Introduction to Distributed Systems

Prof. Dave Bakken

Cpt. S 464/564 Lecture
Auxiliary Material (not from text)
January 28, 2020
(late due to instructor hospitalization)
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, tablets, phones, ...
• 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 & partial 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
Timeline of Distributed Computing

• **1969**: BBN builds the ARPANET, the first internet
  – Builds off of ideas from Kleinrock (UCLA), others

• **1970s**: figuring out how to do network programming
  – Rick Schantz joins BBN in 1973; PhD in 1974

• **1980s**: figuring out how to ship data structures around (to support distributed app programming)
  – Middleware came from this
    • 1981: Cronus (BBN), 1989: OMG (CORBA, later DDS)

• **1990s**: Quality of Service (QoS), mainly Layer3
  – Bakken at BBN 1994-1999

• **2000s**: Cloud Computing

• **2010s**: Pervasive usage of mobile phones
Where have we come from?

- Here is something on the evolution of accessing Unix over the years
- Here is a 1957 5MB hard drive.
- 4K RAM cost in late 1970s
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
Outline of Topics

1. Introduction
2. Comparison of Distributed and Parallel Computing
3. Example Local vs. Remote Procedure Call
   - A lot goes on “unde the hood” in a local (same-process) call in C
   - Even more has to happen for a remote call!
How does control actually pass from caller to called (and back)?
Reminder: Assembler

Example C-like call

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

Equivalent assembler (vars on stack)

- `ldr r1, [sp, Y]`: !load Y into r1
- `mul r1, r1, #4`: !Y * 4 into r1
- `ldr r2, [sp, A]`: !load A into r2
- `ldr r3, [sp, B]`: !load B into r3
- `add r2, r3, r4`: !A + B into r4
- `div r1, r4, r5`: !divide the two: r5
- `add r5, r6, #4`: !add four to result
- `str r6, [sp, X]`: !store result from stack in X
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 (Details)

**Potential assumptions:**

- Object Invocation conventions between caller ("client") and callee
- In same address space (on same computer)
- In same programming language (usually)
- Written by same programmer (often, not always)
- Same operating system for both caller and callee
- Same CPU type for both caller and callee
- 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!**

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

**Callee:**
```
// declare and init stuff
int util::sort(int [] a, int max) {
    // implementation of sort...
    return status;
}
```
**Example Remote Call**

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

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

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

// receive message w/ struct  
// copy from struct to a[], max  
flag = z.sort(a, max)  
// copy a[], flag into struct  
// send message with struct  
...
Many Assumptions do not Hold!

- Not a local procedure call, so need more help
- Not in same programming language (can’t assume this)
- Not written by same programmer
- Not running same operating system for caller and callee
- Not same CPU type for caller and callee
- 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
I don’t think we are in Kansas anymore, Toto!

But help is on the way!
Until Next Time...

Goal of this class: fully understand:

- How and why you are no longer in Kansas
- What you can do about it!

Read [CKDB5] Chap 1 , start on Chap 2
MIPS my_Func(i,j) FYI

my_Func:
daddiu $sp,$sp,-48
sd $fp,40($sp)
move $fp,$sp
move $3,$4
move $2,$5
sll $3,$3,0
sw $3,16($fp)
sll $2,$2,0
sw $2,20($fp)
lw $3,16($fp)
lw $2,20($fp)
addu $2,$3,$2
sw $2,0($fp)
lw $2,16($fp)
addiu $2,$2,-2
sw $2,4($fp)
lw $3,0($fp)
lw $2,4($fp)
addu $2,$3,$2
sw $2,0($fp)
lw $2,0($fp)
move $sp,$fp
ld $fp,40($sp)
daddiu $sp,$sp,48
j $31