7 View-synchronous Group Communication

7.1 Introduction

This chapter starts from where Chapter 4 (Consensus and Reliable Broadcasts) left us, but it takes a different direction than explored in Chapters 5 and 6. We consider only crash failures here.

Consensus and reliable broadcast have been considered in *static* groups. Systems with *dynamic* groups extend this model by providing explicit *join* and *leave* operations to adapt the group membership over time. Moreover, such systems can exclude faulty servers automatically from the membership. Still, reaching agreement on the group membership in the presence of failures is not trivial.

Two approaches have been considered:

1. Run a consensus protocol among the all previous group members to agree on the future group membership. This is the canonical approach, tolerates further failures during the membership change, but involves the potentially expensive consensus primitive.

2. Integrate consensus with the membership protocol and run it only among the (hopefully) correct members. Since this consensus algorithm needs not tolerate failures, it can be simpler; but because further failures may still occur, it provides different guarantees.

The second approach is taken by *view-synchronous* group communication systems and related *group membership* algorithms [Pow96].

The first view-synchronous group communication systems was ISIS [BJ87]; many more followed and have been used in real-world applications like trading floor communication for the stock market or air-traffic control systems. IBM's Reliable Scalable Cluster Technology (RSCT) [IBM05] or *Spread* (`www.spread.org`) are other examples.

The system model is the same as in Chapter 4, including a failure detector $\mathcal{D}_i$ at every P_i.

7.2 Group membership

A *group membership* service receives *join*(S) and *leave*(S) requests with $S \subset \mathcal{P}$ and runs a failure detector to discover faulty servers. It outputs a sequence of group membership sets that are called *views*. Every view $V \subseteq \mathcal{P}$ is delivered through a $view\_change(vid, V)$ event, where $vid \in \mathbb{N}$ denotes a monotonically increasing *view identifier*. We say that the server (or process) *installs* the view V.

A membership service plays the dual role of a failure detector: it should detect the "stable" components of the system, i.e., the set of servers who can reliably communicate with each other.

Definition 7.1 (Membership service). A group membership service satisfies:

Self-inclusion: If P_i installs a new view V, then $P_i \in V$.

Monotonicity: If P_i installs a view V with identifier vid after installing a view V' with identifier vid', then $vid > vid'$.

Precision: For every stable component $S \subseteq \mathcal{P}$, there exists a view $V = S$ such that for all $P_i \in S$:

(i) P_i installs V as its last view; and

(ii) every message that P_i sends is received by every other server in V.

A membership service can be implemented using an eventually perfect failure detector and requires a timing assumption. In order to avoid problems with *monotonicity*, a server that crashes and recovers is usually given a new identity before it can rejoin the group.

7.3 View-synchronous broadcast

Again, one of the most important goals of a group communication system is to implement reliable (FIFO, causally ordered, or atomic) broadcast. Formally, reliable broadcast is characterized by two events *v-send*(m) and *v-deliver*(m) to send or receive a message m, respectively. It is defined with respect to the sequence of views delivered by the group membership service.

Definition 7.2 (View-synchronous reliable broadcast). A group view-synchronous reliable broadcast protocol satisfies:

Same-view-delivery: If a server P_i *v-sends* a message m in some view V and a server P_j *v-delivers* m in view V', then $V = V'$.

View-synchrony: If two servers P_i and P_j both install a new view V in the same previous view V', then any message *v-delivered* by P_i in V' was also *v-delivered* by P_j in V'.

Integrity: Every server delivers at most one message m, and only if m was previously broadcast by the associated sender.

The *view-synchrony* property implies that all servers who proceed together from one view V' through a view change to the next view V have *v-delivered* the same messages in V'. Therefore, they have the same state and no further synchronization is needed between them. Newly joining nodes, i.e., servers in V that were not also in V', need to receive the messages that they missed from a member of V'.

But *view-synchrony* says nothing about which messages were delivered at servers which did not proceed from the same view to the next. In order for a server to find out which others have the same state, additional information in a so-called *transitional set* is needed:

Transitional set: When a server P_i installs a view V in previous view V', then it also delivers a *transitional set* $T_i \subseteq V \cap V'$ such that any P_j that also installs V is contained in T_i if and only if P_j's previous view was also V'.

Algorithm 7.3 (View-synchronous reliable broadcast). A view-synchronous reliable broadcast protocol also delivers the views to the application. This implementation (like many practical ones) must be able to *block* the application during view changes so that it does not *v-send* any messages for some time. It relies on reliable point-to-point links with FIFO message delivery among all pairs of servers. Here is the code for P_i:

initialization:

$s \leftarrow 0$ // P_i's sequence number
$s_j \leftarrow 0$ $(\forall j \in [1, n])$ // sequence number of last *v-delivered* message from P_j
$vid \leftarrow 0$; $view \leftarrow \{P_i\}$ // current view
$new\_vid \leftarrow 0$; $new\_view \leftarrow \perp$ // next view while it is being installed

upon *v-send*(m):
send message $(\texttt{send}, vid, s, m)$ to all servers
$s \leftarrow s + 1$

upon *receiving a message* $(\texttt{send}, vid', s', m)$ from P_j with $vid' = vid$:
if $\big(new\_vid = 0\big)$ **or** $\big(new\_vid \neq 0$ **and** $P_j \in view \cap new\_view\big)$ **then**
v-deliver(m)
$s_j \leftarrow s_j + 1$

upon *view_change*(v, V):
send message $(\texttt{flush}, vid, i, [s_\ell]_{P_\ell \in view})$ to all servers in $view$
$new\_vid \leftarrow v$; $new\_view \leftarrow V$
block the application

upon *receiving* $(\texttt{flush}, vid, j, [s'_\ell]_{P_\ell \in view})$ messages from all $P_j \in new\_view$:
for each $P_\ell \in view$ **do**
compute the maximum t_ℓ of all received values s_ℓ
v-deliver all messages from P_ℓ that were sent with sequence numbers $s_\ell \leq t_\ell$; if some are missing, recover them from those members of new_view that have delivered them
output $view\_change(new\_vid, new\_view)$
$vid \leftarrow new\_vid$; $view \leftarrow new\_view$
$new\_vid \leftarrow 0$; $new\_view \leftarrow \perp$
unblock the application

If the group is stable, then the membership service will install the same view at all group members. Hence, all members who transition together to a new view compute the same *cut*, i.e., the set of maximal sequence numbers t_ℓ for $P_\ell \in view$. Therefore, they *v-deliver* the same set of messages in $view$ before installing new_view. Note that although applications relying virtually synchronous broadcast can be expressed asynchronously, the synchrony assumption is encapsulated in the membership service.

Chockler et al. [CKV01] survey the specifications of various group communication systems. The view-synchronous broadcast algorithm above is a simplified version of algorithm in [KSMD02, KK02].

References

[BJ87] K. P. Birman and T. A. Joseph, *Reliable communication in the presence of failures*, ACM Transactions on Computer Systems **5** (1987), no. 1, 47–76.

[CKV01] G. V. Chockler, I. Keidar, and R. Vitenberg, *Group communication specifications: A comprehensive study*, ACM Computing Surveys **33** (2001), no. 4, 427–469.

[IBM05] *IBM Reliable Scalable Cluster Technology: Administration guide*, 5th ed., April 2005, Available from `http://publib.boulder.ibm.com/clresctr/`.

[KK02] I. Keidar and R. Khazan, *A virtually synchronous group multicast algorithm for WANs: Formal approach*, SIAM Journal on Computing **32** (2002), no. 1, 78–130.

[KSMD02] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev, *Moshe: A group membership service for WANs*, ACM Transactions on Computer Systems **20** (2002), no. 3, 191–238.

[Pow96] D. Powell (Guest Ed.), *Group communication*, Communications of the ACM **39** (1996), no. 4, 50–97.