COMP90015 Distributed Systems

Indirect Communication

School of Computing and Informati

© The University of Melb

2022 Semester II

1. Message Queues

2. Group Communication

3. Publi

4. Distributed Shared Memory
   - Tuple Spaces

5. Comparison

Indirect Communication

- Whereas direct communication is communication that takes place directly between the communicating processes, *indirect communication* is defined as communication between entities in a distributed system *through an intermediary* with no direct coupling between the sender and the receiver(s).
  - *Space uncoupling* – sender does not know or need to know the identity of the receiver(s)
  - *Time*
    to exis
- Time u tion. Async n independent lifetime, in other words we could consider a time coupled asynchronous system.
- Indirect communication paradigms tend to be describ hor that aids in understanding the expectations of the parad inds of distributed applications it is useful:
  - Message Queues
  - Group Communication
  - Publish/Subscribe
  - Shared Memory
  - Tuple Spaces

Message Queues

- Message queues provide a point-to-point service usin *message* for data encapsulation and *queue* point-to-point in that each message is sent by a single proc *cer* – and received by a single process – *consumer*
- Since communication uses messages, the message queue paradigm may not be suitable for applications that require streaming data or bulk data transfer.
- Good for distributing units of work to processes and command/control type operations.

Programming model

Usually the message queue system is expected to provide reliability in that messages are not dropped or lost, and since its a queue, messages are received in the order sent. The API is very much the same as a blocking queue tha

- *send* – p der if the queu
- *blockin* then returns.
- *non-blocking receive* – or *poll*, a consumer will ch there, otherwise it returns without a message.
- *notify* – a signal is sent to the consumer when messages ar queue for consumption.

It is useful to consider this API in terms of actual processes and low-level exchange protocols. E.g. the implementation may use TCP for producers and consumers to connect to the queueing system.

Examples

- A message queueing system typically provides a library for the programming to build a client, either a producer or a consumer, and a server implementation that implements the queue manager itself. The server implementation will typically run a process that allows producers and consumers to connect.
- Modern examples include RabbitMQ and ZeroMQ.

Discussion questions

Questio
impleme

Group Communication

Group communication offers a space uncoupled service whereby a message is sent to a group and then this message is delivered to all members of the group. It provides more than a primitive IP multicast:

- manages group membership
- detects f

Typical as

- group cr
  - creat
  - list/search available groups
- group membership
  - join/leave a group
  - list members of a group

- multicast to selected members of a group, broadcast to all members

Efficient sending to multiple receivers, instead of multiple independent send operations, is an essential feature of group communication.

Group services

- closed groups only allow group members to multicast to it
- overlapping groups allows entities to be members of multiple groups
- synchronous and asynchronous variations can be considered

Implementation issues

- reliability and ordering in multicast
  - FIFO (first in first out) ordering is concerned with preserving the order from the persp
  - caus d in that order i
  - total o s then this is prese

- group membership management
  - group members leave and join
  - failed members
  - notifying members of group membership changes
  - changes to the group address

Discussion questions

Questio
used to imp er or worse
than using

Publish/Subscribe Systems

- *Publish/subscribe* systems are sometimes referred to as *distributed event-based systems*. A publish/subscribe system is a system where *publishers* (event sources) publish structured *events* to an *event service* and *subscribers* express interest in particular events through *subscriptions* which can be arbitrary patterns or query expressions over the structured events.
  - financial information systems
  - live fe
  - supp e informed of even
  - supp g from the ubiq
  - a broad set of monitoring applications, including network monitoring in the Internet
- Types of pub-sub systems include:
  - *Channel Based* – Publishers publish to named channels an events of a named channel.
  - *Type Based* – Subscribers register interest in types of event particular types of events occur.
  - *Topic Based* – Subscribers register interest in particular topics and notifications occur when any information related to the topic arrives.
  - *Content Based* – This is the most flexible of the schemes. Subscribers can specify interest is particular values or ranges of values for multiple attributes. Notifications are based on matching the attribute specification criteria.
- When and event matches a subscriber's subscription then the system sends a *notification* that contains the event to the subscriber.

Programming model

Advertise provides an additional mechanism for publishers to declare the nature of future events, i.e. the types of events of interest that may occur.

Multi-server architecture

The *Broker* exchanges or routes information from publishers to subscribers.

Overall System Architecture

Examples of pub/sub systems

A modern example of a pub/sub system is Apache Kafka. Others are shown below.

Discussion questions

Questio worse than
to implem
the messa

Shared memory approaches

Distributed shared memory is an abstraction for sharing data between computers that do not share physical memory. Processes access DSM by reads and updates to what appears to be ordinary memory within their address space.

Tuple Spaces

The tuple space is a more abstract form of shared memory, compared to DSM.

Example: The LighTS interface

Picco, Balzarotti, et. al., “LighTS: A Lightweight, Customizable Tuple Space Supporting Context-Aware Applications”

```
public interface ITupleSpace {
  String getName(); // name of tuple space
  void out(ITuple tuple); // put to tuple space
  void ou
  ITuple
  ITuple
  ITuple[
  ITuple
  ITuple
  ITuple[] rdg(ITuple template); // blocking read
  int count(ITuple template); // count tuples
}
public interface ITuple {
  ITuple add(IField field);
  ITuple set(IField field, int index);
  IField get(int index);
```

```
  ITuple insertAt(IField field, int index);
  ITuple removeAt(int index);
  IField[] getFields();



  boolean matches(IField field);
}
pu                                 IField {

                                   zable obj);
}
```

Example

```
ITupleSpace ts = new TupleSpace("SAC05");
IField f1 = new Field().setValue("Paolo");
IField f2 = new Field().setValue( new Integer (10));
ITuple t1 = new Tuple().add(f1).add(f2);
ts.out(t1);
```

Example York Linda Kernel

The implementation uses multiple Tuple Space Servers.

Discussion questions

Questio
impleme ueue,
group co

Summary

Discussion questions

Questio s,
which one ideo
conferen hen what
kind of paradigm/metaphor would be more suitable