CSc 8320 Advanced Operating Systems

Distributed Process Implementation

by Marco Valero
Outline

Logical Model of Local and Remote Processes

Application Scenarios
  - Remote Service
  - Remote Execution
  - Process Migration

Mobile Agents
Stub Processes serve as a logical link making the physical boundary between the local and remote processes transparent.

Stub Processes at the server site have the capability of
- Interpreting the messages from the client stub.
- Invoking the respective operations.

This is similar to the RPC model.
Application Scenarios [1]

Depending on how the request messages are interpreted, there are three main application scenarios:

**Remote Service**
- The message is interpreted as a request for a known service at the remote site.

**Remote Execution**
The messages contain a program to be executed at the remote site.

**Process Migration**
- The messages represent a process being migrated to a remote site for continuing the execution.
Remote Services [1]

The primary application of remote services is RESOURCE SHARING in distributed systems.

A request message for a remote service can be generated at three different software levels.

- **Remote procedure calls** at the language level.

- **Remote commands** at the operating system level, like rcp in UNIX.

- **Interpretive messages** at the application level, like FTP or get and put methods.

Primary implementation issues: I/O redirection and security
Remote Execution [1]

Message is sent from the client to the server.

A client program needs to be executed at the server side.

Results in the spawning of a new process at the server side.

Helps in reducing the load on the client side.

**Difference between Remote Service and Remote Execution:**

Remote Service is a system with specific resources with resource sharing.

Remote Execution is a system with the purpose of load-sharing.
Remote Execution [1]

Implementation Issues in Remote Execution are:

- Load-Sharing Algorithm.
- Location Independence.
- System Heterogeneity.
- Protection and Security.
Remote Execution [1]

Implementation Issues in Remote Execution are:

- **Load-Sharing Algorithm.**
- Location Independence.
- System Heterogeneity.
- Protection and Security.
Remote Execution [1]

Load-Sharing Algorithm.

- Each process server can maintain a list of registered hosts that are willing to honor remote execution.
- The list of hosts participating are broadcasted.
- The selection procedure is by a centralized broker process.
- Once a remote host is selected-
  - The client process server indicates the resource requirements to the process server at the remote site.
  - If the client is authenticated and its resource requirements can be met, the server grants permission for remote execution.
  - The transfer of code image follows, and the server creates the remote process and the stub.
  - The client initializes the process forked at the remote site.
Remote Execution [1]

Load-Sharing Algorithm.
Location Independence.
System Heterogeneity.
Protection and Security.
Remote Execution [1]

Load-Sharing Algorithm.

**Location Independence.**

System Heterogeneity.

Protection and Security.
Remote Execution [1]

Location Independence.

- Process created by remote execution requires coordination to accomplish common task.
- So it is necessary to support logical views for the processes.
- Each remote process is represented by an agent process at the originating host.
- It appears as though the process is running on a single machine.
Remote Execution [1]

Load-Sharing Algorithm.
**Location Independence.**
System Heterogeneity.
Protection and Security.
Remote Execution [1]

Load-Sharing Algorithm.
Location Independence.
**System Heterogeneity.**
Protection and Security.
Remote Execution [1]

System Heterogeneity.

- If remote execution is invoked on the host, then it is necessary to recompile the program.
- Overhead Issue.
- Solution:
  - Use canonical machine-independent intermediate language for program execution.
Remote Execution [1]

Load-Sharing Algorithm.
Location Independence.
**System Heterogeneity.**
Protection and Security.
Remote Execution [1]

Load-Sharing Algorithm.
Location Independence.
System Heterogeneity.
**Protection and Security.**
Remote Execution [1]

Protection and Security.

- It is safer to accept only remote execution in the source or intermediate codes.
- Language used to describe a remote execution could be restricted to exclude potential problems.
- Run-time checking of parameters and stack overflows is necessary to protect the integrity.
- If intermediate code is used, then they must be verified that the code is a real source code.
Process Migration [2]

A remote execution is to be preempted and moved to another host.

Remote execution remains at the same remote host until completion of the execution.

The target process for migration is preempted and its state information must be transferred.

State information has two parts:
- Computation State.
- Communication State.
Process Migration [2]

Time

Freezing time

Source Site

Process P1

... ...

Execution suspended

Transfer of control

Destination Site

Execution Resumed

... ...

Process P1
Link Redirection [2]

Merits: easy implementation
Demerits: long delay time

Freezing time rescheduled
Migration decision resumed
Transfer of address space

Freezing time extended
Total time reduced
Migration decision resumed
Transfer of address space
Suspended migration decision
Freezing time reduced
Total time extended
Message Forwarding [2]

Three types of messages:

1. Received when the process execution is stopped on the source node and has not restarted on the destination node

2. Received on the source node after the execution started on destination node

3. Sent to the migrant process after it started execution on destination node
**Message Forwarding [2]**

**Resending messages**

Messages of type 1 and 2 are either dropped or negatively acknowledged, the sender is notified and it needs to locate the migrant process.

**Ask origin site**

Origin node keeps the info on the current location of the process created there, all messages are sent to origin which forwards them to migrant process.
Message Forwarding [2]

Link Traversal
Messages are queued and sent to destination node as part of migration procedure

Link is left on source node to redirect messages, link contains the system-wide unique id of a process and its last known location
Message Forwarding [2]

During the transfer the source node sends the notification (link update) of the transfer to all the nodes to which the process communicates:

- Messages are forwarded by the source node.
- Messages are sent directly to the destination node.
Mobile Agents [3,4]

Mobile agents are agents that can physically travel across a network, and perform tasks on machines that provide agent hosting capability. This allows processes to migrate from computer to computer, for processes to split into multiple instances that execute on different machines, and to return to their point of origin.

Unlike remote procedure calls, where a process invokes procedures of a remote host, process migration allows executable code to travel and interact with databases, file systems, information services and other agents.

Mobile agents can decide when and where to move next. When a mobile agent decides to move, it saves its own state and transports this saved state to next host and resume execution from the saved state.
Mobile Agents [3,4]

Common **applications** include:
- Resource availability, discovery, monitoring
- Information retrieval
- Network management
- Dynamic software deployment

**Conventional Approach**

**Mobile Agent Approach**
## Process Migration vs. Mobile Agents [2]

<table>
<thead>
<tr>
<th></th>
<th>Process Migration</th>
<th>Mobile Agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigational Autonomy</td>
<td>Migration decision is made by system.</td>
<td>Agents decide where and where to go</td>
</tr>
<tr>
<td>Code Execution</td>
<td>Programs are fully compiled and executed in native mode.</td>
<td>Most agents are coded in Java and are interpreted by their execution engine.</td>
</tr>
<tr>
<td>Strong/Weak Migration</td>
<td>Execution is resumed where it has been suspended.</td>
<td>Java-based agents resume their execution from the top of a given method.</td>
</tr>
<tr>
<td>I/O State</td>
<td>Long-term I/Os are forwarded to processes migrated to the destination.</td>
<td>Agents relinquish I/O connections every time they depart for their next destination.</td>
</tr>
</tbody>
</table>
Aglets is a Java mobile agent platform and library that eases the development of agent based applications. An aglet is a Java agent able to autonomously and spontaneously move from one host to another.

Originally developed at the IBM Tokio Research Laboratory, the Aglets technology is now hosted at sourceforge.net as open source project, where it is distributed under the IBM Public License. Aglets is completely made in Java, granting an high portability of both the agents and the platform.

Aglets includes both a complete Java mobile agent platform, with a stand-alone server called Tahiti, and a library that allows developer to build mobile agents and to embed the Aglets technology in their applications.
References.

[1] Distributed operating system and algorithms
    Randy chow, Theodore johnson

[2] Presentation on Parallel & Distributed Computing
    courses.washington.edu/css434/slides/Migration.ppt

    en.wikipedia.org/wiki/Mobile_agent

    http://www.davidreilly.com/topics/software_agents/mobile_agents/

[5] Aglets
    http://aglets.sourceforge.net/home.htm

    http://www.mobilec.org/applications.php