Administrative Items

- Handouts today
  - Syllabus

- Conventions in these slides
  - **Key terms defined are underlined**
  - Items for extended discussion are in red
    - Or URLs underlined and in red by PowerPoint (Thanks, Bill…)
  - Code fragments or something else you might type are usually in a typewriter font (Courier New)
    - Language keywords are also in **bold**
Outline of Topics

1. Introduction
2. Comparison of Distributed and Parallel Computing
3. Example Local vs. Remote Procedure Call
Introduction

• A distributed system is “one in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing”
  – Very broad definition
  – Lots of examples
  – Lots of kinds

• Abbreviations
  – “Distributed System” by “DS”,
  – “Distributed Computing” is “DC”

• “You know you have one when the crash of a computer you’ve never heard of stops you from getting any work done.” Leslie Lamport

• Examples of DSs:
Advantages of Distributed Systems

- Share resources (key)
- Share devices
- Better hardware cost/performance than supercomputers, multiprocessors
- Allows access from many remote users using their simple PCs
- Allows for incremental growth (if done right)
- Increases reliability and availability (if done right)
- Some applications and services are inherently distributed
- Can spread the load of a given service much more easily
- Can potentially increase security (!!!???)
Consequences of Distributed Systems

• Concurrency
  – Concurrent use of low-level resources: processing, storage (memory+disk), communications
  – Mutual exclusion and other synchronization required
  – Access to resources for a given user often best-effort

• No global clock
  – Cannot often know the exact ordering of events: which happened first

• Variable network delays

• Independent failures
  – No longer “all or none” failures for your program!
  – Some computers still running, while others failed or partitioned
  – Failure of a component you are using may not be a clean failure
Outline of Topics

1. Introduction

2. Comparison of Distributed and Parallel Computing

3. Example Local vs. Remote Procedure Call
Comparison: DC and Parallel Computing

• (Note: material from: Claudia Leopold, *Parallel and Distributed Computing: A Survey of Models, Paradigms, and Approaches*, John Wiley and Sons, 2001)

• Common characteristics
  – Multiple processors are used
  – Processors interconnected by some “network”
  – Multiple computational activities (processes) are in progress at the same time and cooperate with each other

• Some consider parallel computing a subfield of DC!
  – Very different….. (Kuwait PDC panel)

• Parallel computing splits an application up into tasks that are executed at the same time, whereas distributed computing splits an application up into tasks that are executed at different locations using different resources.
Differences: DC and Parallel Computing

• Parallel Computing puts emphasis on the following:
  – An application is split into subtasks that are solved simultaneously, often in a “tightly coupled” manner
  – One application is considered at a time, with the goal of speeding up the processing of that single application
  – Programs are generally run on homogeneous architectures, which typically have shared memory
  – Fault tolerance and security are not generally considered
Differences: DC and Parallel Computing (cont.)

- Distributed Computing puts emphasis on the following:
  - Computation uses multiple resources physically separated: processors, memory, disk, databases
  - Multiple applications run at a time for many users
  - Heterogenous systems, open and dynamic
  - No shared memory, at least not in hardware
  - Fault tolerance and security must be dealt with (in some manner)
  - Sometimes the emphasis is on hiding system internals in a manner that the distributed system looks like a single large machine. Feature called a *single system image*, used in *cluster computing*. 
Convergence of DC and Parallel Computing

• Architectures approaching each other
  – Fast network technologies allow cluster computing
  – Parallel machines increasingly used as servers in a DS

• Parallelism and distribution are closely related
  – Main differences in distribution: delay and partial failures

• Some joint meetings of parallel and distributed researchers

• Note: “concurrent programming” becoming much more common last 5 years with multi-core CPUs becoming standard
  – Somewhat different from parallel programming
Outline of Topics

1. Introduction
2. Comparison of Distributed and Parallel Computing
3. **Example Local vs. Remote Procedure Call**
   - A lot goes on under the hood in a local call in C
   - Even more has to happen for a remote call!
Example Local Call

**Caller:**
// declare and init stuff
x = new int [100];
y = new util; // create and rtn ref
flag = y.sort(x, 100);

**Callee:**
// declare and init stuff (not shown)
int util:sort(int [] a, int max) {
   // implementation of sort... bubble sort, quicksort, ...
   return status;
}
Reminder: Assembler

Example C-like call

\[ X = 4 + ((Y \times 4) / (A + B)) \];

Equivalent assembler (vars on stack)

- `ldr r1, [sp, Y]`  
  *load Y*
- `mul r1, r1, #4`  
  *Y * 4*
- `ldr r2, [sp, A]`  
  *load A*
- `ldr r3, [sp, B]`  
  *load B*
- `add r2, r2, r3`  
  *A + B*
- `div r1, r1, r2`  
  *divide the two*
- `add r1, r1, #4`  
  *add four to result*
- `str r1, [sp, X]`  
  *store result in X on stack*
Reminder: Calling Conventions

• To call a function or routine you need to push arguments on the stack (in the right order), push the return address on the stack, ….., branch to the routine, ….
• Calling conventions define this for a given compiler/language
• High-level language compilers do all this for you
• Have to program yourself if using assembler

Calling `myFunc()` in C:

```c
int main() {
    int x = 1;
    int y = 2;
    int z = myFunc(x, y);
}

int myFunc(int x, int y) {
    return x + y
}
```
Reminder: Calling Conventions

myFunc:

```assembly
movl  %edi, -4(%rbp)  !grab x off stack
movl  %esi, -8(%rbp)  !grab y off stack
add   %esi, %edi      !add x and y
movl  %esi, %eax      !return x + y
ret
```

.globl main

main:

```assembly
movl $1, -4(%rbp)    !x = 1
movl $2, -8(%rbp)    !y = 2
```

call myFunc

ret
Example Local Call (2)

• Potential assumptions between caller and callee:
  – Assembler calling conventions
  – In same address space (on same computer)
  – In same programming language (usually)
  – Same operating system
  – Same CPU type
  – Can transfer data and control quickly, effectively in zero time
  – Both fail, or neither do (for the most part)

• None of these assumptions are always true in a distributed system!
Example Remote Call

**Caller:**
// declare and init stuff
x = new int [100];
Y = new util.bind();
Flag = y.sort(x, 100);
...

**Callee:**
// declare and init stuff
int util_impl::sort(int[] a, int max){
    // implementation of sort
    return status;
}

// "proxy" or "stub"
// generated by middleware
int util::sort(int[] a, int max){
    // put a[], max into struct
    // send message with struct
    // receive message with struct
    // copy from struct to a[],
    // status
    return status;
}

// "skeleton" generated
// by middleware compiler
...

// receive message with struct
// copy from struct to a[], max
flag = z.sort(a, max)

// copy a[], flag into struct
// send message with struct
Many Local Call Assumptions don’t Hold!

• Not a local object Invocation, so need more help
  – Need remote equivalent of local (assembler) calling conventions
  – In this class we will come to understand this “plumbing” much better

• Not in same programming language (can’t assume)
• Not written by same programmer
• Not running same operating system for caller and callee
• Not same CPU type for caller and callee
• …
Many Local Call Assumptions don’t Hold! (2)

- Not always in the same administrative domain
- Latency for transfer of control and data can be large and, worse, unpredictable
- Partial failures
- Membership of the system (the computers in its collection) can change
- Unreliable or insecure communication
Bottom Line on Distributed Systems

I don’t think we are in Kansas anymore, Toto!

Goal of this class is to gain a basic understanding of:
- How and why you are no longer in Kansas
- What you can do about it!

Read [CDKB5] Chap. 1 by next lecture (won’t lecture), start on 2.