Concurrent Programming and the Shared State

This semester we have been looking at concurrent programming and how it is supported on a single physical machine by Java. Concurrency can take place on a single-core machine or a multi-core machine.

Under the model we have been exploring, we assume a shared state exists in shared memory and have discussed how to guard that shared state and how to keep certain information out of the shared state.
**Distributed Computing**

We will now look at a different model where processing is meant to be performed on different physical machines without shared memory.

For ease of implementation we can (and will) simulate this on a single machine but the concept is meant to be applied to multiple machines connected via a network.

---

**Why distributed? Scalability.**

A significant advantage to distributed computing is its potential for scalability.

With in-machine concurrency we are restricted by the advances made in multi-processor and multi-core technology.

With distributed computing systems could theoretically scale with the only limits being imposed by cost and network bandwidth.
**Why distributed? Fault-tolerance.**

Distributed computing can also be utilized to provide a level of fault-tolerance in a system. If you had a 16-processor machine with one power supply or one mother board or one network card or one hard drive, the failure of one takes down all 16 processing units. If you have 16 machines, each having one CPU and one power supply one mother board, etc. then you might lose one but still have the others up and running.

It should be noted that to design a truly fault-tolerant system, you need to consider many factors. For example, Amazon’s distributed system in Virginia (which served clients around the world) was brought down both when power to the area was disrupted and when a bad software upgrade propagated through the entire cluster of machines.

---

**Design for failure of individual parts...**

When designing a robust distributed system, you should design a system that expects failures of individual processing elements as well as failures in the network between the processing elements.

It’s also important to note that certain failures (or even certain normalcy) might look alike. Consider the question of how does one distinguish between “it’s taking a long time to process” and “the request to process never got there” and “the results never got back to us” and “the machine that was meant to do the processing is broken”?

We often want to assure that our distributed system is always available and has elements that can easily recover from faults.

Assure that the system is smart enough to prevent double-recording in the event of a mistaken failure (you thought it failed so re-assigned the task but then the original succeeds).
Dealing with reality of computers and networks...

Can we really rely on our network connection? No.
  –Can we assume our network connection is fast? No.
  –Can we assume our bandwidth is unlimited? No.
  –Can we assume our network is secure? No.
  –Can we assume the network tomorrow will look like the network today? No.

Can we assume there is a single person in charge of all the machines? No.

Can we assume that all of the machines used in a distributed system are the same type? No.

Be prepared for opportunities...

By designing a distributed system that is meant to be scalable and tolerant of heterogeneity, you can more easily put servers near where the clients will be (or where they suddenly pop up).

You can also more readily react to increases in demand (you might want to avoid having to redesign when you find your service is successful and you start getting a wave of new customers because those customers might vanish before you improve your system).
Seize opportunities…

If you have a project that might receive support of everyday people, you might be able to capitalize on their idle computing power.

The SETI@Home’s screensaver/data-processor project did exactly that… …of course even as a distributed processing project, things still went astray when an air conditioning unit failed in the server room.

Security

Another consideration (though one that we won’t address in this course) is that of security. With a distributed system we might need to have a secure way to convey information (which might be actual programs to run) without it being intercepted and seen or intercepted and modified en-route or have information injected from others pretending to be part of the system.
What does distributed computing look like?

There are MANY forms of distributed computing. At a very simple level, consider a web page that contains a Javascript program or a Java applet. The web server sends the web client a program and some data. The web client then runs the program using the data. In some cases, new data is sent to the client as it requests it.

You can also have uses that might not typically be thought of as “computing” examples; such as networked file systems.

You can also have uses that might not typically be thought of as “computing” examples; such as networked file systems.

Consider how e-mail works.

A common example of common distributed computing is the SMTP foundation behind our current Internet e-mail system. There are MANY clients and MANY servers around the world. In the days before security was stepped up, I could connect to any SMTP server that I knew of and give it a message to send out and it would do so. Messages are sent from system to system until they get delivered to the recipient. Unlike in early e-mail systems where routing paths were explicitly listed, with SMTP if one server fails, as long as that server isn’t the only one for the recipient, the e-mail still gets delivered, just via some other path.
**Message-passing rather than shared memory**

When you are working with (or simulating the use of) a distributed computing cluster, you no longer have a single shared memory at your disposal. For example, all of the threads on a server might have a shared state, but the client does not have direct access to it.

What you often do is pass information back and forth between a client and a server.

These messages can contain raw data but they can also contain data structures and program code as well.

For the message passing itself, you should consider whether you want things to be synchronous or asynchronous.

---

**Tools to help us!**

There are both language-specific and general-purpose systems that have been designed to support the development of distributed computing systems. Some of these were designed to “hide” the fact that some things were being handled remotely (good for the programmer if all goes well – not so good when things fail).

There have been custom-built systems (early supercomputing clusters or event more recent systems like IBM’s “Blue Gene” series of systems – though not “distributed” really) and also “off the shelf” systems where all of the components can be found in a typical computer or electronics store (supposedly a significant part of Google).

Some common standards/implementations:
- SunRPC(Sun)
- DCOM(Microsoft)
- RMI(Java)
- CORBA(universal)
Programming distributed systems in Java

Java RMI (Remote Method Invocation) package (`java.rmi.*)` is its version of an RPC architecture (Remote Procedure Calls).

When calling a remote object’s method via a stub, the parameters to it are marshaled (basically serialized into a byte stream) and sent across the network and then unmarshaled and given to the remote method.

That object’s method is then run in another thread (RMI controls this, not you) on the server machine.

You need to make sure that any accesses made to the method’s object or any shared state to which that object has access are done in a thread-safe manner.

The Remote interface

The purpose of the `java.rmi.Remote` interface is to be extended by other interfaces. It will then identify the methods in those new interfaces as ones that can be invoked by a remote JVM.

```java
public interface RMIgobetween extends Remote {
    int displaySingleMessageOnServer(String s) throws RemoteException;
}
```
Starting a registry running

There needs to be a registry up and running so that the server of an object can register the object and so that clients can know where to find that object.

This can be done in a variety of ways. For today, we'll write a little program that sets one up and keeps it running.

```java
int theRegistryPort = 33011; //33=433, 01=from my login, 1=1
LocateRegistry.createRegistry(theRegistryPort);
System.out.println("Registry listening to port " + theRegistryPort);
System.out.println("Registry created, about to prevent registry app from terminating...");
stopper.await();
System.out.println("Registry ending.");
```

Creating a proxy for a remote object.

On the server side:

```java
RMIServer object = new RMIServer();
RMIGobetween stub = (RMIGobetween)
    java.rmi.server.UnicastRemoteObject.exportObject(object, 0);
theRegistryPort = 33011; //assumes on same machine
theRegistry = LocateRegistry.getRegistry(theRegistryPort);
theRegistry.bind("RMI_Server", stub);
```

On the client side:

```java
registry = LocateRegistry.getRegistry(
    theRemoteServerAddress, theRemoteServerPort);
theRemoteServerObject = (RMIGobetween)(registry.lookup("RMI_Server"));
```

The client now has a reference to a proxy for the remote object. Calls to the methods defined by the `RMIGobetween` interface can now be made from the client but will take place on the server. The actual object might have additional methods, but those are not directly accessible by the client. Parameters are sent to the method over the network and any return value is sent back over the network as well.
Passing parameters and return values
Primitives are obviously easy to pass back and forth over RMI as bytes.

Objects that implement the `Serializable` interface in Java (Java for example) can also be passed back and forth as bytes.

We can also have our own classes implement this interface by having an instance variable `serialVersionUID` and methods `readObject` and `writeObject` (Java actually provides default handling that we can use for these if we want).

Models of distributed programming
There are many examples and applications of distributed systems. Some examples are…

- A server that waits for clients to ask for tasks to process.
- Many machines all running daemons waiting for a master system to tell them what to do.
- All remote machines doing the same task versus each doing different tasks.
- Having the client know what it will do with the data versus having the clients get data and a program to run to process the code.
- At an extreme, we can talk about things like parallel sorting where each machine compares two values, and then shares what it learns with all the other machines working on the job.
Example full RMI with dynamic code loading

http://docs.oracle.com/javase/tutorial/rmi/overview.html

Not just processing, but data too...

When thinking about distributed computing, we should think about distributed data and also how and what information gets transferred.

1000 terabytes = 1 petabyte

To transfer 1 petabyte in a day, you’d need to achieve around 12 gigabytes per second, so try to keep processing power close to the data…

According to a 2008 ACM article (“MapReduce: Simplified Data Processing on Large Clusters”) the machines in Google’s clusters were processing > 20 petabytes per day!

Facebook claims a single file system with over 100 petabytes

**Other resources?**


Networking – fiber optics technology can be your friend…

Real Estate issues:
- Square footage! – all those machines and drives and networking gear takes up space…
- Water! – all those machines and hard drives generate vast amounts of heat and water-cooling is your friend…
- Location, location, location! – being close to other data centers or to customers…

**How to make it all easier for programmers?**

A good amount of work trying to make it all transparent to the programmer (general purpose language features or compiler-based code transformations).

Current reality is a mix of language-based tools and conceptual models that align well with certain platforms and/or tools. Clever approaches can give ways to use old tools in new ways…