Distributed Computing and Systems
Chalmers university of technology

The Ensemble system

Phuong Hoai Ha

Introduction to Lab. assignments
March 17th, 2006

Schedule

- The Ensemble system
  - Introduction
  - Architecture and Protocols
  - How does Ensemble achieve the group communication properties ?
- Programming with Ensemble in C
  - Framework

Ensemble history

- Three generations
  - ISIS
    - a fixed set of properties for applications
  - Horus
    - more flexible through modular architectures (layers)
  - Ensemble
    - adaptive protocols , performance, formal analysis.

The Ensemble System

- A library of protocols that support group communication.
- Ensemble Provides:
  - Reliable communication,
  - Group membership service,
  - Failure detector,
  - Secure communication.

Group membership service

- Endpoints
  - Abstraction for process' communication.
- Groups
  - Just a ***name*** for endpoints to use when communicating
    ⇒ do not change
  - Corresponding to a set of endpoints that coordinate to provide a service
- Views
  - A snapshot of the group membership at a specified point
    ⇒ may change from time to time
  - **Maintaining membership**

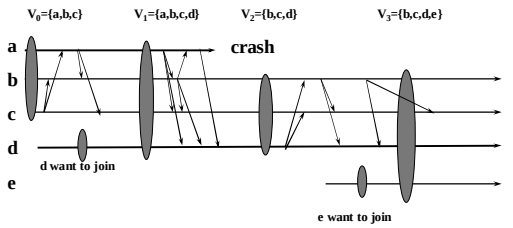
Reliable communication

- Multicast communication
  - Messages are delivered by all group member in the current view of the sender.
  - Based on IP-multicast
- Point-to-Point communication
- Properties:
  - Virtual synchrony
  - Stability
  - Ordering

Virtual synchrony

- Another name: View-synchronous group communication (previous talk)
- Properties:
  - Integrity
    - A process *delivers* a message at most once.
  - Validity
    - *Correct* processes always *deliver* the messages that they send.
  - Agreement
    - Correct processes deliver the same set of messages in any given view.

Virtual Synchrony

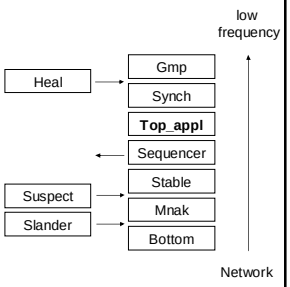


Schedule

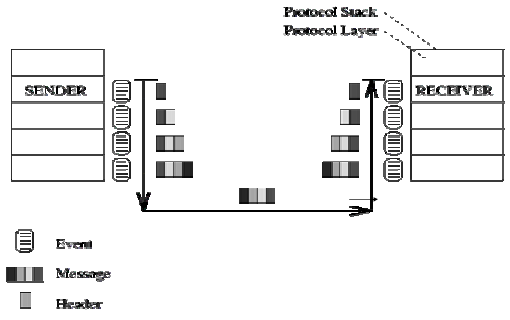
- The Ensemble system
  - Introduction
  - Architecture & Protocols
  - How does Ensemble achieve the group communication properties ?
- Programming with Ensemble in C
  - Framework

Infrastructure

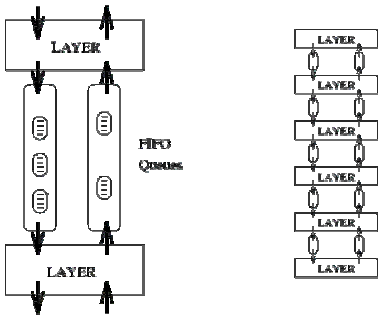
- Layered protocol architecture
  - All features are implemented as micro-protocols/layers
  - A stack/combination ~ a high-level protocol
- A new stack is created for a new configuration at each endpoint
- Ability to change the group protocol on the fly



Messages vs. events



Interface between layers



Layers

- Layers are implemented as a set of callbacks that handle events passed to them.
  - Each layer gives the system 2 callbacks to handle events from its adjacent layers
  - Layers use 2 callbacks of its adjacent layers for passing events.
- Each instance of a layer maintain a private *local state.*

Stacks

- Combinations of layers that work together to provide high-level protocols
- Stack creation:
  - A new protocol stack is created at each endpoint of a group whenever the configuration (e.g. the view) of the group changes.
  - *All* endpoint in the same partition receive the same *ViewState* record to create their stack:
    - select appropriate layers according to the *ViewState*
    - create a new local state for each layer
    - compose the protocol layers
    - connect to the network

Schedule

- The Ensemble system
  - Introduction
  - Architecture & Protocols
  - How does Ensemble achieve the group communication properties ?
- Programming with Ensemble in C
  - Framework

The basic stack

- Each group has a *leader* for the membership protocol.

Layers	Functionality
Gmp	Membership algorithm (7 layers)
Slander	Failure suspicion sharing
Synch	Block during membership change
Top_appl	Interface to the application
Sequencer	Total ordering
Suspect	Failure detector
Stable	Stability detection
Mnak	Reliable fifo
Bottom	Interface to the network

Failure detector

- Suspect layer:
  - Regularly ping other members to check for suspected failures
  - Protocol:
    - If (#unacknowledged Ping messages for a member > threshold) send a Suspect event down
- Slander layer:
  - Share suspicions between members of a partition
    - The leader is informed so that faulty members are removed, even if the leader does not detect the failures.
  - Protocol:
    - The protocol multicasts slander messages to other members whenever receiving a new Suspect event

Stability

- Stable layer:
  - Track the stability of multicast messages
  - Protocol:
    - Maintain Acks[N][N] by unreliable multicast:
      - Acks[s][t]: #(*s*' messages) that *t* has acknowledged
      - Stability vector
        StblVct = {(minimum of row s): ∀s}
      - *NumCast* vector
        NumCast = {(maximum of row s): ∀s}
    - Occasionally, recompute *StblVct and NumCast*, then send them down in a Stable event.

Reliable multicast

- Mnak layer:
  - Implement a reliable fifo-ordered multicast protocol
    - Messages from live members are delivered reliably
    - Messages from faulty members are retransmitted by live members
  - Protocol:
    - Keep a record of all multicast messages to retransmit on demand
    - Use Stable event from Stable layer:
      - *StblVct* vector is used for garbage collection
      - *NumCast* vector gives an indication to lost messages ⇒ recover them

Ordering property

- Sequencer layer:
  - Provide total ordering
  - Protocol:
    - Members buffer all messages received from below in a local buffer
    - The leader periodically multicasts an *ordering message*
    - Members deliver the buffered messages according to the leader's instructions
- See Causal layer for causal ordering

Maintaining membership (1)

- Handle Failure by splitting a group into several subgroups: 1 primary and many non-primary (partitionable)
- Protocol:
  - Each member keeps a list of suspected members via Suspect layer
  - A member shares its suspicions via Slander layer
  - View leader *l:*
    - collect all suspicions
    - reliably multicast a *fail*($p_{i0},\ldots,p_{ik}$) message
    - synchronize the view via Synch layer
    - Install a new view without $p_{i0},\ldots,p_{ik}$
  - A new leader is elected for the view without leader
    - If p_k in view V_1 suspects that all lower ranked members are faulty, it elects itself as leader and does like *l*.
    - A member that agrees with p_k, continues with p_k to the new view V_2 with p_k as the leader.
    - A member that disagrees with p_k, suspects p_k.

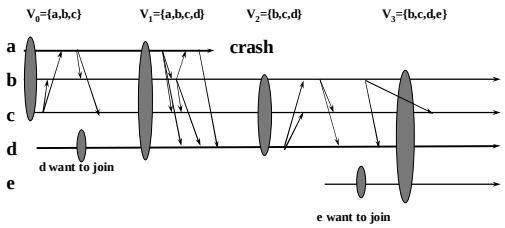
Maintaining membership (2)

- Recover failure by merging non-primary subgroups to the primary subgroup
- Protocol:

  l: local leader, *r*: remote leader

  1. *l* synchronizes its view
  2. *l* sends a merge request to r
  3. *r* synchronizes its view
  4. *r* installs a new view with its mergers and sends the view to *l*
  5. *l* installs the new view in its subgroup

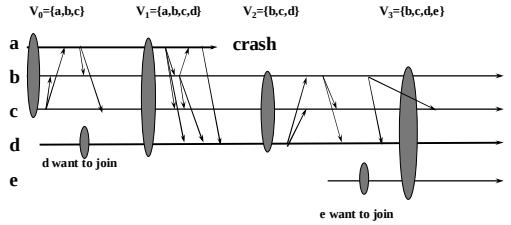
Join Group



Virtual synchrony

- Achieved by a simple leader-based protocol:
  - Idea:
    - Before a membership change from V_1 to V_2 all messages in V_1 must become stable
  - Protocol: before any membership change
    - The leader activates the Synch protocol ⇒ the set, M_{V1}, of messages needed to deliver in V_1 is bounded.
    - The leader waits until live members agree on M_{V1} via sending negative acknowledgements and recovering lost messages (i.e. *StblVct* = *NumCast*)

Virtual Synchrony



Schedule

- The Ensemble system
  - Introduction
  - Architecture & Protocols
- Programming with Ensemble in C
  - Framework
- Examples

Framework

```
typedef struct env_t {
    ce_appl_intf_t *intf;
    <your variables>
}
/*Define 7 callbacks*/
void install(){} //new view installed
void exit(){} //the member leaves
void receive_cast(){} //multicast msg
void receive_send(){} //p2p msg
void flow_block(){} //flow-control
void block(){} // view change
void heartbeat(){} //timeout

/*Create your own input socket*/
input() {
    …
    ce_flat_cast(intf, …, msg);
}
```

```
main( argc, argv){
    ce_appl_intf_t *intf;
    ce_jops_t jops; //endpoint
    env_t *env;

    /*Initialize Ensemble & process arg*/
    ce_Init( argc, argv);
    /*Create an interface/group*/
    intf = ce_create_flat_intf( env, 7 callbacks);
    env->intf = intf; //Keep the view
    /*Create an endpoint to join*/
    jops.hrtbt_rate=3;
    strcpy( jops.group_name, "demo");
    strcpy( jops.properties,
    CE_DEFAULT_PROPERTIES);
    jops.use_properties=1;
    ce_Join( &jops, intf);
    /*Add your own input socket*/
    ce_AddSockRecv(0, input, env);
    /*Pass control to Ensemble*/
    ce_Main_loop();
}
```


Environment variables

- Environment variable

```
setenv ENS_CONFIG_FILE <ensemble.conf>
```

- Makefile

```
CC        = gcc
CFLAGS =
ENSROOT =
    /users/mdstud/phuong/DSII/ensemble
LIB_DIR  = $(ENSROOT)/lib/sparc-solaris
INCLUDE = -I$(ENSROOT)/lib/sparc-solaris
.SUFFIXES: .c.o
LDFLAGS = -lsocket -lnsl -lm
CELIB     = $(LIB_DIR)/libce.a
demo:     demo.c
    $(CC) -o demo $(INCLUDE) $(CFLAGS)
    demo.c $(CELIB) $(LDFLAGS)
clean:
    $(RM) *.o
realclean:
    $(RM) demo
```

- File ensemble.conf

```
# The set of communication transports.
ENS_MODES=DEERING
#The port used for IP-multicast
ENS_DEERING_PORT=6793
# The user-id
ENS_ID= your_name
```

- Login to Sun machines
  persephone.tekno.chalmers.se
- Choose gcc-2.95
- All necessary material (e.g. good tutorial) is in /users/mdstud/phuong/DSII/demo/ensemble
- http://dsl.cs.technion.ac.il/projects/Ensemble/

Lab 1: Create a BBS system with Ensemble

- Using group communication in your program.
- One program.
- Peer group structure.
- C, Java

Lab 2: Construct a reliable and ordered broadcast

- Fixed number of machines.
- Broadcast
  - A message which is received by any machine should also be received by all other machines.
- Reliable
  - Integrity, Validity, Agreement
- Ordered
  - All machines should agree on the order of all received messages.

References

- M. Hayden & O. Rodeh, *Ensemble Tutorial*, Hebrew university, 2003
- M. Hayden & O. Rodeh, *Ensemble Reference Manual*, Hebrew university, 2003
- M. G. Hayden, *The Ensemble system*, PhD dissertation, Cornell university, 1998
- O. Rodeh, *The design and implementation of Lasis/E*, Master thesis, Hebrew university, 1997
- …