Message Busses & Linda Tuple Space

Jonathan Walpole

Department of Computer Science & Engineering
OGI/OHSU

Message Busses

- Publisher-subscriber systems
  - message bus provides a level of indirection between publishers and subscribers
  - persistence allows disconnected operation
  - supports synchronization among processes
- Messages are sent (published) to and retrieved from mail boxes
  - message propagation can be eager or lazy
- Subject-based addressing
- Content-based addressing (content matching)

Message Bus Implementation

- Centralized server
  - easy to implement, but
  - bottleneck and single point of failure
- Replicated caches
  - local accesses
  - space overhead
  - consistency management overhead
- Distributed caches
  - location service?
  - Availability and performance trade-off

Linda

- Linda parallel programming language
  - tuple space
  - *out ( )*, *in ( )* and *rd ( )* primitives to add, remove and read tuples
  - *eval ( )* primitive to execute a tuple
  - associative content-based naming

- Linda communication kernel
  - implements persistent tuple space and tuple exchange

Linda Primitives

- Out (t)
  - adds tuple t to the tuple space
- In (s)
  - either, removes any tuple t that matches template s
  - and, assigns values of actuals in t to valiables in s
  - or, blocks
- Rd (s)
  - like in(s) but the tuple is not removed from the tuple space
- Eval (t)
  - like out (t), but t must be evaluated

Naming

- Content-addressible tuple space
  - based on partial matching (actuals must match)
  - like the select operation in relational databases
- Example

  out (“P”, 5, false)

  in (“P”, int i, bool b)

RPC in Linda?

- Client
  - out (procedure_name, me, invocation-parameters)
  - in (me, result-parameters)

- Linda using RPC?

The Programming Model

- Distributed data structures
  - initialized and cleaned up by coordination processes
  - manipulated by many “worker” processes in parallel
  - workers synchronize through the tuple space
  - naturally supports parallelism that would be difficult in client server architecture with RPC
- Example parallel matrix multiplication

The Initialization Process

- Puts matrix data and worker instructions in the TS

Out (“A”, 1, “row”, A’s-first-row)

Out (“A”, 2, “row”, A’s-second-row)

…

Out (“B”, 1, “col”, B’s-first-column)

…

Out (“dot”, 1, dim, “A”, “B”, “C”)

The Worker Processes

- Find out what work to do

```
in (“dot”, var NextElem, var dim, var mat1, var mat2, var prod);
if (NextElem < dim*dim)
    out (“Dot”, NextElem + 1, dim, mat1, mat2, prod);
I = (NextElem - 1)/dim + 1;
j = (NextElem - 1)%dim + 1;
```

- Do the work and publish the result

```
read (mat1, I, var row)
read (mat2, j, var col);
out (prod, I, j, DotProduct (row, col));
```

Cleanup Process

- Pull in the elements, construct the product matrix and print it

```
for (row = 1 to NumRows)
  for (col = 1 to NumCols)
    in (prod, row, col, var prod[row] [col]);
print prod
```

Whats nice about this?

- Independent of number of worker processes
- easily supports one worker per processor
- good for heterogeneous or unbalanced systems
- supports parallelism, asynchronous communication and overlap of computation and communication

- Is it message passing or shared memory?

How to Implement Tuple Space?

- Tuple space is like a message bus with mail boxes
- in and out are like receive and send of messages
- Both are like writes in a shared memory system
- Can be built using atomic broadcast
  - more efficient if underlying network supports it
  - still has high interrupt overhead since all processors get all messages
  - communication co-processors help

Implementation 1

- Fully replicated tuple space
- outs go to all nodes via broadcast
- ins go to all nodes via broadcast and use a deletion algorithm
  - phase 1: inform all nodes that t is gone, repeating until successful (delete is idempotent)
  - phase 2: t's origin node decides who succeeds and communicates its decision via a reliable point-to-point message
- rds are handled locally

Implementation 2

- For Ethernet with no support for ordering and reliability
- Protocol uses sequence numbers to order messages and synchronize rds with ins and outs
- storage optimizations
  - local outs
  - broadcast ins and rds
  - temporarily store templates