peer-to-peer
and agent-based computing

Programming P2P Applications

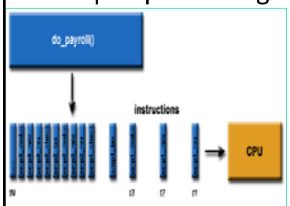
UNIVERSITY of ABERDEEN

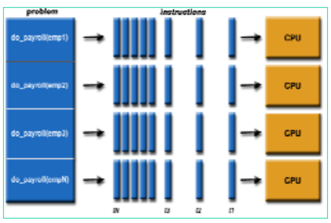
Plan of lecture

- Parallel programming
- Demonstrators and prototypes
- Message-passing vs. memory-sharing
- Tuple spaces as an alternative
- Threads with message-passing

Parallel Programming

- Traditional computing is **serial**
- However, some problems can be naturally broken down into smaller problems which can be tackled in parallel
- Example: processing a pay-roll





Parallel Programming (2)

- Multi-processor devices are very common
  - 2-core, quad-core, eight-core, ...
  - Applications do not need to “take turns” when run
  - Real parallelism!
  - Parallel computing is now very important
- Different kinds of parallel programs:
  - Processes **share variables** (concurrent programming)
  - Process **send messages** (distributed programming)

  To share info/data, coordinate, etc.

Why should we care about this?

- P2P applications are different from conventional computing
  - Message-passing is the name of the game
  - Not just HTTP requests (one-off messages)
  - Starting requests, forwarding messages, etc.
- However, if we want to “play” with P2P apps, then we might need to **simulate** things
  - For instance, start up 5,000 peers to see how it goes...
  - Message-passing can be an issue (more on this later)
- Let’s look at features of P2P applications and options to help us implement these

Important features of P2P applications

From an **individual peer**’s perspective:

- Loosely-coupled message-passing
  - No “blocking” command to read message
- Messages may not arrive in the order you want or need

From the **global** (**system**) perspective:

- High number of peers (50 and beyond)
- Very high volume of messages

Questions:

- How many machines can we use?
- Is our prototype to be deployed later?
- Is it for simulation only (to be re-developed later)?

Demonstrators and prototypes

- Suppose you have a very good idea for a P2P app
  - A protocol (order and kinds of messages)
  - An information model (routing tables, info in messages)
  - Some decision process for peers to follow
- What should you do?
  - We need to get to the core aspects quickly, to check if our ideas are any good
  - A demonstrator/prototype will serve this purpose
    - No GUIs, no sound effects, just core functionalities
  - Once we are satisfied that our ideas work, then an actual implementation could be developed
    - We might need to throw away initial prototype...
    - ... Hence it cannot be too sophisticated!

Demonstrators and prototypes (2)

- The main choices of technologies for a prototype:
  - Actual message-passing (distributed computing)
  - Simulation of message-passing (concurrent computing)
- Actual message-passing:
  - **Advantages**:
    - Realism (taste of what you will use for the final app)
    - Many machines can be used when testing
  - **Disadvantages**:
    - If you have only one machine, message-passing can be costly
    - Low-level protocols (HTTP, TCP) may delay, lose or corrupt messages
- Simulation of message-passing
  - Pretty much the reverse situation
  - NB: ideal when only one machine to be used

Message-passing

- Basic socket and ports
  - Programming language must offer this feature
- Decision: messages passed around as
  - Strings (which need to be parsed to be processed) OR
  - Serialised objects (no need to parse)
- Non-blocking receive
  - Programming language must offer this feature
- Single vs. many-threaded:
  - Each peer is a stand-alone program
  - All peers are threads in a single program
  - Peers are threads, but many copies of a program running
  - Peers themselves can be multi-threaded

Message-passing (2)

- Example: P2PTool
  - In-house educational tool (not for actual deployment)
  - Scale-up (more than 20 peers) is error-prone
  - Messages are serialised objects
  - Peers running in many machines can join network
- Further complication:
  - To provide a "global" view of the network, all messages are "intercepted" by the server
  - This puts a lot of strain on one process
  - Messages go missing and get corrupted

Simulating message-passing

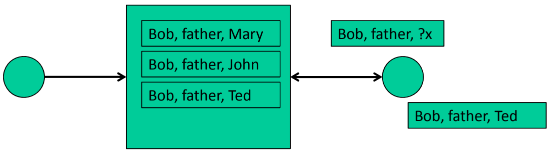
- Peers are threads running in one same program
  - Peers themselves could be multi-threaded...
  - Threads sharing working memory (RAM)
- Messages are managed via **a shared data structure**
  - An array, where all messages are stored
  - All peers/threads have access to shared data structures
- Challenge: discipline peers' access to messages
  - Being a shared variable, we must ensure **fairness**
  - For instance, each peer operates for *n* ms (round-robin)

Simulating message-passing (2)

- PeerSim (http://peersim.sourceforge.net/) simulates message-passing
  - Shared data structure
  - Multi-threaded (each peer is a thread?)
  - Peers take turns accessing this data structure
- Only one JVM (Java Virtual Machine)
  - Very efficient (complex scenarios quickly modelled)
  - Scalable (10,000 peers and beyond)
  - Various levels of control (useful in simulations)

An alternative: tuple spaces

- Tuple spaces
  - Processes write onto and read from shared tuple space
  - Tuple: pair, triple, quadruple,..., of pieces of information
  - A kind of “data-base” (centralised) where info is kept
- Associative memory:
  - Use of patterns to access tuples – anything which matches the pattern is returned



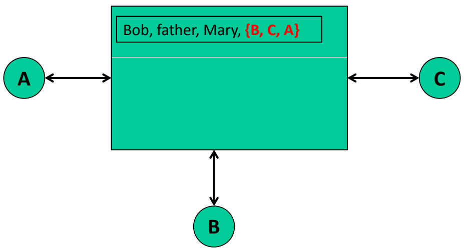
An alternative: tuple spaces (2)

- Programming languages offer extensions
  - JavaSpaces for Java
- Infrastructure to start up & manage tuple spaces
- Extensions to
  - Write onto tuple space
  - Read from tuple space
  - Remove tuples from space

  Remotely, with exceptions
- Interesting functionality:
  - Express interest on a tuple pattern and be warned when it arrives
  - Better than that: a method is invoked in your program, when a tuple arrives

An alternative: tuple spaces (3)

- Programming with tuple spaces is fun...
- Example: ensure 3 processes all read tuple
  - Assume you “own” all processes
  - Design info in tuple to help the solution



JADE

- Java Agent Development Environment
  - http://jade.tilab.com/
- Agents ≈ peers
  - Agents are lightweight Java threads
  - Messages sent/received within one JVM
- However,
  - It is possible to have JADE copies running in various devices (mobile phones!)
    - Many agents in each of them
  - Agent in a device can send message to agents in other devices
  - Agent can migrate (move) from one device to another
    - Mobile agents