IMPORTANT DATES

- **Sept 21**
  - Deadline for forming groups (2 people per group).
  - Email the TA (Name and Netid of the members)

- **Oct 22**
  - Project 1 submission and demo.

- **Oct 27**
  - Project 2 tutorial.

- **Dec 1**
  - Project 2 submission and demo.
PROJECT OBJECTIVE

- Implement client-server applications using
  - Socket
  - RPC (Remote Procedure Call)
  - RMI (Remote Method Invocation)
  - SOAP (Simple Object Access Protocol)
  - REST (REpresentational State Transfer)
  - Apache Thrift
  - Google Protocol Buffer

- Each application must have two implementations
  - **Client**: Generate fifteen (15) arrays of 15,000 random numbers (double) and send them to a server.
  - **Server**: Calculate the average (arithmetic mean) of each array and return the array with the highest mean.
**PROJECT OBJECTIVE (CONT.)**

- **Scenario 1**
  - A client sends its arrays to the server and waits for results. Repeat this process 30 times.
  - Compute the average round-trip time per call.

- **Scenario 2**
  - Variable number of clients send multiple calls (15 calls) simultaneously to the server.
  - Compute the average round-trip time per call.

<table>
<thead>
<tr>
<th>Number of Clients</th>
<th>Avg. Response Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>nnn</td>
</tr>
<tr>
<td>...</td>
<td>nnn</td>
</tr>
<tr>
<td>10</td>
<td>nnn</td>
</tr>
</tbody>
</table>
DELIVERABLES

- **Project Demo (20 marks)**
  - You should set up and try the demo well in advance on the machines where you will be demonstrating.
  - Submit the source code electronically.
  - Duration: Maximum **20 minutes** per group.
  - Time and location: **TBA**

- **Report (10 marks)**
  - A brief overview of the seven technologies.
  - Comparing the results with graphs or charts.
  - Assumptions and implementation details.
  - Who did what in the project?
SOCKET

- Originated with 4.2 BSD UNIX, at UC Berkley, 1983
- A socket is the combination of an IP address and port number
- Applications or processes could use sockets to remotely receive and send data to each other
- Unlike RPC, RMI and the like, sockets do not involve remote function calling; they merely provide a means to send and receive data
- Datagram sockets use UDP; Stream sockets use TCP
SOCKET DIAGRAM

Server Side
- ServerSocket()
  - bind()
  - listen()
  - accept()
  - Socket()
    - read()
    - write()
    - close()

Client Side
- Socket()
  - connect()
  - write()
  - read()
  - close()
SOCKET: REFERENCE

- UNIX POSIX Socket API (for C)

- Java Socket API (for Java)
  - Jim Farley, "Java Distributed Computing," O'Reilly, 1998
REMOTE PROCEDURE CALLS (RPC)

- A way to transparently call a procedure on a remote system (a server) from a host on another system (a client).

- Roughly three components: client code, server code, protocol code.

- The protocol involves the "interface" of the remote procedure, namely the procedure name, argument types and return type.

- This protocol can be written in RPCL, which will then be compiled into C code by a tool called rpcgen.

- Both the client and server code must include the protocol.
RPC: REFERENCE

- UNIX POSIX RPC API (for C)
REMOTE METHOD INVOCATION (RMI)

- Object oriented RPC; allows distributed objects instead of just distributed procedures.

- While RPC involves defining a protocol in RPCL, RMI involves writing a Java interface for the distributed object; this object must extend the java.rmi.Remote interface.

- The RMI registry is used to bind remote objects to names, such that clients can look up the objects by address and name.
RMI Diagram

Client
- Lookup (Interface)
  - Stubs

Server
- Binding (Implementation)
  - Skeletons

Remote Reference Layer (RRL)

Transport Layer
- Network
  - Transport Layer
RMI: REFERENCE

A standard for Web Service communication that provides support for:
- Remote procedure call (RPC) to invoke methods on servers
- Messaging to exchange documents

Based on XML

Language independent
SOAP DIAGRAM

Simple Object Access Protocol

- Client
- Client Proxy
- Web Service
- SOAP Request
- SOAP Message
- SOAP Response
- Phase I Serialize
- Phase II DeSerialize
- Phase III Serialize
- Phase IV DeSerialize

Network
SOAP: REFERENCE

- [http://www.w3schools.com/webservices/ws_soap_intro.asp](http://www.w3schools.com/webservices/ws_soap_intro.asp)
- [http://ralitechpe.blogspot.ca/](http://ralitechpe.blogspot.ca/)
REST is an architectural style, not a standard

REST emphasizes
- Scalability of component interactions
- Generality of interfaces
- Independent deployment of components
- Support for intermediaries
KEY PRINCIPLES OF REST

- "Identification of resources"
  - Resources are anything that can be named
  - Naming authority assigns an identifier to a resource

- "Manipulation of resources through representations"
  - Representations capture current/intended state of resources
  - Representations are transferred between components
  - Representations often contain links to related resources

- "Self-descriptive messages"
  - Resource identifiers, representation data formats, control data

- "Hypermedia as the engine of application state"
  - Servers are stateless, messages are independent
  - Clients maintain state (representations) & traverse links

- Induces scalability, generality, evolvability, extensibility
REST DIAGRAM
REST: REFERENCE

- [ksim.tistory.com/attachment/fk1400000000002.ppt](ksim.tistory.com/attachment/fk1400000000002.ppt)
GOOGLE PROTOCOL BUFFER

- Language independent data-serialization format developed by Google.
- Significantly faster and smaller than XML.
- Types of the messages are defined in .proto files.
- A message is a small logical record of a pairs of names and values.
- Transmitted information is encoded into binary format.
- Google provides necessary tools for binding this protocol with C++, Java and Python.
- Does not provide a RPC stack.
PF DIAGRAM

- .proto file
  - protocol buffer compiler
    - => .java, .cc, .py
      - Client code
      - Client stub
      - Message classes
      - Server stub
      - Server code

- javac, jar, gcc
  - compiler/linker
    - Client-side program
    - 3rd-party library
    - Server-side program

RPC
PF: REFERENCE

APACHE THRIFT

- Software framework for scalable cross-language services development.
- Originally developed at Facebook. Open sourced in April 2007.
- Easy exchange of data.
- Cross language serialization with minimal overhead.
- Thrift tools can generate code for C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk and Ocaml.
- Provides a RPC stack.
Thrift Code Generator Tool
(written in C++)

Create a thrift file
e.g., demo.thrift

Define Data types
and Service
interfaces

Build Thrift
platform files

Create Server/Client App
Run the Server

Demo.php
Demo.cpp
Demo.py
Demo.java

Server
implements
Services and
Client calls them
THRIFT: REFERENCE

- https://en.wikipedia.org/wiki/Apache_Thrift
- https://thrift.apache.org/
- http://www.slideshare.net/indicthreads/indic-threads-java10thrift
**ADDITIONAL NOTES**

- Any RPC (e.g., JAX-RPC, XML-RPC) may be used instead of C's "traditional" RPC API if desired.
- Your application should not only run with localhost and local port, but also with another PC connected to LAN.
SEVERAL WEBSITES MIGHT HELP YOU!

- Java Socket
  - http://cs.lmu.edu/~ray/notes/javanetexamples/
- RPC
- RMI
SEVERAL WEBSITES MIGHT HELP YOU!

- **SOAP**

- **REST**
  - [https://code.google.com/p/staff/wiki/BriefExamples](https://code.google.com/p/staff/wiki/BriefExamples)

- **Protocol Buffer**
  - [https://developers.google.com/protocol-buffers/docs/javatutorial?hl=en](https://developers.google.com/protocol-buffers/docs/javatutorial?hl=en)

- **Thrift**
  - [https://thrift.apache.org/tutorial/](https://thrift.apache.org/tutorial/)