MidTerm Prep

Juan Besa
Main Topics

1. What is a distributed system
2. Architectures
3. Threads and Code Migration
4. Fundamentals of Communication
Main Topics

1. What is a distributed system
2. Architectures
3. Threads and Code Migration
4. Fundamentals of Communication
What is a Distributed System

1. Definition

2. Goals

3. Types
   1. System Types
   2. Hardware Types
What is a Distributed System

• A Distributed System (DS) is a collection of independent computers that appear to its users as a coherent system

• Normally implemented as a middleware between the operating systems and the applications

• Parallel systems are a special case
  • Homogeneous, designed to run a single program
Goals of DS

• Resource sharing
  • Share printers, memory, processors…

• Transparency

• Openness

• Scalability
Transparency

• Make DS look like a single computer

• Different types

• Degrees of transparency
  
  • Different levels, more transparent is more convenient, but there is a performance hit
Types of transparency

- Access
- Location
- Migration
- Relocation
- Replication
- Concurrency
- Failure
Openness

- Allow heterogeneous components
  - Different machines
- Adding/Removing components
- Interface Description Language (IDL)
  - Specifications for interprocess communication
Scalability

• Ability to grow

• Centralization reduces scalability

• Replication and caching improve it
  • Consistency issues

• 3 Different dimensions
  • Size, geographic, and administrative
Types of Systems

- Distributed Computing
  - Targets increase in application performance
  - Cluster —> Homogenous, local
  - Grid —> Heterogeneous, internet
- Distributed Information
  - Supports enterprise computing
  - Transactions
- Distributed Pervasive System
  - Small devices
Transactions

• Mainly used on databases

• Begin_Transaction
  • Everything until the end transaction is part of the transaction

• End_Transaction
  • Commits (makes permanent) the transaction if all went well

• Abort transactions
  • Rolls back everything if the transaction failed
Transaction Properties

- **Atomic**
  - All or nothing

- **Consistent**
  - Maintains system invariants

- **Isolated or serializable**
  - Concurrent transaction don’t affect each other
  - Results run as in some sequential way

- **Durable**
  - After commits results are permanent
Hardware Types

• Multiprocessor
  • A single system with multiple CPUs
  • Shared memory (threads!)
  • Scalable up to a point

• Multicomputer
  • A set of independent nodes (machines)
  • No shared memory, other communication types
    • Message Passing (MPI!)
  • Very scalable, just add more nodes
Main Topics

1. What is a distributed system

2. Architectures

3. Threads and Code Migration

4. Fundamentals of Communication
Architectures

• Software
  • Describes how the software works (components)

• System
  • Defines the real architecture once implemented
Architecture Styles

• Layered
  • Organized in layers, each layer knows the layer above and below

• Object
  • Each component is an object, they communicate using RPC

• Data
  • Processes communicate through a common data repository

• Event
  • Processes communicate through events
System Architectures

- Centralized
  - Client server —> think webpages
  - Layered: User —> Processing —> Data
  - Vertical

- Decentralized
  - Peer to Peer, no one is special
  - Horizontal
Main Topics

1. What is a distributed system
2. Architectures
3. Threads and Code Migration
4. Fundamentals of Communication
Threads and Code Migration

- Processes
- Threads
- Code Migration
Processes

• An instance of a program execution
• Managed by the OS
  • Memory
  • Protection —> each processes runs isolated
  • Context Switches and Interrupts
• Expensive to create
Threads

• Lightweight processes

• Are created and run inside of a process
  • No memory protections, threads see the same memory
  • Fast context switches

• Exploit parallelism without IPC (interprocess communication)

• See techniques in discussion on Friday
Code Migration

• Instead of passing the data pass the code
• Hard and costly

Advantages
• Reduce communication
• Improve load balancing
• Flexibility

• Very hard on heterogeneous systems
Models for Code Migration

- Models assume a code consists of
  - A code segment
  - A data segment
  - A execution segment (a state)
  - Weak/Strong mobility
Mobility Characteristics

• Sender/Client initiated
• Run as new process
• Or run as same process
• Or clone the process (needs strong mobility)
Migrating Resources

• Very difficult

• 3 kinds of binding
  • Identifier, Value, Type

• 3 types of resource-to-machine binding
  • Unattached, fastened, fixed
Main Topics

1. What is a distributed system
2. Architectures
3. Threads and Code Migration
4. Fundamentals of Communication
Communication

- Types
- Remote Procedure Calls (RPC)
- Messages
- Streams
Types of communication

- Persistent
  - Message stored until it is delivered
- Transient
  - Message is discarded if it can’t be delivered
- Synchronous
  - Sender blocks until it receives answer
- Asynchronous
  - Sender continues after submitting message
Remote Procedure Call (RPC)

- Implements a function call on a different machine
  - Process on machine A (client) calls process in machine B (server)
    - Process in A is suspended
    - Process in B executes, does all the work!
    - Process in A resumes with returned values
  - Client/Server stubs for transparency
Difficulties

• RPC seems like a function call but it is more difficult to pass resources

  • How do you pass parameters

    • Ints ok… what about arrays? References?

  • How do you return results?

• Dealing with heterogeneity

• Only synchronous communication
Messages

• Alternative to RPC, allows concurrency no need to suspend process in machine A

• MPI is a standard that allows message passing
  • Platform independent

• Allows synchronous and asynchronous communication
MPI

- Message Passing Interface
  - A standard
  - Available in Fortran, C, and C++
- Single-Program Multiple-Data model
  - One program, each node executes on different data
- Communication mode primitives
- Data distribution primitives
MPI Functions

• MPI defines functions that can be used to communicate between instances of the program
  • MPI_Init : start MPI program
    • Sets up infrastructure
  • MPI_Finalize: finalize MPI program
    • Cleans up infrastructure
MPI provides

- Point to point communication
- Collective communication
- Synchronization
- Synchronous and asynchronous communications
Point to Point

• MPI_Send(…) and MPI_Recieve(…) primitives

  • Blocking —> synchronous!

    • MPI_Send(…) only returns when the receiving function gets the message

    • MPI_Recieve(…) only returns when it gets the data

  • Watch out for deadlock

• There are also “Non-blocking” send and receives

  • MPI_Isend(…) and MPI_Irecieve(…)
Collective

- MPI_Broadcast(…)
  - From a root node communicate data to all the other nodes
  - Also blocking. Root until everybody receives, the others until they receive the data.
MPI needs only 6 functions

- The others are nice to have
  - Optimizations
  - Better abstractions

- 6 Functions are:
  1. MPI_Init
  2. MPI_Finalize
  3. MPI_Comm_Size (how many nodes are there?)
  4. MPI_Comm_Rank (what is my rank (id))
  5. MPI_Send
  6. MPI_Recv
MPICH

- Portability and good performance
- ADI a set of low level primitives
- Specifies functions for:
  - Specifying a message to be sent/recieve
  - Moving the data
  - Handling pending messages
  - Environment information
- Separates control from data
- How to handle buffering
Persistent Message Passing

- Uses a message queue
- No guarantee when message will be delivered, but it will be delivered
- Think email...(Outlook, not Gmail)
Streams

- A continuous flow of data —> Audio, video, …
- Not independent packets
- A sequence of data units, continuous or discrete
- May have to worry about timing
- Have QoS requirements, can’t wait forever!
Stream Transmission Modes

- Asynchronous
  - Just require in order transmission, no timing constraints
- Synchronous
  - Requires MAX end-to-end delay for each unit
- Isochronous
  - Requires MIN and MAX end-to-end delay
  - Have to transmit on time
- Can have multicast
  - One to many broadcast
QoS

• Required bit rate at which data should be transported
• Maximum delay until application is set up
• Maximum end-to-end delay
• Maximum delay variance (jitter)
  • Token bucket algorithm
• Maximum round-trip delay
Synchronization

- Streams may have dependencies
  - Audio and video
- Need to synchronize between discrete and continuous streams
  - Do at the data level
  - Or at a high-level interface
See you Monday!