inf5040

Group Communication

Siri Fagernes Simen Hagen

November 22, 2007

Outline

The Base for This Talk

This talk is mainly based on

- *Group Communication Specifications: a Comprehensive Study*, Gregory V. Chockler, Idit Keidar and Roman Vitenberg
- *Virtual synchrony*, article at Wikipedia

Group Communication System

- *Group communication* is a means for providing *multi-point* to *multi-point* communication in a network.
- *View-oriented group communication systems* provide *membership* and reliable multicast services.
- A *group* is a set of processes that use the GCS for communicating with each other.
- GCSs facilitate development of fault-tolerant distributed systems.
- The first, and best know GCS is the ISIS Project

Services

- *Membership service:* maintains a list of the currently active and connected processes of the group (*the view*).
- *Multicast service:* delivers messages to the group members.

Group Communication System

- Uses multicast for communication
- Sends messages to the group
  - Update the state
  - Change of shared data

Example Use of GCSs

- Replications/Backup
- Distributed OS
- Resource Allocations
- Load Balancing
- System management
- High availability services
- Video-on-demand
- Video conferencing
- Distance Learning
- Shared applications (e.g. Whiteboard)
- Fault tolerance in CORBA

Different GCSs

- Isis
- Transis
- Ensemble
- The configurable service
- Newtop
- xAMp
- Phoenix
- QuickSilver
- Totem
- Horus
- RMP
- Consul
- Relacs
- Spread
- Apia framework
- JGroups

Properties

There are different classes of properties:

- Safety properties of the membership service.
- Safety properties of the multicast service.
- Ordering and Reliability properties.

Safety Properties of the Membership Service

The membership services should

- maintain a list of the currently active and connected processes.
- report changes to the members by installing new views

Self Inclusion

A property requires that a process always is member of its view.

Self Inclusion

If process p installs view V, then p is a member of V.

Membership of a group reflects the ability to communicate with the processes in the group. A process can always communicate with itself.

Local Monotonicity

Require that view identifiers of the views that each process installs, are monotonically increasing.

Local Monotonicity

If a process p installs view V after installing view V' then the identifier of V is greater than that of V'.

Ensures that

- each view is installed only once.
- views are installed in the same order.

Local Monotonicity

Challenges

In case of a process crash

- It may receive old messages
  - Initial identifier is smaller then the last installed
  - Old messages may be floating in the system and be mistaken for a new one
- Fixes:
  - ISIS: assign a different identifier
    - Many ways to assign identifiers
  - Save the state to file before installing new views

Initial View Event

At startup, an initial view is typically installed. Thus,

Initial View Event

*Every **send**, **recv**, and **safe_prefix** event occurs within some view.*

The processes may have an initial view that is automatically installed, or they may negotiate at startup with other processes what the initial view should be.

Initial View Event

At startup, an initial view is typically installed. Thus,

Initial View Event

*Every **send**, **recv**, and **safe_prefix** event occurs within some view.*

The processes may have an initial view that is automatically installed, or they may negotiate at startup with other processes what the initial view should be.

Note: It is important that no **send** events is sent before the first **view_chng** event.

Safety Properties of the Multicast Service

- Give some guarantee of how messages are delivered.
  - This is dependent on the underlying QoS.
- Different orderings may be given by different systems.

Delivery Integrity

There should be no spontaneously generated messages by the GCS.

Delivery Integrity

*For every **recv** event there is a preceeding **send** event of the same message.*

No Duplications

Every message should be received only once by each process.

No Duplications

Two different events with the same content cannot occur at the same process.

No Duplications

Challenges

Not all system removes duplicates.

If the GCS provides the same QoS as the underlaying communication layer, duplicates are not removed, for example in RMP.

Sending View Delivery

The message should arrive in the same view as it was sent.

Sending View Delivery

If a process p receives message m in view V, and some process q (possible $p = q$) sends m in view V', then $V = V'$.

Not all GCSs give this guarantee.

Sending View Delivery

More on delivery

- It may be necessary to block sending of messages to fulfill this property.
- Supported by
  - ISIS
  - Totem
- Not supported by
  - Newtop
  - RMP
- Horus give the choice to the user

FIFO

The FIFO service guaranties that messages from the same sender arrives in the order in which they were sent.

FIFO

If a process p sends two messages, then these messages are received in the order in which they were sent at every process that receives both.

Causal Delivery

Causal order is an extension to FIFO order in that the response to a message m, have to be delivered after m.

Causal Delivery

If two messages m and m' are sent so that m causally precedes m', then every process that receives both these messages, receives m before m'.

Virtual Synchrony

The best know property of GCSs, and has its own model.

Virtual Synchrony

If process p and q install the same new view V in the same previous view V′, then any message received by p in V′ is also received by q in V′.

If two processes participate in two consecutive views, they have to deliver the same set of messages in the former.

Virtual Synchrony

- Perhaps the best known property of GCSs.
- Applications/processes can “see” shared/replicated data evolve in what seems to be a synchronous manner, although this might not be the real scenario.
- Key feature: a variable is replicated as soon as the update message reach the relevant group members, without waiting for all members acknowledging the update message.
- Main motivation: improved performance!

Virtual Synchrony: Ordering of messages

- All processes belonging to a group see the same events, in the same order.
- Multicast messages may be initiated concurrently by multiple senders, but will be delivered in some fixed order selected by the protocols implementing the model.
- Possible optimization: Application programmers can specify what kind of ordering of events are necessary.
- If “allowed”, VS might re-order the event messages if this is more optimal.

Applications

- VS replication is used mostly when applications are replicating information that evolves extremely rapidly.
  - Multi-user games.
  - Air traffic control systems.
  - Stock exchanges.
  - Telecommunication switches.
- VS has been standardized as a part of the CORBA reference model.
- Event notifications (publish-subscribe).

Performance vs Failures

- High level of performance, but weaker fault-tolerance properties.
- VS sometimes reorders events for performance, but this is not noticeable to applications.
- Problem: what if all the members that have the updated variable crash simultaneously?

Quicksilver

- Quicksilver Scalable Multicast (QSM).
- Project at Cornell University.
- Scalable, secure, reliable distributed applications.
- Ability to “regenerate” after failure.

`http://www.cs.cornell.edu/projects/quicksilver/`

The ISIS Project

- (One of) the first GCS projects
- A toolkit for distributed programming
  - Managing replicated data
  - Synchronizing distributed computations
  - Automating recovery
  - Dynamically reconfiguring a system to accommodate changing workloads

`http://www.cs.cornell.edu/Info/Projects/Isis/`

Project seems to be dead, as we were unable to find any “live” pages.

Spread Toolkit

- Fault resilient across local and wide area networks
- Services range from reliable messaging to fully ordered messages with delivery guarantees
- Libraries for
  - C/C++
  - Java
  - Perl
  - Python
  - Ruby
- Active at least until December 2006

`http://www.spread.org/`

Appia Framework

- Layered communication toolkit
- Implemented in Java
- Provide group communication, ordering guaranties and atomic broadcast
- Appia still seems to be active

`http://appia.di.fc.ul.pt/wiki/index.php?title=Main_Page`

The End

Questions?