DACIA: A Mobile Component Framework for Building Adaptive Distributed Applications

Radu Litiu
University of Michigan, EECS
Design Goals

- Manage heterogeneity and adapt to variability
- Runtime reconfiguration of the application
- Support for application and user mobility
- Persistent connectivity between mobile components
- Location- and context-aware components
- Low overhead for both local and remote inter-component communication
An Adaptive Application

An application is a graph of connected components. Possible changes:

- Execute the computation on the client machine
- Store computed images instead of raw data
- Place data caches at various points in the network
- Add compress/decompress modules
DACIA Architecture

PROC - Processing and Routing Component
- Communication through ports
- Synchronous/asynchronous communication
- Message queue
- Mobile components
- Unique identifier

*Dynamic Adjustment of Component InterActions
DACIA Architecture (contd)

- **Engine**
  - Maintains the list of PROCs and their connections
  - Partial knowledge about PROCs running on other hosts
  - Migrates PROCs
  - Establishes and maintains connections between hosts
  - Communicates between hosts

- **Monitor** - monitors the application performance and makes reconfiguration decisions

- **Component mobility**
  - Transfer the PROC’s state, including the messages in the queue, and the state of its connections
  - Java serialization - efficient implementation
  - Message integrity
  - Locating a mobile component
Connectivity

- Multiple virtual connections between PROCs are multiplexed over the same physical network connection
- Hide temporary network failures
- Persistent connectivity between moving PROCs
- Low communication overhead
  - Local communication - procedure calls within the same address space
  - Asynchronous communication - cost of thread scheduling and queue management
  - Remote communication
    - batching
    - message forwarding
Dynamic Application Reconfiguration

- Change the connections between components
- Change the location of execution of various components
- Replicate components
- Dynamically load new components
- Replace a set of components with a different set of components

Mechanisms:
- Specialized monitors
  - Dynamic loading
  - Functionally equivalent configurations
- Command-line interface
Command-Line Interface

- `connect [hostname] [portnumber]` - connect the local engine to another engine
- `connectProcs [sourceProcID] [sourcePortNo] [destProcID] [destPortNo]` - connect two PROCs
- `disconnectProcs [sourceProcID] [sourcePortNo]` - disconnect two PROCs
- `exit/quit` - stop execution and exit
- `help` - print a help menu
- `move [procID] [hostname]` - move a PROC to the host indicated
- `print` - print information about the local and remote PROCs and the application configuration
- `start [procID]` - trigger an action on the PROC indicated
- `startMonitor` - start the monitoring service that performs runtime adaptation
- `update [hostname/all] <allProcs>` - updates the information about PROCs known by other engines
Micro-benchmarks - latencies (in microseconds) for inter-PROC communication and raw TCP

<table>
<thead>
<tr>
<th>message size (bytes)</th>
<th>local PROCs synchronous</th>
<th>local PROCs asynchronous</th>
<th>local procedure call</th>
<th>local TCP</th>
<th>remote PROCs</th>
<th>remote TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6.6</td>
<td>44</td>
<td>6.4</td>
<td>370</td>
<td>7800</td>
<td>770</td>
</tr>
<tr>
<td>1000</td>
<td>6.6</td>
<td>44</td>
<td>47.2</td>
<td>400</td>
<td>11000</td>
<td>2400</td>
</tr>
</tbody>
</table>

- Cost of PROC movement - 130 msec
- Macro-benchmarks - average time to serve a request