Critical Infrastructure Security: The Emerging Smart Grid

Cpt S 580-04, Cpt S 483-02, EE 582-03, EE 483-01

Dr. Anurag K. Srivastava, Dr. Carl Hauser, Dr. Dave Bakken

Computation Lecture 4: Overview of Distributed Computing (1/2)
Today’s Content

1. Administrivia
2. Intro to Distributed Computing
3. Example Local vs. Remote Object Invocation
4. Examples (1.2)
5. Trends (1.3)
6. Focus on Resource Sharing (1.4)
Administrivia

• Computation segment HW coming in the next week
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
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 (and mobile devices)
- 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

• 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

• Variability
  ▪ Performance of resources (network, CPU, storage) may vary widely across a remote call chain
Today’s Content

1. Administrivia
2. Intro to Distributed Computing
3. Example Local vs. Remote Object Invocation
4. Examples (1.2)
5. Trends (1.3)
6. Focus on Resource Sharing (1.4)
Example Local Call

util
sort(x, 100)

util sort(int [] a, int max)
Example Local Call (BACKUP SLIDE)

**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;
}

• 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!
Reminder: Assembler

Equivalent assembler (vars on stack)

Example C-like call

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

\[ \text{ldr } r1, [sp, Y] \text{ !load } Y \]
\[ \text{mul } r1, r1, #4 \text{ ! } Y \times 4 \]
\[ \text{ldr } r2, [sp, A] \text{ !load } A \]
\[ \text{ldr } r3, [sp, B] \text{ !load } B \]
\[ \text{add } r2, r2, r3 \text{ ! } A + B \]
\[ \text{div } r1, r1, r2 \text{ !divide the two} \]
\[ \text{add } r1, r1, #4 \text{ !add four to result} \]
\[ \text{str } r1, [sp, X] \text{ !store result in } X \text{ on stack} \]
Reminder: Calling Conventions

- 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

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

int myFunc(int x, int y) {
    return x + y
}
```
**myFunc:**

**Reminder: Calling Conventions**

- `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:**

- `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:**

```java
// declare and init stuff
x = new int [100];
Y = new util.lookup(...);
Flag = y.sort(x, 100);
```

**Callee:**

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

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

// “skeleton” generated  
// by middleware compiler
```java
...  
// 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
Goal of these two lectures is to gain a basic understanding of:

• How and why you are no longer in Kansas
• Where to dig more to find info about what you can do about it!
Today’s Content

1. Administrivia
2. Intro to Distributed Computing
3. Example Local vs. Remote Object Invocation
4. Examples (1.2)
5. Trends (1.3)
6. Focus on Resource Sharing (1.4)
<table>
<thead>
<tr>
<th>Domain</th>
<th>Associated Networked Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finance and commerce</td>
<td>eCommerce e.g. Amazon and eBay, PayPal, online banking and trading</td>
</tr>
<tr>
<td>The information society</td>
<td>Web information and search engines, ebooks, Wikipedia; social networking: Facebook and MySpace.</td>
</tr>
<tr>
<td>Creative industries and entertainment</td>
<td>online gaming, music and film in the home, user-generated content, e.g. YouTube, Flickr</td>
</tr>
<tr>
<td>Healthcare</td>
<td>health informatics, on online patient records, monitoring patients</td>
</tr>
<tr>
<td>Education</td>
<td>e-learning, virtual learning environments; distance learning</td>
</tr>
<tr>
<td>Transport and logistics</td>
<td>GPS in route finding systems, map services: Google Maps, Google Earth</td>
</tr>
<tr>
<td>Science</td>
<td>The Grid as an enabling technology for collaboration between scientists</td>
</tr>
<tr>
<td>Environmental management</td>
<td>sensor technology to monitor earthquakes, floods or tsunamis</td>
</tr>
</tbody>
</table>
E.g., Google Infrastructure

- Underlying physical infrastructure: huge number of networked computers at data centers all over the world
- Distributed file system
  - Designed for very large files
  - Heavily optimized for style of usage by google apps (high rates)
- Associated structured distributed storage system: fast access to very large datasets
- Lock service: distributed locks and agreement
- Programming model supporting managing very large parallel distributed computations across infrastructure
Massive Multiplayer Online Games (MMOGs)

• Widely-spread users interacting via a persistent virtual world

• Huge challenge to engineer
  ▪ Fast response times for a user’s actions
  ▪ Scalability
  ▪ Propagating updates to other users
  ▪ Consistency

• Number of solution kinds
  ▪ Client-server: single state in one location (clever server)
  ▪ Distributing state to many servers (allocate users by usage patterns)
  ▪ Research into more radical: peer-to-peer
Financial Trading

• HUGE amount of money traded automatically
  ▪ Ergo on the cutting edge of distributed systems technology

• E.g., share prices, trends, economic and political development, ...

• Subscribe to items of interest, an event
  ▪ Publish-subscribe
  ▪ Distributed event-based systems
• **CEP**: compose event occurrences into logical, temporal, or spatial patterns

• Trading strategies NOT hard coded into the architecture!

• Looking for patterns that indicate a trading opportunity
Trading strategies example

WHEN

MSFT price moves outside 2% of MSFT Moving Avg

FOLLOWED-BY (  

MyBasket moves up by 0.5%

AND

HPQ’s price moves up by 5%

OR

MSFT’s price moves down by 2%

)

)

ALL WITHIN

any 2 minute time period

THEN

BUY MSFT; SELL HPQ
Today’s Content

1. Administrivia
2. Intro to Distributed Computing
3. Example Local vs. Remote Object Invocation
4. Examples (1.2)
5. Trends (1.3)
6. Focus on Resource Sharing (1.4)
Mobile and ubiquitous computing

- Computing devices smaller and smaller...
  - Laptops
  - Handhelds
  - Wearable & implantable devices
  - Embedded in appliances

- **Mobile computing**: performing tasks when not at “home”
  - Includes location-aware or context-aware

- **Ubiquitous computing**: harnessing lotsa small and cheap devices in user’s environment
  - Useful only when they can communicate!

- Some overlap, but distinct areas mostly
• With right toys, can get lotsa work done away from “home”: work, hotel room, partner company, ...

• Support **spontaneous interaction!** Service discovery
Distributed multimedia systems

• **DEFN**: Support range of media types in integrated manner

• Always includes a temporal dimension & preserving relationships ("continuous media")

• Access to huge amount of rich info (compared to text/html)

• Places huge demands on distributed infrastructure:
  - Support extensible range of encoding and encryption formats
  - Provide range of mechanisms to meet user’s **quality of service (QoS)**
  - Provide appropriate resource management strategies incl. scheduling
  - Provide adaptation strategies when QoS cannot be met
Distributed computing as a utility

- **Grid Computing** (e.g., Globus) ➔ **Cloud computing**
- Tries to tread distributed resources as a commodity or utility
- Analogies from power grid, networking cloud
- Applies to
  - Physical resources: storage, processing .... Often virtualized
  - Software services: email, calendars,
- (See [http://blogs.computerworld.com/18768/cloud_computing_philosophy](http://blogs.computerworld.com/18768/cloud_computing_philosophy))
- Clouds usually implemented on (huge) clusters
  - Example: Quincy, Washington & Grand County PUD
Figure 1.5 Cloud computing

Clients

Internet

Application services

Storage services

Computational services
Today’s Content

1. Administrivia
2. Intro to Distributed Computing
3. Example Local vs. Remote Object Invocation
4. Examples (1.2)
5. Trends (1.3)
6. Focus on Resource Sharing (1.4)
Focus in resource sharing

• Users (and app programmers) want to share higher-level resources, not lower-level ones

• Computer-supported cooperative work (CSCW) AKA workflow: docs

**Service**: distinct part of a computer system that
  - Manages a collection of related resources
  - Presents their functionality to users & apps (restricted access)

**Server**: a running program or computer on net that:
  - Accepts requests from programs on other computers
  - Performs a service
  - Responds accordingly

• Client (object), server (obj), client-server, remote invocation

• Calls can be nested!
Today’s Content

1. Administrivia
2. Intro to Distributed Computing
3. Example Local vs. Remote Object Invocation
4. Examples (1.2)
5. Trends (1.3)
6. Focus on Resource Sharing (1.4)
7. Bonus (if time): Parallel vs. Distributed computing
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….. (e.g., 1995 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*. 
• 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
    – Or perhaps a checkpoint every hour for a day-long computation
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 (maybe, eventually)

- Architectures (vaguely) 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: variable delays and partial failures

- Some joint meetings of parallel and distributed researchers