Distributed Systems
Principles and Paradigms

Maarten van Steen

VU Amsterdam, Dept. Computer Science
Room R4.20, steen@cs.vu.nl

Chapter 13: Distributed Coordination-Based Systems

Version: December 2, 2009

vrije Universiteit amsterdam

Contents

Chapter
01: Introduction
02: Architectures
03: Processes
04: Communication
05: Naming
06: Synchronization
07: Consistency & Replication
08: Fault Tolerance
09: Security
10: Distributed Object-Based Systems
11: Distributed File Systems
12: Distributed Web-Based Systems
13: Distributed Coordination-Based Systems

Coordination models

Essence

We are trying to separate computation from coordination; coordination deals with all aspects of communication between processes, as well as their cooperation.

Couplings

Make a distinction between

- Temporal coupling: Are cooperating/communicating processes alive at the same time?
- Referential coupling: Do cooperating/communicating processes know each other explicitly?

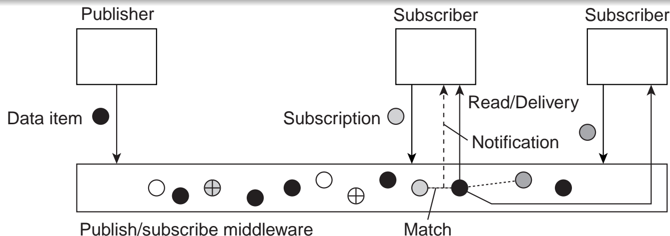
Coordination models

		Temporal	
		Coupled	**Decoupled**
Referential	**Coupled**	Direct	Mailbox
	Decoupled	Meeting oriented	Generative communication

Architectures: Overview

Essence

- A data item is described by means of attributes.
- When made available, it is said to be published.
- A process interested in reading an item, must provide a subscription: a description of the items it wants.
- Middleware must match published items and subscriptions.



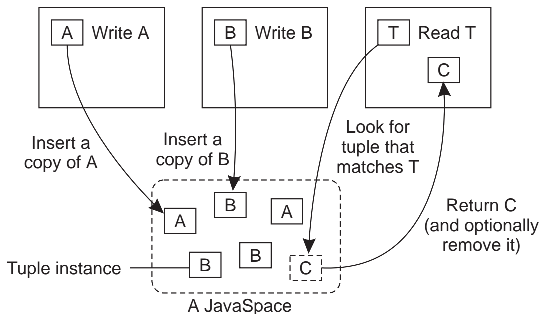
Example: Jini/Javaspaces

Coordination model

Temporal and referential uncoupling by means of JavaSpaces, a tuple-based storage system.

- A tuple is a typed set of references to objects
- Tuples are stored in serialized, that is, marshaled form into a JavaSpace
- To read a tuple, construct a template, with some fields left open
- Match a template against a tuple through a field-by-field comparison

Example: Jini/Javaspaces



Write: A copy of a tuple (tuple instance) is stored in a JavaSpace

Read: A template is compared to tuple instances; the first match returns a tuple instance

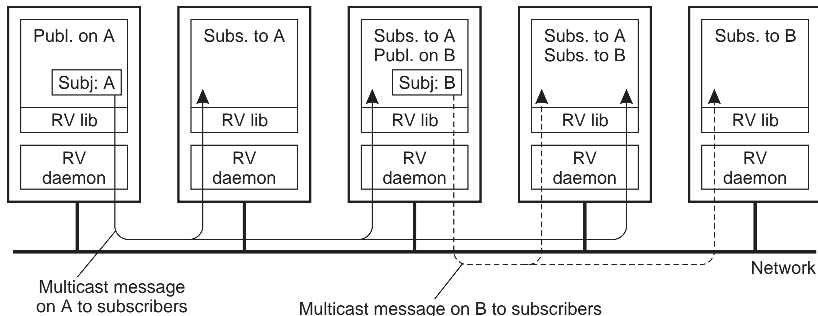
Take: A template is compared to tuple instances; the first match returns a tuple instance and removes the matching instance from the JavaSpace

Example: TIB/Rendezvous

Coordination model

Uses of subject-based addressing ⇒ publish-subscribe system.

- Receiving a message on subject X is possible only if the receiver had subscribed to X
- Publishing a message on subject X ⇒ message is sent to all (currently running) subscribers to X.

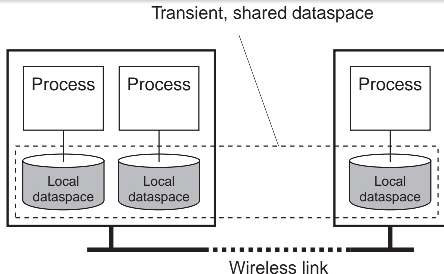


Example: Lime

Lime

Every node has its own dataspace:

- When P and Q are in each other's proximity, dataspaces become shared
- Published data items are stored locally, until removed
- P can publish data items from specific process
- Reactions describe what to do when a match is found



Content-based routing

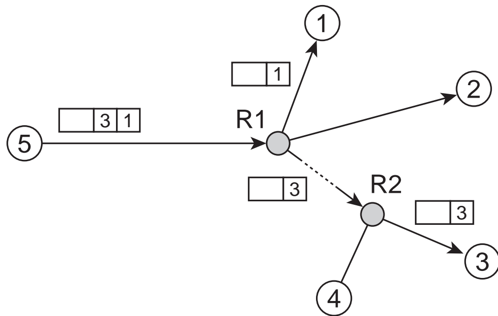
Observation

When a coordination-based system is built across a wide-area network, we need an efficient routing mechanism (centralized solutions won't do).

Solution

- Naive: Broadcast subscriptions to all nodes in the system and let servers prepend destination address when data item is published
- Refinement: Forward subscriptions to all routers and let them compute and install filters.

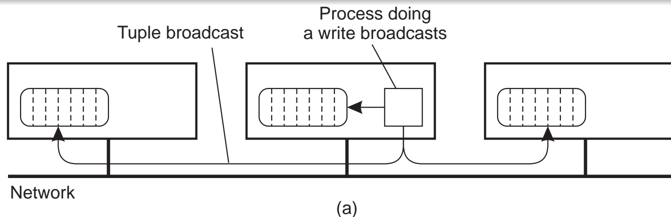
Content-based routing: naive solution



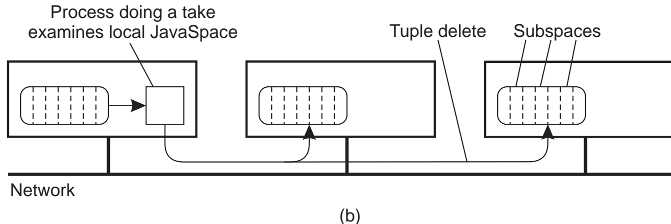
Replication: Static approaches

Note

Replicating data items to all machines implies broadcasting removals.



(a)



(b)

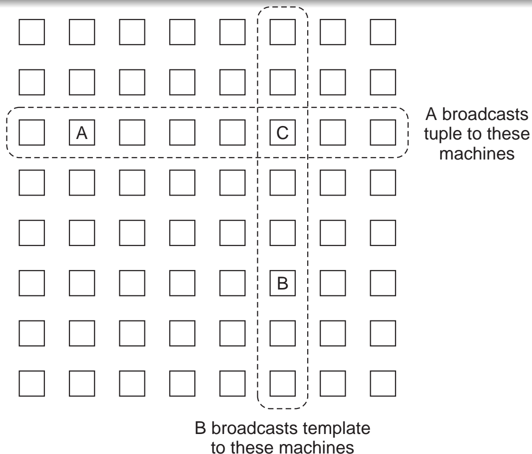
Balancing read/write operations

Problem

Find a balance between the costs for reads, and writes/removals ⇒ organize dataspace as 2D grid

Example

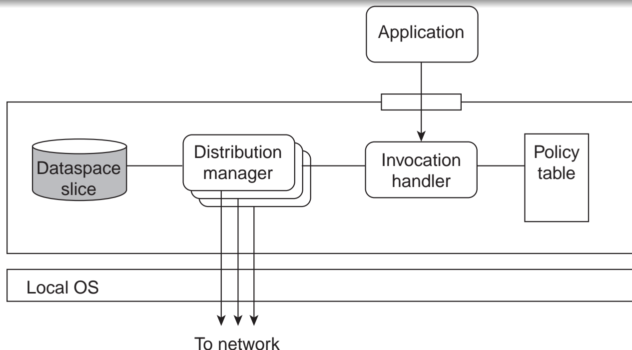
A writes a data item;
B wants to read it.



Dynamic replication

Observation: Not all data items are equal

- Decide on replication on a per-type basis
- Refinement: Let a central component observe read/write patterns and decide on replication strategy (self-replication)



Fault tolerance

Observation

In many cases, fault tolerance is achieved by using a primary-backup approach for a central dataspace server.

Refinement

Decide per data type the required availability, and replicate based on availability of nodes:

- MTTF: mean time to failure
- MTTR: mean time to repair
- Node availability:

$$\frac{MTTF}{MTTF + MTTR}$$

- Let nodes estimate MTTF and MTTR by logging the current time.

Security

Dilemma

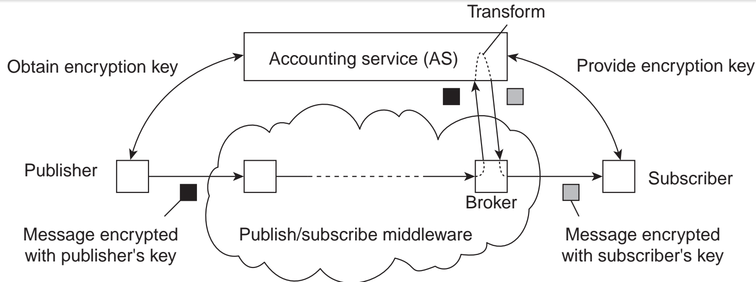
We wanted anonymity between processes, but security requires that we authenticate publishers and subscribers $\Rightarrow$ we need to trust the servers that establish the matching between the two.

- Information confidentiality: the middleware is not allowed to see what data is published. In practice, only restricted number of fields can be used.
- Subscription confidentiality: the middleware is not allowed to see what subscriptions look like. Solution: Match on encrypted data fields, although this alone will often reveal too much info on publishers and subscribers.
- Publication confidentiality: ensure that specific processes are not even allowed to see certain messages.

Secure decoupling

Solution

Let an accounting service manage keys, and re-encrypt a data item before it is forwarded to a subscriber ⇒ (1) routers work on encrypted data, (2) publisher and subscriber need not share a key.



Dilemma

Is security the show-stopper for publish/subscribe systems?