DISTRIBUTED SYSTEMS

Midterm Review

Dr. Jack Lange
Computer Science Department
University of Pittsburgh

Fall 2015

Distributed System Definition

A distributed system is

“A collection of independent computers that appears to its users as a single coherent system”
Definition Distributed System

- A distributed system organized as middleware.
- The middleware layer extends over multiple machines, and offers each application the same interface.

Distributed System – Transparency

<table>
<thead>
<tr>
<th>Transparency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Hide differences in data representation and how a resource is accessed</td>
</tr>
<tr>
<td>Location</td>
<td>Hide where a resource is located</td>
</tr>
<tr>
<td>Migration</td>
<td>Hide that a resource may move to another location</td>
</tr>
<tr>
<td>Relocation</td>
<td>Hide that a resource may be moved to another location while in use</td>
</tr>
<tr>
<td>Replication</td>
<td>Hide that a resource is replicated</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Hide that a resource may be shared by several competitive users</td>
</tr>
<tr>
<td>Failure</td>
<td>Hide the failure and recovery of a resource</td>
</tr>
</tbody>
</table>

Different Forms Of Transparency In A Distributed System (ISO, 1995).
INTRODUCTION

TYPES OF DISTRIBUTED SYSTEMS

Transaction Processing Systems

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN_TRANSACTION</td>
<td>Mark the start of a transaction</td>
</tr>
<tr>
<td>END_TRANSACTION</td>
<td>Terminate the transaction and try to commit</td>
</tr>
<tr>
<td>ABORT_TRANSACTION</td>
<td>Kill the transaction and restore the old values</td>
</tr>
<tr>
<td>READ</td>
<td>Read data from a file, a table, or otherwise</td>
</tr>
<tr>
<td>WRITE</td>
<td>Write data to a file, a table, or otherwise</td>
</tr>
</tbody>
</table>

Transaction Primitives
TPS – Transaction Characteristic Properties

- **Atomic** – To the outside world, the transaction happens indivisibly.
- **Consistent** – The transaction does not violate system invariants.
- **Isolated** – Concurrent transactions do not interfere with each other.
- **Durable** – Once a transaction commits, the changes are permanent.

Architectural Styles

- An architectural style describes a particular way to configure a collection of components and connectors.
  - Component - a module with well-defined interfaces; reusable, replaceable
  - Connector – communication link between modules
- Architectures suitable for distributed systems:
  - Layered architectures
  - Object-based architectures
  - Data-centered architectures
  - Event-based architectures
Architectural Styles –
Layered and Object-Based

Layered

Object-Based

Object based is less structured
Component = object
Connector = RPC Or RMI

Data-Centered Architectures

- Main purpose is data access and update
- Processes interact by reading and modifying data in some shared repository
  - Repository can be active or passive
    - Traditional data base – Passive repository responds to requests
    - Blackboard system – Active repository, where clients solve problems collaboratively and system updates clients when information changes.

- Web-based distributed systems are largely data centric
  - Processes communicate through shared Web-based data services
Architectural Styles – Event-Based

- Communication is via event propagation
- Often associated with Publish/Subscribe systems – register interest in market information and receive email updates
- Referential Decoupling
  - Loosely couples sender and receiver – space decoupling

Event-based arch. supports several communication styles:
- Publish-subscribe
- Broadcast
- Point-to-point

Centralized vs Decentralized Architectures

- Vertical Distribution – Traditional client-server architectures exhibit vertical distribution.
  - Each level serves a different purpose in the system.
  - Logically different components reside on different nodes
- Horizontal distribution – e.g., P2P architectures
  - Each node has roughly the same processing capabilities and stores and manages part of the total system data.
  - Better load balancing, more resistant to denial-of-service attacks, but harder to manage than C/S
  - Communication and control is not hierarchical, all nodes are peers with equal functionalities
Idempotency

- Retransmission, after timeout, is the typical response to lost request in connectionless communication
- Consider effect of re-sending a message such as “Increment X by 1000”
  - If first message was acted on, now the operation has been performed twice
- Idempotent operations can be performed multiple times without harm
  - “Return current value of X” and “Check on availability of a product” are idempotent
  - “Increment X”, “Order Product Y” are non-idempotent

Two-tiered Client/Server Architectures

- Thin-Client Architecture – Server provides processing and data management and client provides simple graphical display
  - Perceived performance loss at client
  - Easier to manage, more reliable, client machines don’t need to be so large and powerful
- Fat-Client Architecture – At the other extreme, all application processing and some data resides at the client
  - Pro – Reduces work load at server; more scalable
  - Con – Harder to manage, and potentially less secure
Multitiered Architectures

System Architecture

DECENTRALIZED ARCHITECTURE
Peer-to-Peer

- Nodes act as both client and server; interaction is symmetric
- Each node acts as a server for part of the total system data
- Overlay networks connect nodes in the P2P system
  - Nodes in the overlay use their own addressing system for storing and retrieving data in the system
  - Nodes can route requests to locations that may not be known by the requester.

Overlay Networks

- ONs are logical or virtual networks, built on top of a physical network
  - A link between two nodes in the overlay may consist of several physical links.
  - Messages in the overlay are sent to logical addresses, not physical (IP) addresses
  - Various approaches used to resolve logical addresses to physical.
Overlay Networks

- Each node in a P2P system knows how to contact several other nodes.
- The overlay network may be:
  - Structured – Nodes and content are connected according to some design that simplifies later lookups, or
  - Unstructured – Content is assigned to nodes without regard to the network topology

Structured Peer-to-Peer Architectures

The connections between nodes are logical connections, not necessarily physical connections.

Mapping of Data items onto nodes in Chord for $m = 4$
Collaborative Distributed Systems **BitTorrent**

- Clients contact a global directory (Web server) to locate a `.torrent` file with the information needed to locate a tracker;
  - A tracker is a server that can supply a list of active nodes that have chunks of the desired file.

- Using information from the tracker, clients can download the file in chunks from multiple sites in the network.
  - Clients must also provide file chunks to other users.

---

**PROCESSES AND THREADS**
### Process Address Space

<table>
<thead>
<tr>
<th>Name</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Segment (Text)</td>
<td>Program executable statements</td>
</tr>
<tr>
<td>Data Segment</td>
<td></td>
</tr>
<tr>
<td>1. Initialized Data</td>
<td>1. Static or global data initialized with non-zero values</td>
</tr>
<tr>
<td>2. Zero-Initialized or Block Started by Symbol (BSS) data</td>
<td>2. Uninitialized static or global data</td>
</tr>
<tr>
<td>Stack Segment</td>
<td></td>
</tr>
<tr>
<td>Stack Segment</td>
<td></td>
</tr>
<tr>
<td>• Local variables of the scope.</td>
<td></td>
</tr>
<tr>
<td>• Function parameters</td>
<td></td>
</tr>
</tbody>
</table>

### Process control block (PCB)

- PCB
  - state
  - memory
  - files
  - accounting
  - priority
  - user
  - CPU registers
  - storage

- kernel
  - text
  - data
  - heap

- user
  - text
  - data
  - heap

- CPU
  - PSW
  - IR
  - PC
  - SP
  - general purpose registers
Switching Between Processes

- An important task of the OS is to manage CPU allocation among concurrent processes
  - When the OS takes away the CPU from a running process, the OS must perform a switch between the processes
    - This referred to as context switch
      - It is also referred to as a “state save” and “state restore”

- What is a context?
- What operations must take place to achieve this switch?
- How can the OS guarantee that the interrupted process can resume correctly?

CPU Switch From Process to Process

<table>
<thead>
<tr>
<th>process $P_i$</th>
<th>operating system</th>
<th>process $P_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>executing</td>
<td>interrupt or system call</td>
<td>idle</td>
</tr>
<tr>
<td>idle</td>
<td>save state into PCB$_i$</td>
<td>executing</td>
</tr>
<tr>
<td>executing</td>
<td>reload state from PCB$_i$</td>
<td>idle</td>
</tr>
<tr>
<td>idle</td>
<td>save state into PCB$_i$</td>
<td>executing</td>
</tr>
<tr>
<td>executing</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Threads: “Processes” Sharing Memory

- Process == Address Space
- Thread == Program Counter / Stream of Instructions
- Two examples
  - Three processes, each with one thread
  - One process with three threads

Why Use threads?

- Allow a single application to do many things at once
  - Simpler programming model
  - Less waiting
- Threads are faster to create or destroy
  - No separate address space
- Overlap computation and I/O
  - Could be done without threads, but it’s harder
- Example: word processor
  - Thread to read from keyboard
  - Thread to format document
  - Thread to write to disk

When in the Course of human events, it becomes necessary for one people to dissolve the political bands which have connected them with another, and to assume among the powers of the earth, the separate and equal station to which the Laws of Nature and of Nature’s God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.

We hold these truths to be self-evident, that all men are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the pursuit of Happiness.--That to secure these rights, Governments are instituted among Men, deriving their just powers from the consent of the governed, --That whenever any Form of Government becomes destructive of these ends, it is the Right of the People to alter or to abolish it, and to institute new Government, laying its foundation on such principles and organizing its powers in such form, as to them shall seem most likely to effect their Safety and Happiness. Prudence, indeed, will dictate that Governments long established should not be changed for light and transient causes; and accordingly all
Implementing threads

- Thread switching does not involve the kernel – no need for mode switching
  - Fast context switch time,
  - Threads semantics are defined by application
  - Scheduling can be application specific
    - Best algorithm can be selected
  - ULTs are highly portable – Only a thread library is needed

ULTs Pros and Cons

**Advantages**
- Most system calls are blocking for processes
- All threads within a process will be implicitly blocked
- Waste of resource and decreased performance
- The kernel can only assign processors to processes.
- Two threads within the same process cannot run simultaneously on two processors
### KLT Pros and Cons

#### Advantages
- The kernel can schedule multiple threads of the same process on multiple processors.
- Blocking at thread level, not process level.
  - If a thread blocks, the CPU can be assigned to another thread in the same process.
- Even the kernel routines can be multithreaded.

#### Disadvantages
- Thread switching *always* involves the kernel.
- This means two mode switches per thread switch are required.
- KTLs switching is **slower** compared ULTs.
- Still faster than a full process switch.

### Solaris Threads
- Task 2 is equivalent to a pure ULT approach – Traditional Unix process structure.
- Tasks 1 and 3 map one or more ULT’s onto a fixed number of LWP’s, which in turn map onto KLT’s.
- Note how task 3 maps a single ULT to a single LWP bound to a CPU.
Role of Scheduler Activations

Abstraction

User-level Threads

P1 P2 ··· Pn

Virtual Multiprocessor

Invariant: There is one running scheduler activation (SA) for each processor assigned to the user process.

Implementation

Thread Library

SA SA ··· SA

Communication via Upcalls

- The kernel-level scheduler activation mechanism communicates with the user-level thread library by a set of upcalls

  - Add this processor (processor #)
  - Processor has been preempted (preempted activation #, machine state)
  - Scheduler activation has blocked (blocked activation #)
  - Scheduler activation has unblocked (unblocked activation #, machine state)

- The thread library must maintain the association between a thread’s identity and thread’s scheduler activation number.
Multithreaded Servers (1)

- A multithreaded server organized in a dispatcher/worker model.

![Diagram of multithreaded server model]

Servers and state

- Stateless servers: Never keep accurate information about the status of a client after having handled a request:
  - Don’t record whether a file has been opened (simply close it again after access)
  - Don’t promise to invalidate a client’s cache
  - Don’t keep track of your clients

- Consequences:
  - Clients and servers are completely independent
  - State inconsistencies due to client or server crashes are reduced
  - Possible loss of performance because, e.g., a server cannot anticipate client behavior (think of prefetching file blocks)
Servers and state

- Stateful servers: Keeps track of the status of its clients:
  - Record that a file has been opened, so that pre-fetching can be done
  - Knows which data a client has cached, and allows clients to keep local copies of shared data

- Observation: The performance of stateful servers can be extremely high, provided clients are allowed to keep local copies. As it turns out, reliability is not a major problem.

Remote Procedure Call

- Explicit message passing has been typically used in early distributed systems
  - The paradigm does not achieve access transparency
    - `send()` and `receive()` primitives do not conceal communication from the communicating entities

- Alternative method to message passing – Allow programs to call procedures located in other machines
  - Simple and elegant idea, but subtle problems may exist
Parameter Passing

- Call-by-Value – To the called procedure a value is just an initialized local variable
- Call-by-Reference – The reference parameter is a pointer to a variable, rather than the value of the variable
  - The called procedure modifies the variable in the calling procedure
- Call-by-Copy/Restore – The variable is copied onto the stack by the caller and copied back after the call, overwriting the caller's original value
  - Not often used in computer languages

Which Mechanism to Use in RPC?

RPC Information Flow
RPC COMPILEATION

Writing a Client and a Server

The steps in writing a client and a server in DCE RPC
**Binding a Client to a Server**

- **Client-to-server binding in DCE**

![Diagram](image)

**Overview of the Sockets Interface**

![Diagram](image)
**Group and Context**

- MPI assumes communication takes place with a group of processes – (groupID, procID)
  - Use in lieu of a transport-level address
- MPI ties the concepts of process group and communication context into a communicator

**MPI Abstractions**

- A *process group* is a high-level abstraction, visible to users
- A *context* is a system-defined object that uniquely identifies a communicator – Mechanism to isolate messages in distinct libraries and the user programs from one another
  - A message sent in one context can't be received in other contexts.
  - The communication context is a low-level abstraction, not visible to users
- A *communicator* is a data object that specializes the scope of a communication.
  - MPI_COMM_WORLD is an initial communicator, which is predefined and consists of all the processes running when program execution begins
Message-Queuing Model

- MQM, aka Message Oriented Middleware, provides support for persistent asynchronous communication
  - MOM offers intermediate-term message storage capacity, without requiring either the sender of receiver to be active

Sender Running

Receiver Running

Sender Passive

Receiver Passive

MOM Basic Semantics and Interface

- MOM supports **loosely-coupled in time** semantics
  - Senders are given only guarantees that message will eventually be inserted in the receiver’s queue
  - No guarantee about **if** or **when** will messages be read
    - Recipient-behavior dependent
  - Basic interface to a queue in a message-queuing system.

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put</td>
<td>Append a message to a specified queue</td>
</tr>
<tr>
<td>Get</td>
<td>Block until the specified queue is nonempty, and remove the first message</td>
</tr>
<tr>
<td>Poll</td>
<td>Check a specified queue for messages, and remove the first. Never block.</td>
</tr>
<tr>
<td>Notify</td>
<td>Install a handler to be called when a message is put into the specified queue.</td>
</tr>
</tbody>
</table>
MQM Brokers Functionalities

- Message brokers act as an application-level gateway
  - Their main purpose is to convert incoming messages to a format that can be understood by the recipient
  - Properly match end-of-record delimiters between database applications
  - More advanced broker are designed to handle conversations between two different database applications
    - Difficult task, as information and semantic “mapping” between two different applications may not always be possible
- Brokers are commonly used in Enterprise Application Integration for mediation
  - Publish/Subscribe is the typical communication model
  - Brokers “matches” messages to applications’ interests

QoS Specification and Services

- A flow specification model and services

<table>
<thead>
<tr>
<th>Characteristics of the Input</th>
<th>Service Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum data unit size (bytes)</td>
<td>Loss sensitivity (bytes)</td>
</tr>
<tr>
<td>Token bucket rate (bytes/sec)</td>
<td>Loss interval (µsec)</td>
</tr>
<tr>
<td>Token bucket size (bytes)</td>
<td>Burst loss sensitivity (data units)</td>
</tr>
<tr>
<td>Maximum transmission rate (bytes/sec)</td>
<td>Minimum delay noticed (µsec)</td>
</tr>
<tr>
<td></td>
<td>Maximum delay variation (µsec)</td>
</tr>
<tr>
<td></td>
<td>Quality of guarantee</td>
</tr>
</tbody>
</table>
Specifying QoS – Token Bucket

- The principle of a token bucket algorithm.