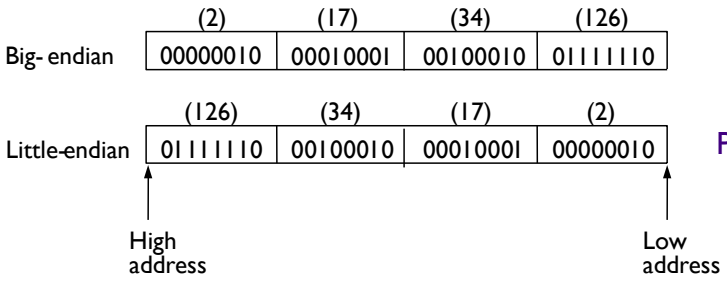
- Data types to consider:
  - integers
  - floats
  - strings
  - arrays
  - structs

- Types of data we do not consider:
  - images
  - video
  - multimedia documents

RPC Difficulties

- Representation of base types
  - Floating point: IEEE 754 versus non-standard
  - Integer: big-endian versus little-endian

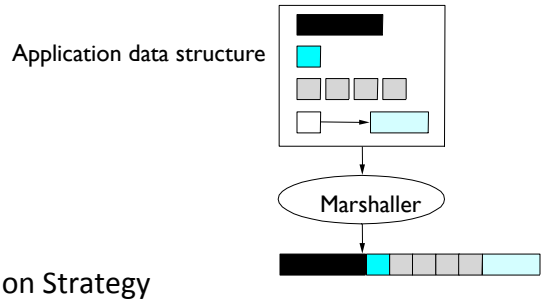
Example:



- Compiler layout of structures

Taxonomy

- Data types
  - Base types (e.g., ints, floats); must convert
  - Flat types (e.g., structures, arrays); must pack
  - Complex types (e.g., pointers); must linearize

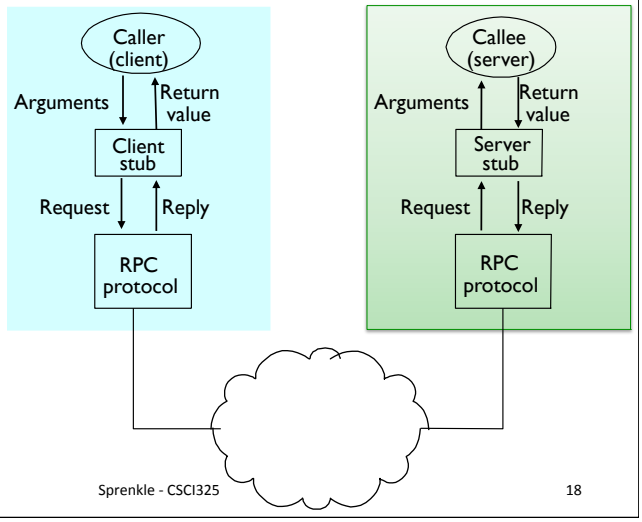


- Conversion Strategy
  - Canonical intermediate form (`htonl()` and `ntohl()`)
  - Receiver-makes-right

Performance Issue

- Client blocks while remote procedure executes …

Can't make progress on other code while waiting



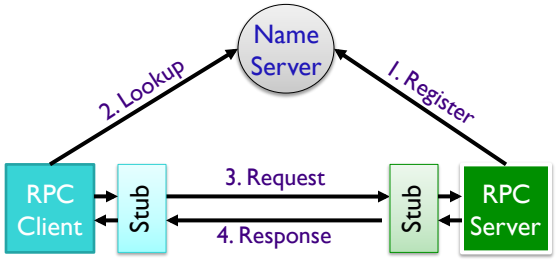
Performance Issue

- Client blocks while remote procedure executes …
- Does the client have to block? Is there any computation it can do while the server processes the request?

Solution: Asynchronous RPCs

- Client does not block
- Continues operation until actually needs the response back from the server to do useful work
- Idea: Overlap communication (remote) and computation (local)
- Overlap process can be coded manually
  - Open Question: Can process be done automatically? How well?
  - Researchers are working on leveraging compilers to determine automatically what computation/ communication can be overlapped

RPC Binding



- Binding
  - Servers register *service* they provide to a *name server*
  - Clients lookup services they need from server
  - Bind to server for a particular set of remote procedures
  - Operation informs RPC layer which machine to transmit requests to

Case Study: XML-RPC

- XML-RPC is an RPC protocol
  - XML to encode its calls
  - HTTP as transport mechanism
- Apache's Java implementation
  - `http://ws.apache.org/xmlrpc`

Client.java

```
public class Client {
    public static void main(String[] args) {
        XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
        String serverURL = "http://" + args[0] + ":" + 8888;
        try {
                config.setServerURL(new URL(serverURL));
        } catch (MalformedURLException e1) {
                e1.printStackTrace();
        }
        XmlRpcClient client = new XmlRpcClient();
        client.setConfig(config);
        ArrayList<Integer> params = new ArrayList<Integer>();
        params.add(10);
        params.add(40);

        try {
            Object[] result = (Object[]) client.execute(
                        "sample.SumAndDifference", params.toArray());
            int sum = ((Integer) result[0]).intValue();
            System.out.println("The sum is: " + sum);
            int diff = ((Integer) result[1]).intValue();
            System.out.println("The difference is: " + diff);
        } catch (XmlRpcException exception) {
            System.err.println("Client: " + exception);
        }
    }
}
```

Transparency breaks

client.py

```
import xmlrpc.client

proxy = xmlrpc.client.ServerProxy("http://
localhost:8888/")

result = proxy.sample.sumAndDifference(5, 6)
sum = result[0]
difference = result[1]
print("Sum: ", sum)
print("Difference: ", difference)
```

Server.java

```
public class Server {
    public Integer[] SumAndDifference(int x, int y) {
        Integer[] array = new Integer[2];
        array[0] = new Integer(x + y);
        array[1] = new Integer(y - x);
        return array;
    }

    public static void main(String[] args) {
        try {
            WebServer server = new WebServer(8888);
            XmlRpcServer xmlRpcServer = server.getXmlRpcServer();
            PropertyHandlerMapping phm = new PropertyHandlerMapping();
            phm.addHandler("sample", Server.class);
            xmlRpcServer.setHandlerMapping(phm);
            server.start();
        } catch (Exception exception) {
            System.err.println("Server: " + exception);
        }
    }
}
```


PropertyHandlerMapping

- A handler mapping based on a property file
- The property file contains a set of properties
  - The property *key* is the handler name
  - The property *value* is the name of a class being instantiated
- For every non-void, non-static, and public method in the class, an entry in the handler map is generated
- A typical use would be to specify interface names as the property keys and implementations as the values

Case Study: XML-RPC

- XML is a standard for describing structured documents
  - Uses tags to define structure:
    - <tag> … </tag> demarcates an element
    - Tags have no predefined semantics …
  - Elements can have attributes
    - Encoded as name-value pairs
    - A well-formed XML document corresponds to an element tree

```
<?xml version="1.0"?>
<methodCall>
    <methodName>SumAndDifference</methodName>
      <params>
          <param><value><i4>40</i4></value></param>
          <param><value><i4>10</i4></value></param>
      </params>
</methodCall>
```


(thanks to Vijay Karamcheti)

XML-RPC Wire Format

- Scalar values
  - Represented by a <value><type> … </type></value> block
- Integer, 4-byte signed integer
  - <i4>12</i4>
- Boolean
  - <boolean>0</boolean>
- String
  - <string>Hello world</string>
- Double
  - <double>11.4368</double>
- Also Base64 (binary), Date/Time, etc.

XML-RPC Wire Format (struct)

- Structures
  - Represented as a set of <member>s
  - Each member contains a <name> and a <value>

```
<struct>
    <member>
      <name>lowerBound</name>
      <value><i4>18</i4></value>
    </member>
    <member>
      <name>upperBound</name>
      <value><i4>139</i4></value>
    </member>
</struct>
```


XML-RPC Wire Format (Arrays)

- Arrays
  - A single <data> element, which contains any number of <value> elements

```
<array>
    <data>
     <value><i4>12</i4></value>
     <value><string>Egypt</string></value>
     <value><boolean>0</boolean></value>
     <value><i4>-31</i4></value>
    </data>
</array>
```

A 4-element array

XML-RPC Request

- HTTP POST message
  - URI interpreted in an implementation-specific fashion
  - Method name passed to the server program

```
POST /RPC2 HTTP/1.1
Content-Type: text/xml
User-Agent: XML-RPC.NET
Content-Length: 278
Expect: 100-continue
Connection: Keep-Alive
Host: localhost:8080

<?xml version="1.0"?>
<methodCall>
  <methodName>SumAndDifference</methodName>
  <params>
    <param><value><i4>40</i4></value></param>
    <param><value><i4>10</i4></value></param>
  </params>
</methodCall>
```

(Indentation for readabilty)

XML-RPC Response

- HTTP Response
  - Lower-level error returned as an HTTP error code
  - Application-level errors returned as a <fault> element (next slide)

```
HTTP/1.1 200 OK
Date: Mon, 22 Sep 2003 21:52:34 GMT
Server: Microsoft-IIS/6.0
Content-Type: text/xml
Content-Length: 467

<?xml version="1.0"?>
<methodResponse>
  <params><param>
  <value><struct>
   <member><name>sum</name><value><i4>50</i4></value>
   </member>
   <member><name>diff</name><value><i4>30</i4></value>
   </member>
  </struct></value>
  </param></params>
</methodResponse>
```

XML-RPC Fault Handling

- Another kind of MethodResponse

```
<?xml version="1.0"?>
<methodResponse>
<fault>
<value><struct>
<member>
<name>faultCode</name>
<value><i4>500</i4></value>
</member>
<member>
<name>faultString</name>
<value><string>Arg 'a' out of range</string></value>
</member>
</struct></value>
</fault>
</methodResponse>
```


Asynchronous RPCs

- executeAsync()
- org.apache.xmlrpc.AsyncCallback
  - handleResult
  - handleError

Thrift

- "The Apache Thrift software framework, for scalable cross-language services development, combines a software stack with a code generation engine to build services that work efficiently and seamlessly between C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml and Delphi and other languages."

- Originally developed by Facebook
- Open-source on Apache in 2007
- http://thrift.apache.org/

PROJECT 2

Project 2 Overview: Tiny Bookstore

- Implement 3 servers and a client
- All communicate using XML-RPC
  - Apache XMLRPC for Java
  - Client → FrontEnd
  - FrontEnd → Order/Catalog
  - Order → Catalog
- Concurrent queries
- One purchase at a time
- Create a Python client