Distributed Systems
A Brief Overview

Dr. Pierre Vigneras
pierre@giki.edu.pk
Assistant Professor
Faculty of Computer Sciences,
GIKI, Topi,
District Swabi, NWFP,
Pakistan
Outline

• Usage of Distributed Systems
• Characteristics of Distributed Systems
• Distributed Object Oriented Applications
• Conclusion & Future Works
Usage of Distributed Systems
Usage of Distributed Systems

- **Parallelism**
  - Performance
  - Memory Distribution (volatile (RAM, cache, registers) or not (Disk, Tape, ...))

- **Fault Tolerance by replication**
  - NIS, DNS, NTP, ...

- **Load Balancing**
  - RoundRobin DNS
High Performance Computing
High Performance Computing

Manufacturer / Systems

Operating System / Systems

Others
Self-made
Dell
Hitachi
TMC
Intel
NEC
Fujitsu
Sun
SGI
Cray Inc.
HP
IBM

Others
BSD Based
Linux
Unix

12/11/2006
http://www.top500.org/

12/11/2006
http://www.top500.org/
High Performance Computing

Processor Family / Performance

12/11/2006
http://www.top500.org/
Trends: clusters

- **Benefit:** use mass-market standard hardware to reduce cost

- **Challenge:**
  - Programming Paradigm
    - Message Passing? (fits for clusters)
    - Shared Memory (Multithreading)? (fits for constellations)
  - Administration
    - System Installation, upgrades, etc...
    - System Globalization
    - Fault Tolerancy
Grid Computing

- Network of geographically distributed parallel machines (clusters, constellation, mpp, ...)
  - Transparency for the end-user (similar to the power grid)
- Challenges:
  - Humans! (software installation, security, ...)
  - Development Paradigm (set of jobs)
  - Heterogeneity (load balancing)
  - Services (monitoring, naming, ...)
    - Specifications (Open Grid Services Infrastructure)
Peer To Peer

- Original Philosophy of IP
  - First application: news, e-mail and the DNS!
- ICQ, Napster, seti@home, etc.
  - exploiting the leaf of the Internet
- Peer to peer != piracy
  - Human-links (searching), CycleTraders (web monitoring)
- Giant Corporations are going into that business:
  - IBM (Decryption), Intel (Philantropic P2P), ...
Peer To Peer: Challenges

- Scalability: depends on the model
  - Centralized (Napster), Decentralized (Gnutella), Hybrid (Emule, FastTrack)
- Anonymous P2P (danger?)
  - Freenet
- Coherency of data
- Programming Paradigm
Web Site Development

• Distribution of software components
• Reliability, Availability, Fault-Tolerancy are more important than absolute performance
• Layered Architecture (multi-tiers)
Multi-Tier Architecture

- **Advantages**
  - Role Separation (Ease the software engineering)
  - Fairly good scalability

- **Cons**
  - High Learning Curve
  - Applications are heavy and not portable
Characteristics of Distributed Systems
A distributed system is a collection of independent computers that appears to its users as a single coherent system.
# Definition: 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 may be shared by several competitive users</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>
<tr>
<td>Persistence</td>
<td>Hide whether a (software) resource is in memory or on disk</td>
</tr>
</tbody>
</table>
Characteristics [1]

- Latency: various access time
- Memory: different models of memory access
- Partial Failure: various parts can fail
- Concurrency: multiple requests

Latency

- Usually the most considered but the least important!
  - Influences performance of applications
- Software solution: mapping communicating processes
  - Deployment at starting-time after static analysis
  - Dynamic load balancing at runtime by migration (processes, data or threads)
- Hardware solution: specific networks (BIP/Myrinet, SCI, etc.)
Memory Access

- **Transparent Programming Model**
  - Shared Memory System: management at the OS level
  - Middleware: framework forbids use of memory references (compile-time check or lack of pointers)

- **Non-Transparent Programming Model**
  - Explicit Message Passing (MPI, PVM, ...)
    - Development Complexity
  - Remote Procedure (Method) Call
    - Call semantic (passed-by-copy (deep or not), passed-by-reference)
Partial Failures

- Software or Hardware Component of the global system may fail
  - Essential characteristic of distributed systems
- Hard to get a global state
  - Central Management = weak link and bottleneck
  - Distributed Management = overhead and scalability
- Failure Handling should be part of the interface (NFS protocol as a bad example)
  - Reason why in Java-RMI, any remote method should declare 'throws RemoteException'
Partial Failure

- Causes and Consequences of a failure are often unknown
  - An RPC has failed (timeout at client side)
    - network: message has been sent and received
    - processor: the call has not been executed (no side effect)

- Solutions:
  - Atomic Actions (Transactions – all or nothing)
  - Replication
    - active: n calls on n servers (-overhead, -latency, +coherency)
    - passive: 1 call on 1 server, replication in background (+latence, +overhead, -coherency)
Concurrent

- A remote component is used by many different entities at the same time
  - Essential characteristic of distributed systems
- Remote component must be \textit{thread-safe}
  - Complexity of the development (deadlocks, debugging, ...)
- The concurrent aspect should appear at the interface level
Distributed Object Oriented Systems
Method Invocation: \texttt{r\textasciitilde o.m(args)}
- Abstracted as passing message 'm(args)' to object \texttt{o}
- Natural extension to the remote case

Frameworks: Java/RMI, CORBA, .NET
- Only the remote method invocation

Latency, memory and partial failure is not really dealt with by those frameworks
Latency

- Object Mapping (static)
  - Analysis of the use graph
- Object Migration
  - Manual (Mobile Agent Model)
  - Automatic (dynamic analysis)
- Activity Migration
  - strong/weak, proactive/reactive
  - Virtual (execution flow), Real (threads)
Migration Challenge

- Data can be
  - Referenced remotely
  - Moved
  - Copied
- System should handle data coherency
  - Not easy!
Memory

- Java provides
  - Locally, a clearly specified and coherent memory model
  - Serialization
  - No pointers
  - Handling of Architecture Heterogeneity

- Good candidate for a remote implementation of the local reference concept
Partial Failure

- Should Appear in the Interface
  - Really? .Net says No! Highly debatable!
- Separate Business Logic Exception and Framework/Transport Exception
  - On a call per call basis
  - Globally using an event mechanism
Concurrency

- Remote objects should be *thread-safe*
  - Synchronisation impossible at the client side (multiple clients)
- Encapsulate the business object inside a sequentializer layer
  - Any remote call is sequentialized
  - Still many concurrent requests possible, but they are buffered and processed one after the other
Remote Method Invocation
Classes should be written specifically for the remote case
Lack of dynamism
  - Aspect Oriented Programming not possible
Multiple call semantic
  - By copy, by reference
  - Constraints (Serializable) not enforced by the type system at compile time
Proposition: Mandala

- OO based
- Extends the synchronous standard reference to
  - Asynchronous and possibly remote
- Focuses on Dynamism
  - Any class can be accessed remotely and asynchronously
- Provides Code Migration Naturally
- Based on strong foundations in $\pi$-calculus
- Open-source: http://mandala.sf.net
Conclusion & Future Works
Conclusion

• Distributed Systems many usage
• Four characteristics
  – Latency, Memory, Partial Failures, Concurrency
• OO in Distributed Systems
  – Good for Software Engineering
  – Java a good candidate
Conclusion

- RMI and CORBA provides solutions but have limitations
  - Latency, concurrency, Failures
  - Call Semantics
- Other solutions
  - JavaParty, ProActive, Do!, ...
- Our solution: Mandala (http://mandala.sf.net)
Future Works

- Concurrency
  - Degree
  - Refactoring (threads to AMI)

- Distribution
  - Network Protocols (Myrinet, BIP)
  - Distributed Garbage Collector

- Operating System
  - Build an OS based on new paradigms (AMI)