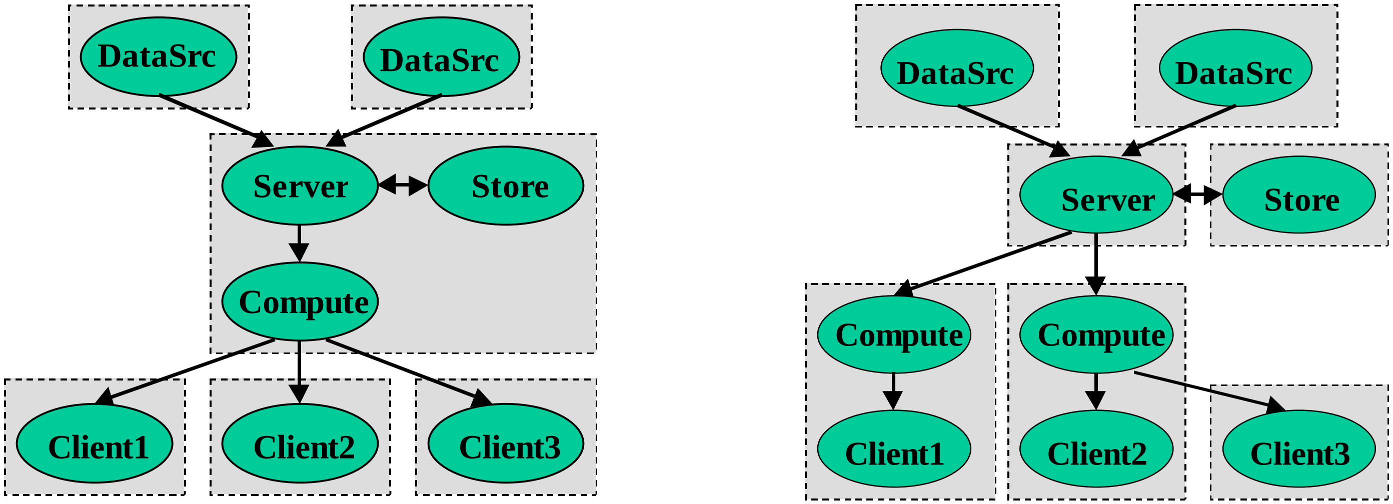
DACIA:
A Mobile Component Framework for Building Adaptive Distributed Applications

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

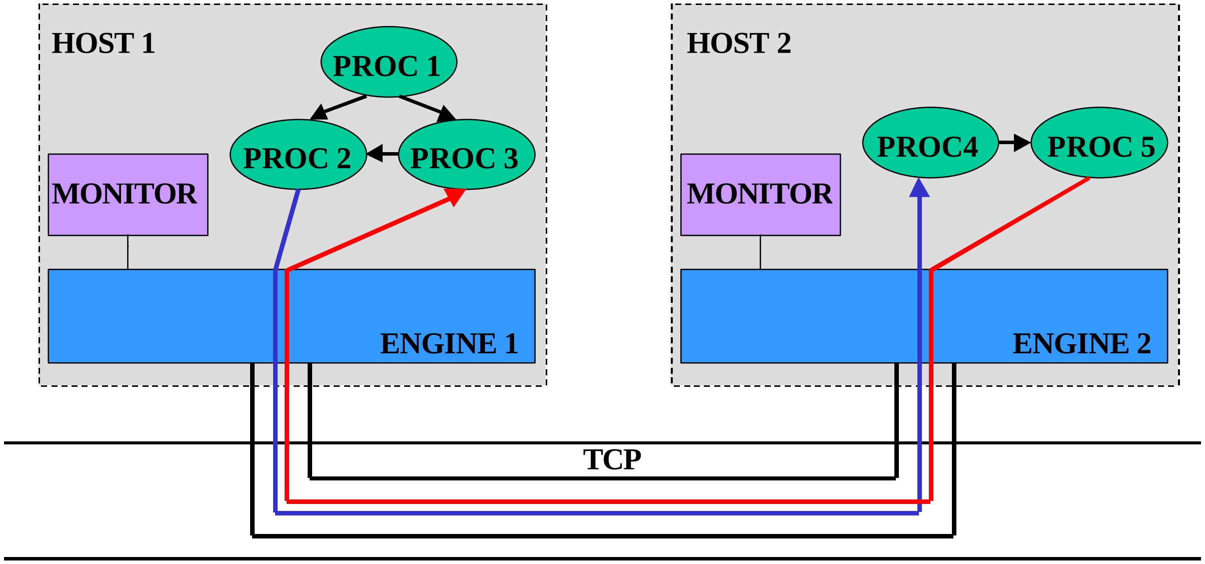


An application is a graph of connected components.

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

Performance (Java implementation)

- Micro-benchmarks - latencies (in microseconds) for inter-PROC communication and raw TCP

message size (bytes)	local PROCs synchronous	local PROCs asynchronous	local procedure call	local TCP	remote PROCs	remote TCP
0	6.6	44	6.4	370	7800	770
1000	6.6	44	47.2	400	11000	2400

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