Distributed Operating Systems

CS 239, Spring 1995

May 2, 1995

Outline

- Introductory material
- Distributed IPC
- Distributed File Systems
- Security for Distributed Systems

Introduction to Distributed Operating Systems

Distributed operating systems are different in many ways

Tasks in distributed OS require different handling

And the distributed nature makes new kinds of tasks possible

Outline of Introductory Materials

- Why distributed operating systems?
- Important issues in distributed OS's
- Important distributed OS tools and mechanisms

Why Bother?

- Economics of hardware
- Local autonomy
- Resource sharing
- Effective use of networks
- Reliability

Economics of Hardware

Cheaper to build many small machines than one large one

Due to -

Economies of scale

Chip design and fabrication issues

Gives purchasers easy options to increase compute power

Local Autonomy

Single user machines better suited for most computing tasks

Allow dedication of resources to a user's task

E.g., easier to guarantee response time

Owning user can control his computer power

Resource Sharing

But users need to share resources

Hardware resources -

Sharing printers & tape drives

Software resources -

Sharing data

Sharing access to software services

Network Usage

Users often want to communicate

Especially with other local users

System needs to support user interactions

Generally demands cooperation among multiple machines

Reliability

Failure of a single machine no longer halts everyone

Generally graceful degradation of the overall system's resources

Ability to apply fault tolerance for important tasks at a high architectural level

Problems With Distributed Systems

- More complex model of the system
- Harder to provide correct operation
- Harder to allocate resources properly
- Security
- Dealing with partial failures
- Scaling issues
- Heterogeneity

Complexity of the Model

Problem for - designers

- users

- system software

Harder to understand what will happen in any given case

Harder to design software to handle even understood complexities

Difficulties With Correct Operation

- Distribution requires more complex synchronization
- Differences between similar operations when remote and local
- New sources of non-uniform timings

Difficulties of Allocating Resources

- Local machine may have inadequate resources for a task
  - While a remote machine lies idle
- Infeasible to control resources centrally
  - Do I need to go remote to satisfy `malloc()`?
- Using remote resources conflicts with local autonomy

Security

- Security problems much trickier when no centralized control
- Data communications more subject to eavesdropping
- Physical security measures typically infeasible for many problems

Dealing With Partial Failures

- Single machines usually have easy failure modes
- Distributed systems face complications
- Even detecting failure of a remote machine is non-trivial
  - E.g., what's the difference between a slow network, a failed network and a crashed machine?

Scaling Issues

- Distributed systems control much larger pools of resources
- So algorithms that scale well become much more important
- Scaling puts severe limits on close cooperation

Heterogeneity Problems

- Most distributed systems must address problems of differing hardware and software
- Problems with data formats, executable formats
- Problems with software versioning
- Problems with different OS's

Resource Sharing

Resource sharing helps with some of the problems

Motivations for resource sharing -

- Information exchange
- Load distribution
- Computational parallelism

The fundamental distributed system problem

Distribution Complicates Everything

- Process control and synchronization
- Interprocess communications
- File systems
- Security
- Device management

Important Research Areas in Distributed Operating Systems

In the area of processes -

- Remote interprocess communications
- Synchronization
- Naming
- Distributed process management

More Research Areas

In the area of resource management -

- Resource allocation
- Distributed deadlock mechanisms
- Protection and security
- Managing communications resources

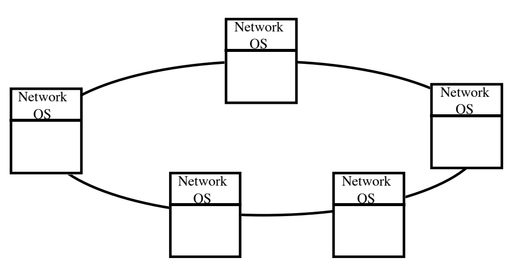
Network OS's vs. Distributed OS's

Network OS's control a single machine, plus some remote access facilities

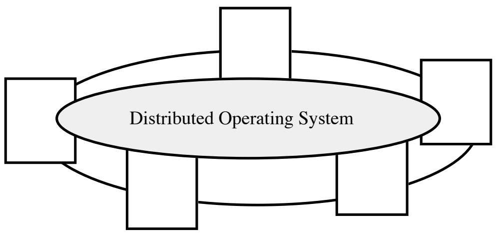
Distributed OS's control a collection of machines

Not a hard and fast distinction

Network OS Diagram



Distributed OS Diagram



Characteristics of Network OS’s

- Private per-machine OS
- Normal operations only on local machine
- Machine boundaries are explicit
- Little per-user fault tolerance

Characteristics of Distributed OS’s

- Single system controls multiple machines
- Use of remote machines invisible
- Users treat system as virtual uniprocessor
- Strong fault tolerance

Reality is Somewhere In Between

Relatively few true distributed OS’s

Network OS model a failure

But many modern systems have distributed OS-like capabilities

Like remote file access

And they also support network OS operations

Like rlogin and remote shell execution

The Role of the Network

Distributed OS’s made possible by networks

Two fundamental types -

Local area networks

Long haul networks

With very different characteristics

Local Area Networks

- High bandwidth
- Low delay
- Shared by modest number of machines
- Covers modest geographical area
- Dedicated to small group of users

Can be regarded as extension to computer’s backplane

Long Haul Networks

- Lower bandwidth
- Longer delays
- Shared by large numbers of machines
- Covers very wide area
- Typically shared by many independent groups

Fundamentally new kind of resource

Communications Protocols

Well defined methods of intermachine data exchange

To automatically handle problems of connecting network

Many different types required/available

Using Protocols in Distributed Operating Systems

Any intermachine operation requires a protocol to control it

So all machines involved can understand data exchange

Fundamental choice -

General vs. special purpose protocols

General vs. Special Purpose Protocols

General protocols try to handle any kind of traffic

Special purpose protocols are customized for one situation

General protocols simplify everything

Special purpose protocols may perform better

Important Issues in Distributed Operating Systems

- Communications model
- Process interaction
- Transparency
- Heterogeneity
- Autonomy
- Consistency and transactions

Communications Models for Distributed Operating Systems

How do machines communicate?

Generally message-based, at some level

ISO model adds too much overhead

So special-purpose protocols or simplified protocol stacking model is used

Process Interaction in Distributed Operating Systems

How do processes interact in a distributed system?

- Pipe model
- Uninterpreted message model
- Client/server model
- Integrated model
- RPC model
- Shared memory model

Pipe Model

Processes interact through pipes

Named or unnamed

Local or remote

Pros/Cons of Pipe Model

+ Simple transfer of large blocks of data
+ Hides many aspects of distribution
– Offers little organizational benefits
– Short on flexibility

Uninterpreted Message Model

Processes send explicit messages

System provides general message delivery service

Higher level semantics handled by processes

Libraries can provide useful message services

E.g., Isis

Pros/Cons of Uninterpreted Message Model

+ Simple and powerful
+ Relatively easy to implement
+ Can scale well
– Offers little organizational support
– Encourages asynchrony
– Not everyone's favorite programming paradigm

Client/Server Process Interaction Model

Processes are either clients or servers

Clients send request messages to servers

Servers send response messages to clients

Clients compete for server resources

Control of total system effectively distributed among servers

Name servers, IPC servers, file servers, etc.

Pros/Cons of Client/Server Model

+ Simple model
+ Hides much distribution
– Control of resources centralized in server
– Servers are bottlenecks
– Multiple implementations of servers to overcome bottlenecks increase complexity

Integrated Process Interaction Model

All system resources implemented in integrated way

Remote/local resources treated identically

System makes decisions on resource allocations

E.g., Locus

Pros/Cons of Integrated Process Interaction Model

+ Hides distributed complexity
+ Reduces bottlenecks
– Hard to implement correctly
– Performance problems likely
– Big scaling problems

RPC Model

Processes communicate through RPC

- Client/server often built on top of this
- But this model makes lower level more explicit

Pros/Cons of RPC Model

+ Simple programming model
+ Good scaling potential
+ Potentially good performance
– Potential for deadlock and blocking
– Implicit close connection between processes

Shared Memory Model

Provide distributed shared memory as the basic interprocess communications mechanism

Emulating local shared memory as closely as possible

Possibly without substantial hardware support

Pros/Cons of Shared Memory Model

+ Simple user model
+ Easy to build other mechanisms on top
– Hard to provide complete transparency
– Hard to provide good performance
– Serious scaling, heterogeneity questions

Transparency

Hiding machine boundaries
  From both users and system itself
Transparent systems much easier to work with
Providing at a low level has strong benefits
Not everything <u>should</u> be transparent

Kinds of Transparency

- Process access transparency
- Location transparency
- Name transparency
- Control transparency
- Data transparency
- Execution transparency
- Performance transparency

Benefits of Transparency

- Easier software development
- Support for incremental changes
- Potentially better reliability
- Simpler user model

Where Can You Provide Transparency?

- In applications (especially databases)
- In programming languages
- In operating system itself

When Don't You Want Transparency?

- When it's too complex to provide
  E.g., in heterogeneous systems
- When you want particular resources
  E.g., when you specify the local `/tmp`
- When remote performance is terrible
  E.g., over very slow links

<u>Always</u> must be able to bypass transparency

Heterogeneity

How transparent should heterogeneous networks be?

And at what cost?

Generally, how does the network deal with heterogeneity?

Types of Heterogeneity

- Computer heterogeneity
- Network heterogeneity
- Operating system heterogeneity
- Programming language heterogeneity

Solutions to Heterogeneity Problems

- Enforced coherence
  Happening at de facto level
- High level standards
  E.g., external data representations
- Bridges

Largely an unsolved problem