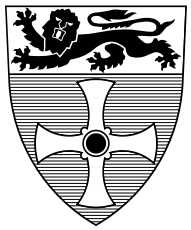
UNIVERSITY OF
NEWCASTLE



# COMPUTING SCIENCE

# Enhancing Replica Management Services to Cope with Group Failu

Paul D Ezhilchelvan and Santosh K Shrivastava

TECHNICAL REPORT SERIES
No. [ CS-TR-665]
October 1999

Contact: Paul Ezhilchelvan
[Paul.Ezhilchelvan@ncl.ac.uk]

A short version of this paper appeared in the Proceedings of the s
Symposium on Object Oriented Realtime Computing (ISORC99), St Malo,
April 99.

Copyright ©1999 University of Newcastle upon Tyne
Published by the University of Newcastle upon Tyne,
Department of Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, UK

# Enhancing Replica Management Services to Cope with Gro Failures

Paul D Ezhilchelvan and Santosh K Shrivastava

Department of Computing Science
University of Newcastle upon Tyne
Newcastle upon Tyne, NE1 7RU, UK

## Abstract

In a distributed system, replication of components, such as objects, is a well known way of achieving availability. For increased availability, crashed and disconnected components must be replaced by new components on available spare nodes. This replacement results in the membership of the replicated group 'walking' over a number of machines during system operation. In this context, we address the problem of reconfiguring a group after the group as an entity has failed. Such a failure is termed a group failure which, for example, can be the crash of every component in the group or the group being partitioned into minority islands. The solution assumes crash-proof storage, and eventual recovery of crashed nodes and healing of partitions. It guarantees that (i) the number of groups reconfigured after a group failure is never more than one, and (ii) the reconfigured group contains a majority of the components which were members of the group just before the group failure occurred, so that the loss of state information due to a group failure is minimal. Though the protocol is subject to blocking, it remains efficient in terms of communication rounds and use of stable store, during both normal operations and reconfiguration after a group failure.

Keywords — system availability, object groups, group failures, network partitions, membership views, membership services.

# 1. Introduction

In a distributed system, component replication (where a component computational entity such as a process, module) is a well known high availability. Equally well-known are the techniques for build using services such as membership and message ordering services. will consider the issue of enhancing the availability of a replica of failures, while preserving the strong consistency property wh states of all replicas that are regarded as available be mutually voting paradigm is an efficient way of achieving this end [Jajo network failures partition the replica group into disjoint subgr maintained only in the partition (if any) that contains a majority the master subgroup), with the replicas in all other partitions b That is, the majority partition (if any) forms the master sub-g services previously provided by the group; if a partition fur majority of the current master subgroup from the rest, then this becomes the new master subgroup. Thus, each newly formed maste contains a majority of the previously existed master sub-group. Re with dynamic voting offers a better way of maintaining system avai voting that requires a majority of all the members that initially remain connected. The following example illustrates dynamic voting:

Stage 0: Let the group configuration be initially $G_0$ = {C1, C2, C3 where Ci is the $i^{th}$ component.

Stage 1: Say, a network partition splits $G_0$ into $G_1$ = {C1, C2, C3, {C6, C7}; $G_1$ now becomes the master subgroup and thereby the new, configuration.

Stage 2: Say, $G_1$ splits into $G_2$ = {C1, C2, C3} and $G'_2$ = {C4, C5}; the master subgroup and thereby the third group configuration.

The above example indicates how the dynamic voting can preserve th group services even though the original group $G_0$ got split into isl having less than half the members of $G_0$. Availability can be howeve the master subgroup exists after a failure. Suppose that after sta $G_2$ detaches from other members. Now, no master subgroup exists a normal services can no longer be provided. We call this a group fa short). Note that many combinations of failures can lead to a g-fa g-failure after stage 2 can be caused by simultaneous crashing of crashing of C3 and detachment of C2 and C1, and so on. When t communication delays between components is not known with certain can occur even in the absence of any physical failure in the syst burst of network traffic, for instance, increases the communicatio connected components beyond what was considered to be likely, each falsely conclude that the other is not responding and hence must disconnected. Therefore, g-failures should not be regarded as rar on message delays cannot be estimated accurately.

Let us assume that the components have stable states which do not node crashes. Given that the component state survives node cra

preferable to have the replica management service enhanced to cop
instead of relying only on cold-start to resume the group services
achieve this, we propose a configuration protocol that enables the
master subgroup prior to a g-failure, to reconstruct the group on
of those members have recovered and got reconnected. Of course, t
ensure that only one such group is formed. The protocol objectiv
solely by the services used to build a replica group, in partic
service. To illustrate this, let us continue on the above example
below:

Stage 3: C3 crashes before it could record in its stable store
master subgroup $G_2$ ={C1, C2, C3} has been formed; the remaining me
C1 and C2, record in their stable store that $G_2$ is the latest mas
disconnect from each other.

No master-subgroup now exists and a g-failure has occurred. Next,
recovers and reconnects with C4 and C5, and C2 reconnects to C1.
C5} forms the 'master subgroup' on the basis that its members forr
last group configuration $G_1$ that is known to all of them, while {
the 'master subgroup' on the same basis that its members are a m
known configuration $G_2$. Now, we have two live master subgroups. T
from happening, we require that (i) a new master subgroup be consi
formed only after a majority of the previous master subgroup hav
stable store the composition of the new master subgroup (req1); a
subgroup constructed after a g-failure include at least a majority
latest master subgroup formed prior to the g-failure (req2). Requi
that there can be only one group configuration that qualifies to
subgroup formed before a g-failure (and, in general, at any given
example, a majority of $G_1$ did not record $G_2$ before the occurrence
stage 3; so, only $G_1$ is the latest master subgroup formed bef
Requirement req2 permits no more than one master subgroup to eme
failure.

We assume that the construction of the replica management system
voting) can avail the use of a group membership service which
operational component with an agreed set of components that are c
to be functioning and connected. For such a replica management syst
configuration management subsystem - the main contribution of t
provides (i) a group view installation service to enable membe
subgroup to record group membership information on stable store;
configuration service that makes use of these stable views to enal
after a g-failure as soon as enough number of the components of th
have recovered and reconnected. A prototype version of the
management service described here has been implemented [Black97]
replica management system called Somersault [Murray97]. Our ser
Somersault by providing recovery from group failures.

The paper is structured as follows: section two introduces the s
some definitions and notations; it also specifies the two servi
configuration management subsytem, namely the view installation
configuration services. The next two sections describe in detail h

provided. Section five compares and contrasts our work with the ap
the published papers in this area, and concludes the chapter.

# 2. System Overview and Requirements

## 2.1. Assumptions and System Structure

It is assumed that a component's host node can crash but conta
whose contents survive node crashes. Components communicate with
passing messages over a network which is subject to transient or l
We assume that a partition eventually heals and a crashed node ev
the bound on repair/recovery time is finite but not known wit
increased availability, we permit new components created on spare
group, with no restriction on the number of such joining nodes and
joining. Our system leaves to the administrator to decide how
available spare nodes should be instructed to join the group, and
spares instructed by the administrator are attempting to join th
enables them to join with a guarantee that they could compute t
component state from the existing members. For simplicity, we assu
of a group do not voluntarily leave the group, but are only for
crashes or partitions.

### *2.1.1. View Maker (VM) Subsystem*

We assume that our replica management system has been constructed
of the services provided by a group membership subsystem. This su
in the host node of an active component, say p, constructs membe
where a view is the set of components currently believed to be
connected to p. We call this subsytem the View Maker, or VM for s
the VM of p as $VM_p$. In delivering the uptodate views constructed,
to provide the abstraction of view synchrony or virtual synchrony
model is assumed for the underlying communication subsystem. We re
[Babaoglu95, Babaoglu97] and [Schiper94] for a complete list of
view synchronous and virtual synchronous abstractions, respecti
highlight some of these properties that are considered important f

vs1: p is present in any view constructed by $VM_p$. (self-inclusion.

vs2: a message m from another component q is delivered to p only
constructed by $VM_p$ prior to the delivery of m contains q. (view-me

vs3: the delivery of constructed views is synchronised with the d
such that components receive identical set of messages between c
that are identical. (view-message synchrony.)

vs4: If $VM_p$ delivers a view v, then for every component q in v, e
v or $VM_p$ constructs consecutive view w that excludes q. (view agre

There are many protocols in the literature which can be used t
assumed VM subsystem; e.g., [Birman87, Ricciardi91, Mishra9
asynchronous system with the primary-partition assumption, [Me
Moser96, Amir92, Ezhilchelvan95, Babaoglu95] for partitionable

systems. These protocols are not designed to cope with g-failure described below deals with g-failures using the services of the VM

### *2.1.2. Configuration Management (CM) Subsystem*

On top of the VM service exists a configuration management (CM) s Figure 2). CM of component p, denoted as $CM_p$, carefully reco information provided by $VM_p$ in the local stable store. In a t management system, a new view decided by $VM_p$ is usually delivered s our system, it reaches p via $CM_p$. $VM_p$ regards $CM_p$ as an applicati every new view it decides.

$CM_p$ of member p essentially provides the following three functiona

(i) it considers each view delivered by $VM_p$ and decides whether a g occurred. If g-failure occurrence is ruled out, $CM_p$ passes on that certain conditions are met which ensure strong consistency.

(ii) if a new view delivered to p contains a spare node attemptin $CM_p$ facilitates the spare node (in co-operation with CM of other new view) to compute the most recent component state.

(iii) if a view constructed by $VM_p$ indicates that a g-failure may executes a configuration protocol with CM of connected components. ensures that if the group is reconfigured, it is the master subgro that existed just before the g-failure was suspected to have occurr

It must be emphasized here that $CM_p$ can only suspect, not accurat occurrence of a g-failure when it inspects a new view from $VM_p$. consider the disconnected component C3 in figure 1. With the membership being {C1, C2, C3}, when CM of C3, $CM_3$, is delivered a s {C3}, it cannot know whether the partition has split the group in a g-failure (as in Fig. 1(a)), or in two ways (as in Fig. 1(b)) p form the next master sub-group. So, in both cases, $CM_3$ would suspe execute the configuration protocol. In case of 3-way partition, C3 g-failure master subgroup with, say, C1, if it re-connects to C1 executing the protocol. In the second case, when the partition hea it has been 'walked over': C1 and C2 have formed the new master sub C3 will then join the pool of spares. Note that it is also possib over in the first case: if the isolation of C3 lasts so long that the mean time and form the next master group. Thus, the out reconfiguration attempts by components is decided by the patter components recovery and reconnection.


Figure 1. (a) 3-way partitioning (b) 2-way partiti

## 2.2. View Names within the System

Our replica management system (above the communication layer) is s
layers as shown in fig. 2. Recall that $CM_p$ delivers to p a view c
only if certain conditions are met. That is, a view becomes more si
up within the system. To reflect this, we call a view differently
views constructed by $VM_p$ are called the membership views or si
$VM_p$ delivers Mviews to $CM_p$ via a queue called $ViewQ_p$ where Mviews
in the order of delivery. $CM_p$ deals with one Mview at a time, a
Mview reaches the head of $ViewQ_p$, which is denoted as $head_p$. CM
$head_p$ in the stable store as the new component view, provided a se
met. The component view of p is called $Cview_p$. Only $Cview_p$ is m
component p and provides p with the current membership view.
discussed earlier (see req1 of Section 1), making $head_p$ as Cview
stages; $head_p$ is first recorded in stable store as the stabilised
and then installed as $Cview_p$. CM uses a view numbering scheme fo
numbering the view contents of $Sview_p$ and $Cview_p$.

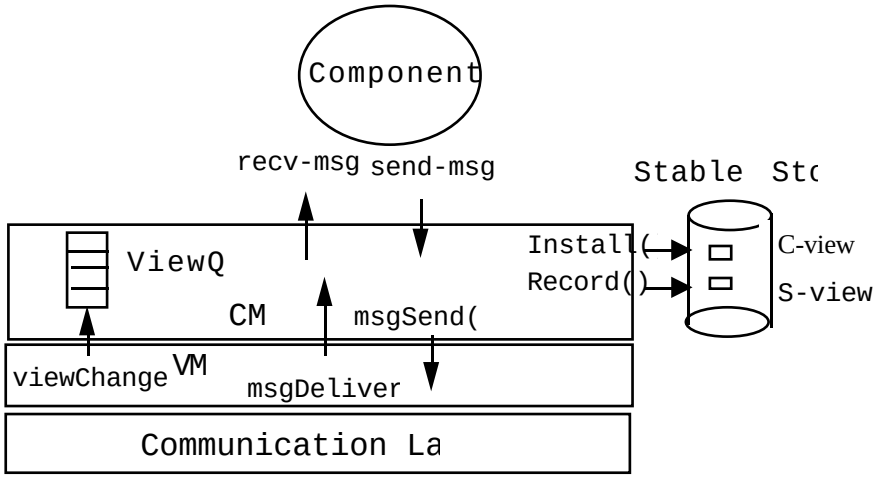



Figure 2. The system Architecture.

## 2.3. Notations and definitions

Each component p maintains three variables $status_p$ = (member, sp
(normal, reconfiguration, waiting, joining), and $view\text{-}number_p$ (an
its stable store. In addition, it also maintains two view variable
initialised to null set, if p is spare. $Sview_p$ has a view number
the view number of $Cview_p$ is indicated by $view\text{-}number_p$. $status_p$ i
when p considers itself to be a member of the group, or to spare
member p (with $status_p$ = member) observes a g-failure and subse
execute the configuration protocol, it sets its $mode_p$ to reconfig
changes to normal if p succeeds in becoming a member of the re
otherwise p becomes a spare setting $status_p$ = spare and $mode_p$
$mode_p$ of a spare component p can be either waiting or joining; th
is waiting to be informed by its $VM_p$ that it has been connected
group; once connected, p attempts to join the group by setting its
the join attempt by p succeeds, $status_p$ is set to member and mode

The variable view-number$_p$ is intialised to -1 at system start time formed) and whenever p becomes a spare; it is incremented every ti new Cview.

We define the terms survivors and joiners for a pair of Mviews co of a component p. Let $vu_i$, $vu_{i+1}$, ... $vu_j$, $j \geq i+1$, be a sequence c by $VM_p$ in that order. The set survivors($vu_i$, $vu_j$) is the set of survive from $vu_i$ into every Mview constructed upto $vu_j$: survivors $vu_{i+1} \cap ... \cap vu_j$. The term joiners($vu_i$, $vu_j$) will refer to the set which are not in survivors($vu_i$, $vu_j$): joiners($vu_i$, $vu_j$) = $vu_j$ - Finally, we define M_SETS(g) for a set g of components as the se subsets s of g: M_SETS(g) = {s | s $\subseteq$ g $\wedge$ |s| > (|g|)/2}.

## 2.4. View Maintenance

When viewQ$_p$ is non-empty, CM$_p$ of member p checks for the occurre failure by inspecting the contents of head$_p$, and by evaluatir survivors(Cview$_p$, head$_p$) $\in$ M_SETS(Cview$_p$). If this condition is n failure is assumed to have occurred. CM$_p$ first sends an Abort components in joiners(Cview$_p$, head$_p$), informing the CM of any attempt at recording/installing head$_p$. We will denote this Abort (which contains head$_p$) as AMsg$_p$(head$_p$). CM$_p$ then sets its vari reconfiguration and executes the configuration protocol to reconfig above condition is met, a copy of head$_p$ is atomically recorded in as the new Sview$_p$ with the view number = (view-number$_p$+1), provi conditions are satisfied. This Sview$_p$ represents the potential recording conditions are not met, the CM$_p$ either concludes tha occurred and proceeds to execute the configuration protocol se reconfiguration, or dequeues head$_p$ and proceeds to work with th any). The recording conditions, the need for them, and how they a discussed in the next section.

The newly recorded Sview$_p$ is regarded ready for becoming the nex installation condition is satisfied (again, the need for this c verified will be discussed in the next section). In which case, component view by replacing the current Cview$_p$ by Sview$_p$, and discards head$_p$. The local stable store update operations are in curly braces and are carried out atomically: {Cview$_p$:= Sview$_p$; v view-number$_p$ + 1;} If the installation condition is not met, a g-f have occurred and the configuration protocol is executed.

The view number of Cview$_p$ is indicated in view-number$_p$. Since Cvie are modified along with their view number as an atomic operation, t one Cview$_p$ and one Sview$_p$ associated with a given view number, pro numbers increase monotonically. Further, CM$_p$ installing the Svie Cview$_p$) can be interrupted only by its suspecting a g-failure; in failure is suspected, CM$_p$ will not record a new Sview$_p$ until th installed. Thus, in the absence of g-failure suspicions, either Sview$_p$ = the view number of Cview$_p$, or the view number of Sview$_p$ =

of $Cview_p+1$, the latter being true while the installation condit
upon to be satisfied. Let $Vu_p(k)$ be the Sview or the Cview that
view number k; similarly, let $Vu_p'(k')$ be an Sview or a Cview that
view number k', where p and p' may be the same component or distin
say $Vu_p'(k')$ is later than $Vu_p(k)$, denoted as $Vu_p'(k') \gg Vu_p(k)$, i

## 2.5. Requirements of the CM subsystem

We now state the two requirements the CM subsystem must meet. Th
concerned with the "normal service" period during which no g-failu
the second one is concerned with group formation after a g-failure.

Existence of at most one master subgroup at any time is achieved b
two components that install Cviews with identical view number,
views. Let $Cview_p(k)$ denote the Cview that p installs with view
The predicate $installed_p(k)$ is true if p has installed $Cview_p(k$
$!Cview_p(k)$ is true if $Cview_p(k)$ is unique, i.e., no component q ca
that is different from $Cview_p(k)$:

$$!Cview_p(k) \Rightarrow \forall\ q: \neg installed_q(k) \vee Cview_q(k) = Cview_p(k).$$

Note that any view installed by a component must contain the inst
So, if $Cview_p(k)$ is unique, then no component outside $Cview_p(k)$
with view number k; so, there can be only one kth membership se
hence only one kth master subgroup.

During normal service period, the CM modules of components ensure
installed are sequentially numbered, and that the kth Cview insta
provided that (k-1)th Cview installed is unique.

Formally, CM subsystem ensures:

Requirement 1:

$$\forall k > 0,\ installed_p(k) \Rightarrow \exists p': installed_p'(k-1);\ \text{and,}$$
$$\forall k > 0,\ installed_p(k) \wedge !Cview_p'(k-1) \Rightarrow !Cview_p(k).$$

Section 3 discusses how this requirement is met.

If we assume that Cview(0) is unique when the group is initially
above requirement is met, then there will exist a unique latest Cv
define this latest view as the last Cview, or simply the last.

Requirement 2: following a g-failure, a set $\Sigma$ of functioning and cc
with identical Cview, restart-view, should be formed as soon as
following properties:

Uniqueness: $\Sigma \cap last \in M\_SETS(last)$. If last is unique before g-fa
only one $\Sigma$ that can contain a majority of the last.

Continuity: $restart\text{-}view \neq last \Rightarrow$ view-number(restart-view) =view-
The sequentiality of CView numbering is preserved across g-failur
with g-failures is transformed into a view installation of di

nevertheless preserves the uniqueness and numbering of Cviews du service period. Section 4 discusses how requirement 2 is met.

# 3. Maintaining Unique Component Views

We describe the recording and installation conditions mentioned e how they help meet Requirement 1. We will first define a predicate becomes true when $CM_p$ of component p receives a message $m_q$ fro another component q, and becomes false if $CM_p$ believes that q ha disconnected before $m_q$ is sent. We later present a non-blocking al evaluate this predicate.

## 3.1. Recording Conditions for a member component

Let us assume (as induction hypothesis) that any two members have with identical view number. That is, for members p and q, $Cview_p$ $view\text{-}number_p = view\text{-}number_q = k$ (say). Let $head_p$, the Mview at $viewQ_p$, become non-empty for member p. $survivors(Cview_p$, he $joiners(Cview_p, head_p)$ follow different procedures for recordin consider the survivor or member p first and let survivors(Cv M_SETS($Cview_p$). As discussed in subsection 2.4, $CM_p$ can record a as $Sview_p$ only if recording conditions are satisfied. These re essentially ensure that all $joiners(Cview_p, head_p)$ have obtained well as replica states from $survivors(Cview_p, head_p)$ and made necessary, as a joiner component j has no replica state and o information. (It will only have $view\text{-}number_j = -1$, $Sview_j$ = Cvie waiting and $status_j$ = spare.) So, the recording conditions need t there are joiners in $head_p$, i.e., $joiners(Cview_p, head_p) \neq \{\ \}$.

Suppose that there are joiners in $head_p$. $CM_p$ multicasts a State component in $head_p$ (including itself). This message contains a $Cview_p$, $survivors(Cview_p, head_p)$, $view\text{-}number_p$, and p's state. We message of $CM_p$ as $SMsg_p(head_p)$. $CM_p$ then waits to see whether number of survivors in $head_p$ have sent their State messages, and computed and recorded the component state and also the view info stable store.

We will suppose that a joiner j in $head_p$ can compute the compone receiving State messages from some minimum number of distinct c $Cview_p$ which is the group membership when $head_p$ is being dealt assume that this number is proportional to the size of $Cview_p$ and $|Cview_p|$, denoted as $\Phi(Cview_p)$. (If it is a fixed one and not pro of $Cview_p$, then $\Phi(Cview_p)$ will be a constant function.) Since a $(|Cview_p|/2)$components need not survive into $head_p$ without caus $\Phi(Cview_p)$ cannot exceed $(|Cview_p|/2)+1$. So, $1 \leq \Phi(Cview_p) \leq (|Cvie$

Recording Condition 1 (rc1): It is to verify that at least $\Phi(Cview$ have sent their State messages. Formally,

$$|\{q \in survivors(Cview_p, head_p): recd_p(SMsg_q(head_p))\}| \geq \Phi(Cvi$$

Recording Condition 2 (rc2): It is to ensure that all joiners in and stored the component state and also recorded Sview which is t We will suppose that after $CM_j$ of joiner j has stored the com recorded an Mview, say vu, as $Sview_j$, it multicasts a Recorded component in vu. This message contains the recorded view vu and $RMsg_j(vu)$. So, the second condition is that $CM_p$ receive an $RMsg_j(h$ joiner j in $head_p$. Formally, $\forall j \in joiners(Cview_p, head_p)$: $recd_p(RM$

If rc1 and rc2 are met, $CM_p$ atomically records a copy of $head_p$ a with view number = $view\text{-}number_p+1$. It then multicasts an $RMsg_p($ components (including itself) in $head_p$. If rc1 is met but not r $head_p$ from $ViewQ_p$ but retains a copy to evaluate survivors($Cview_p$ next $head_p$. If rc1 is not met, $CM_p$ proceeds to execute the reconf after setting $mode_p$ to reconfiguration. Since no joiner can send R first receiving at least $\Phi(Cview_p)$ State messages, it is not poss without rc1.

### 3.2. Recording Condition for a joining component

The recording condition is verified by $CM_j$ of joiner j (with $mode_j$ as its $head_j$ - the first Mview in $ViewQ_j$ - is constructed by designed to become false if it is not possible for $CM_j$ to receive of State messages from members in $head_j$. The design is made somew the fact that when $VM_j$ delivers an Mview it cannot indicate who i members and who else (except j itself) are joiners. $VM_j$ can obtai only from VMs of member components. Recall that, as far as VM modu components are concerned, the local Cview is transparent and is r variable used by a local application called CM (see figure 2). Mo met but not rc2, $CM_p$ of member p dequeues $head_p$, and proceeds to next Mview in $ViewQ_p$; therefore, $VM_p$ cannot even assume that w Mview reaches the $head_p$, the Mview it delivered immediately befor have been installed as $Cview_p$. So, $CM_j$ cannot rely on $VM_j$ to indi members in $head_j$.

When $CM_j$ does not know $Cview_p$ of member p in $head_j$, its attempt t can result in a deadlock if $head_j$ contains more than one joiner. F member p in $head_j$ crashes before sending the State or the Abort m joiner will wait for ever to receive State messages from other jo essential that $CM_j$ first constructs a reference Cview which can be place of $Cview_p$ of member p in $head_j$ until an $SMsg_p(head_j)$ is rece will contain a copy of $Cview_p$. This reference Cview constructed $head_j$ is denoted as $RefCview_j(head_j)$ and is initially set to hea discussions are for a given $head_j$, we will refer to $RefCview_j($ $RefCview_j$.) $CM_j$ then sends a Join message to every component in he that it is a joiner. We denote this message as $JMsg_j(head_j)$. When $JMsg_j'(head_j)$, it removes the sender j' of that message from RefCv receives an $SMsg(head_j)$, it irreversibly sets $RefCview_j$ to the Cvi message. No $JMsg(head_j)$ that is received after receiving the first

$RefCview_j$. The survivors and view-number contained in the received noted in variables $members_j$ and $RefCviewNo_j$, respectively.

Once $RefCview_j$ is initialised to $head_j$, the recording condition st upon to become true or false. (Verifying the recording conndition i to modifying or irreversibly setting $RefCview_j$.) This condition is above for a member:

Recording Condition for joiner (rc_joiner): It verifies whether a distinct components sent their State messages. Formally,

$$|\{q \in RefCview_j: recd_j(SMsg_q(head_j))\}| \geq \Phi(RefCview_j).$$

If rc_joiner is met, $CM_j$ atomically records a copy of $head_p$ as i view number = $RefCviewNo_j$+1 and sets $mode_j$ = joining. It then $RMsg_j(head_j)$ to all components (including itself) in $head_j$. If rc_ an Abort message $AMsg(head_j)$ is received, $CM_j$ dequeues $head_j$ fro discards it.

Recall that $CM_p$ multicasts $AMsg_p(head_p)$ only if survivors(Cvie M_SETS($Cview_p$) when it starts to deal with $head_p$. So, it $SMsg_p(head_p)$ or $AMsg_p(head_p)$, not both, for a given $head_p$; hence receive an $AMsg_q(head_j)$ once rc_joiner is met. Otherwise, this wou sent $SMsg_p(head_j)$ without suspecting a g-failure at the start, whi has $head_q = head_j$ and survivors($Cview_q$, $head_q$) $\notin$ M_SETS($Cview_q$). T turn mean that $Cview_p$ and $Cview_q$ are not identical which is a induction hypothesis.

To illustrate how certain failure cases that could lead to deadlock the group {p,q,r} with unique $Cview_p$; i.e., $Cview_p = Cview_q = Cv$ Let the VM modules deliver an enhanced Mview such that $head_p$ = he {p,q,r,j,j1,j2,j3} = $head_j$, where j, j1, j2, and j3 are joiners. Sa multicasting their State messages; if {j, j1, j2, j3} remain co eventually changes to {p, q, r} from its initial value of head $recd_j(SMsg_c(head_j))$ will become false for crashed c = p, q, and r a that rc_joiner cannot be met.

### 3.3. Installation Conditions

Having recorded $head_p$ as $Sview_p$, $CM_p$ installs the $Sview_p$ as the n after verifying that a majority of the existing $Cview_p$ have recorde

Installation condition (ic): {q $\in$ survivors($Cview_p$, $head_p$): $recd_p$( M_SETS($Cview_p$).

The $CM_j$ of a joiner j has two installation conditions. The first o joiners of $head_j$ have recorded $head_j$; the second one is the same a for member p. Note that $CM_j$ has recorded $head_j$ means that it ha messages from some member p in $head_j$; so, $RefCview_j = Cview_p$ and survivors($Cview_p$, $head_p$).

Installation condition 1 for joiner (ic1_joiner):

$\forall c \in head_j$ - $members_j$: $recd_j(RMsg_c(head_j))$.

Installation condition 2 for joiner (ic2_joiner):

$\{p \in members_j: recd_j(RMsg_p(head_j))\} \in M\_SETS(RefCview_j)$.

If both conditions are met $CM_j$ makes component j a member by atomic
{$Cview_j := Sview_j$; $view\text{-}number_j := RefCviewNo_j + 1$; $status_j$ = mer
normal; }. The $head_j$ is then dequeued and discarded. If the first
no member p in $head_j$ would have recorded $head_j$; so, $CM_j$'s recor
undone by atomically executing: {$Sview_j$ = null; $mode_j$ = waiting;}.
dequeued and discarded. If only the second condition is not met,
to reconfiguration and executes the reconfiguration protocol. Obse
sets $mode_j$ to reconfiguration, $Cview_j$ and $view\text{-}number_j$ remain un
initial values which are null and -1 respectively.

### 3.4. Correctness and Liveness

Correctness: Suppose that $CM_p$ installs $head_p$ as the new $Cview_p$ w
number (k+1). The majority requirement in the installation conditi
a majority of $Cview_p(k)$ have recorded $head_p$ as their Sview with vi
The recording condition (rc2) ensures that every CM of joine
$Cview_p(k+1)$) has also recorded $head_p$ as its Sview with view-number
records a new Sview before the existing one is installed. Ther
$Cview_p(k)$ is unique, if $CM_q$ of a survivor or joiner q installs
$Cview_q(k+1) = Cview_p(k+1)$. This means that $Cview_p(k+1)$ is also uni

Liveness: $CM_p$ verifying the recording/installation condition requ
of the predicate $recd_p(m_q)$ which in turn involves checking whet
message $m_q$ has been/can be received from $CM_q$. Since the node of q
$m_q$ can be sent, the evaluation of $recd_p(m_q)$ must involve chec
continues to be present in the subsequent Mviews constructed by V
mind, we present an algorithm for evaluating $recd_p(m_q)$ which
indefinitely.

Figure 3 shows the ViewQ's of $CM_p$ and $CM_q$ which, for simplicity,
identical. We will also assume that $Cview_p = Cview_q$ = {p, q, r1,
$number_p = view\text{-}number_q$ = k (say). That is, $Cview_p(k)$ is unique. L
Mviews of $ViewQ_p$ and $ViewQ_q$ as: vu1 = {p, q, r1, r2, j}, vu2 =
vu3 = {p, q, j}. vu1 indicates the disconnection of member r3 (fr
inclusion of a new component j, vu2 the disconnection of r2
disconnection of r1.

We define $List_p(Mview)$ as the set of messages which $VM_p$ intend
between the delivery of Mview and the delivery of the immediate s
Mview. $List_p(vu3)$ is shown to be open and will remain so until a
vu3 is constructed. $List_p(vu1)$ and $List_p(vu2)$, on the other hand,
to indicate that no received message can enter these lists any l
message synchrony property of the VM subsystem (see §2.1.1),
$List_q(vu1)$, and $List_p(vu2) = List_q(vu2)$.

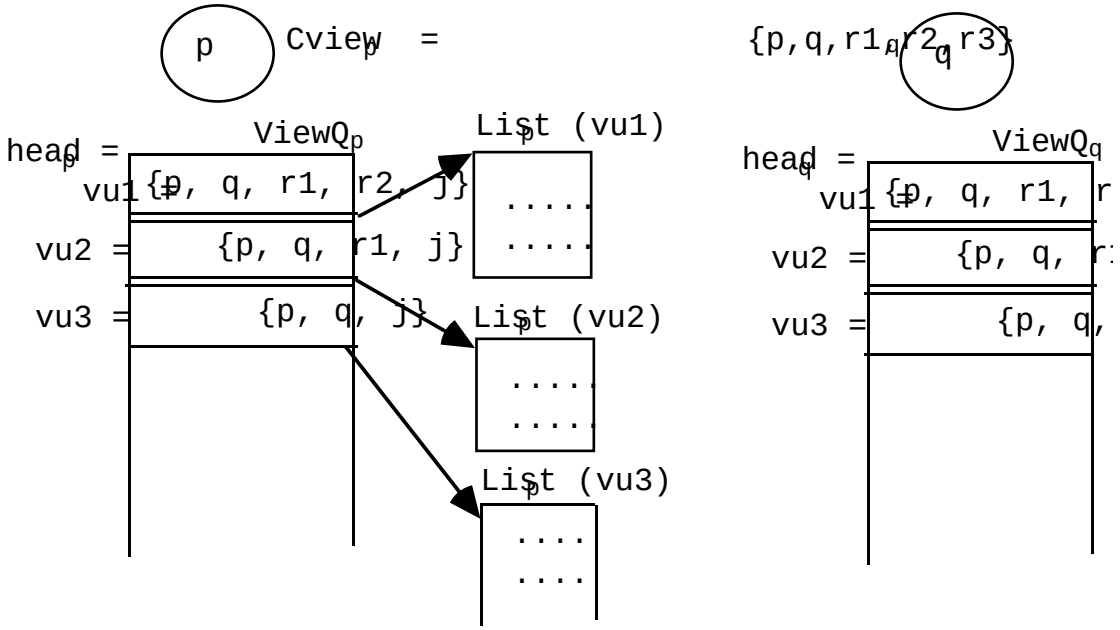



Figure 3. Closed and Open lists of messages delivered by VM after

The algorithm for evaluating $recd_p(m_q)$ is as follows: CMp waits following two comditions to become true.

Evaluation condition 1 (ec1): $\exists$ vu $\in$ $ViewQ_p$: $m_q \in List_p(vu) \wedge$ (vu $\in$ survivors($head_p$, vu)).

Evaluation condition 2 (ec2): $\exists$ vu $\in$ $ViewQ_p$: q $\notin$ vu.

The condition ec1 is true when $m_q$ is present in $List_p(vu)$ for s $ViewQ_p$ and q is present in all the views $VM_p$ constructed from he this vu; ec2 becomes true when $VM_p$ constructs an Mview without q.

boolean $recd_p(m_q)$
{wait until ec1 $\vee$ ec2; if ec1 then return true else return fal

Recall that $CM_p$ evaluates $recd_p(m_q)$ only for such q $\in$ $head_p$. Sup constructs an Mview vu that does not contain q (ec2). By messag property of VM, the expected message from q cannot be in List $List_p(vu')$ for vu' constructed prior to vu, is closed. If none contains $m_q$ (not ec1), then q crashed or disconnected before $recd_p(m_q)$ is evaluated to be false.

### 3.5. Examples

We illustrate the working of the view recording and installation examples. The first one is based on Figure 3. We assume that Cvie view-number$_p$ = k; also that p, q, and j remain connected and func $VM_j$ also constructs vu1, vu2, and vu3 as shown in the figure. Let $\lfloor(|CVu|/2)\rfloor$+1.

Suppose that r1 crashes after multicasting its State message SMsg( message RMsg(vu1). $CM_c$, c = p, q, or j, will find their respect installation conditions being met, and install vu1 = {p, q, r1, Cview. Following the installation of vu1, $CM_c$ delivers messages i which will be identical for every c. When $CM_c$ has $head_c$ = vu2, suspected, as $head_c$ contains a majority of components in the curr Since $head_c$ has no joiner, no recording condition needs to be met remain connected and functioning, $CM_c$ will find the installation c install vu2 as the (k+1)th Cview, and deliver messages in $List_c$(vu will install vu3 as the (k+3)th Cview and deliver messages in List( shows that when $VM_c$ and $VM_c$' construct an identical sequence of and $CM_c$' behave identically; they also deliver an identical set o two consecutive Cviews they install.

Suppose that r1 and r2 crash before multicasting their State messa $CM_c$, c = p, q, or j, will find the recording conditions not being $CM_p$ and $CM_q$ will proceed to execute the configuration protocol, wh with no change in its status (= spare) and mode (= waiting).

***Example with Concurrent Mviews***

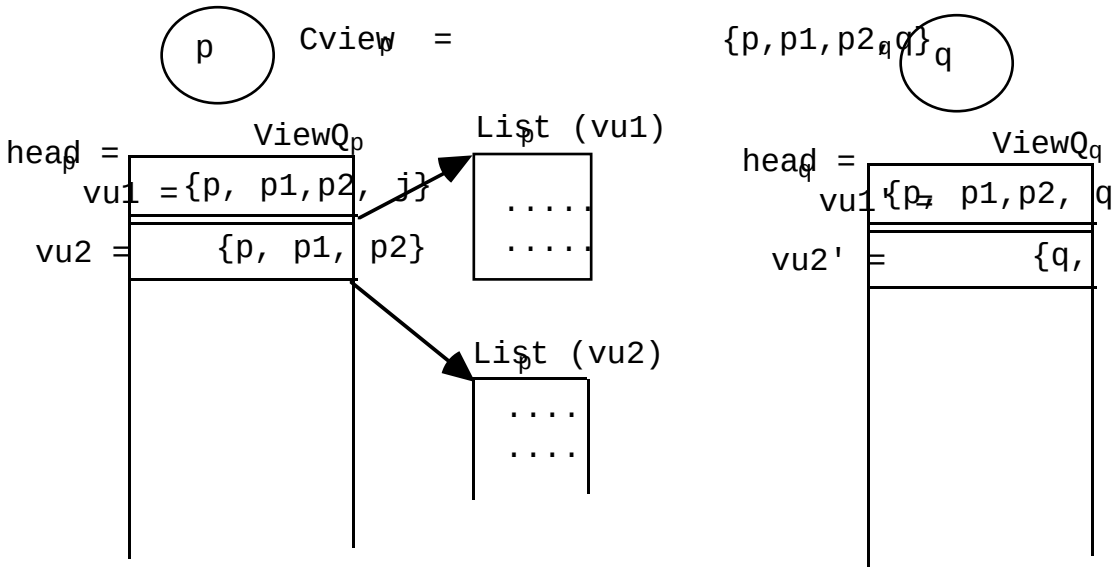



Figure 4. Concurrent and overlapping head views.

The second example is based on Figure 4 and illustrates scenarios member p to dequeue $head_p$ without recording it, and $CM_j$ of joiner configuration protocol with $Cview_j$ = null. As in the previous assume that $Cview_p$ is unique and $view\text{-}number_p$ = k. The figure sho attempting to join the group {p, p1, p2, q}, and $VM_p$ and $VM_q$ rea view agreement due to the subsequent detachment of {p, p1, p2} fr suppose that p, p1, and p2 remain connected to each other, and s denote p, p1, or p2, and c' denote q or j. Every $VM_c$ constructs for p in the figure, and every $VM_c$' constructs vu1' and vu2' show $VM_c$ and $VM_c$' have constructed non-identical, overlapping vu1 and

subsequently construct vu2 and vu2' respectively, due to the property (see subsection 2.1.1). Let $\Phi$(CVu) be defined to be 1, compute the component state by receiving a single State message.

When $CM_C$ has $head_C$ = vu1, it will find rc1 met; but it will not re from $CM_j$ and will find rc2 not being met. Hence, it dequeues hea messages in $List_C$(vu1) when $Cview_C$ is still {p, p1, p2, q}. Note not contain any application message from j as j does not yet c member. Thus, the view-message integrity property (of subsection 2 by $CM_C$. Since $\Phi$(CVu) = 1, both $CM_q$ and $CM_j$ will find for $head_C$ recording conditions being met, but not the installation condition to execute the reconfiguration protocol, with $Cview_q$ unchanged, $Sview_j$, and $Cview_j$ = null.

# 4. Reconfiguration after a g-failure

The configuration protocol presented here meets Requirement 2 following a g-failure, a unique set of functioning and connected c to become the (first post-g-failure) master subgroup. These comp group operations with an identical Cview called the restart-view. restart-view is taken to be either the last view or an Sview SVu last, the latter being the case if and only if a majority of last c SVu before the g-failure occurred. This invariant qualifies the res and ensures the continuity in the numbering of Cviews despite a g- subgroup is guaranteed unique by ensuring that it contains a maj rationale behind the formation of the master subgroup is briefly de

## 4.1. The Rationale

Let R be a set of components that get connected after a g-fai reconfiguration, it needs to be determined whether a subset of cc become the master subgroup. Let $Sview_r$ and $Cview_r$ denote the Sview of a component r in R, respectively. (If r is recovering from a cr and $Sview_r$ from its stable store.) Let presumed_last be the lates $Cview_r$ of all r in R: for every r in R, either presumed_last = Cvi » $Cview_r$. By definition, presumed_last is either the last Cview installed prior to the last.

Let us consider the Sviews recorded by the components of presume those in presumed_last $\cap$ R). One of the following three (mutu situations must exist:

(i) A majority of presumed_last components recorded (at some time identical Sview that is later than the presumed_last.

(ii) A majority of presumed_last components never recorded an Sv than the presumed_last.

---

[1]No last component could have installed SVu; otherwise the installed version of which, by definition, is the latest Cview installed by a component.

(iii) Neither 1 nor 2. That is, the number of presumed_last compor an Sview that is later than the presumed_last, is at most |presume the number of presumed_last components that never recorded an Svi than the presumed_last, is at most |presumed_last|/2.

Let us first consider situation (1). Suppose that R contain |presumed_last|/2 components with Sview = presumed_last+ an presumed_last, and (b) a majority subset of presumed_last+. We cla (b) are met, restart-view = presumed_last+. The (simple) proof is

Proof: Suppose that (a) and (b) are met but presumed_last+ ≠ rest of (a) implies that a majority of components in presumed_last presumed_last+. By the definition of restart-view, if restart-vie then restart-view » presumed_last+. For this to be true, a majori presumed_last+ must have installed presumed_last+ as their Cview have proceeded to record an Sview presumed_last++ (say), presur presumed_last+. None of these components that installed presumed Cview, can be in R, as per the way presumed_last is computed. Thi cannot be true - a contradiction.

Thus, when (a) and (b) are met, presumed_last+ becomes the res R∩presumed_last+ consider themselves to be the master subgroup.

In both situations (2) and (3), a majority of presumed_last ha progressive Sview that is later than presumed_last; therefore pre the last, and also the restart-view. To deduce the existence of (2) than |presumed_last|/2 components with Sview not later than presur (3) R must contain all presumed_last components.

Observe that deducing which one of the three situations exists, re least a majority of presumed_last components with appropriate Svie in the third situation. So, it is possible that a given R does no In that case, the attempt to form the master subgroup with R is recovery and reconnection of more number of components need to be a

### 4.2. The Protocol

The protocol is made up of five steps:

Step 0. $CM_p$ sets $mode_p$ to reconfiguration and waits for p to be other components, i.e., for $viewQ_p$ to become non-empty. Say, Vie non-empty and R = $head_p$. (Note: the first Mview in $ViewQ_p$ is only not dequeued.) The remaining four steps are done using R.

Step 1. Send {$Sview_p$, $Cview_p$} to every r in R (including itself); Receive {$SviewRecd_r$, $CviewRecd_r$} from every r in R;

Step 2. Determine the presumed_last to be the latest $CviewRecd_r$;

Step 3. Determine the restart-view if possible; if not possible ( $viewQ_p$, discard R and go to step 0.

Step 4. components of R ∩ restart-view:
install restart-view and resume group services;

```
components of R - restart-view:
     become spares;
```

Each step is described in detail in the subsections below.

### *4.2.1. Step 1: View Exchange*

$CM_p$ multicasts a message msg($Sview_p$, $Cview_p$, $mode_p$) containing Svi
and $mode_p$. It then evaluates the predicate $recd_p(msg_r(SviewRecd_r$
$mode_r))$ for every $r \in R$. If this predicate is true for a given
whether $CviewRecd_r \gg Cview_p$ and $mode_r$ = normal. If this conditi
exception Walked-Over is raised indicating that p has been slow i
which time the group is reconfigured without p. This exception is
p a spare and exiting the execution of the protocol. If the predic
then working with R is given up: terminate the execution with R,
$viewQ_p$, and go to step 0. The pseudo-code for step 1 is given belo

```
multicast msg(Sview_p, Cview_p, mode_p) to all r in R;
evaluate recd_p(msg_r(SviewRecd_r, CviewRecd_r, mode_r))  for every
catch (Walked-Over): { write atomically: {Sview_p := null; sta
                                    mode_p := waiting;}
                     exit; }
if (∃r ∈ R: ¬recd_p(msg_r(SviewRecd_r, CviewRecd_r, mode_r)))
     then {give up on R;}
```

### *4.2.2. Step 2: Determine presumed_last*

presumed_last is computed to be the latest non-null view among the
If a majority of presumed_last is not in R, then the execution wi
up.

```
{  presumed_last :=  CviewRecd_r of some r ∈ R: (presumed_last ≠ nu
     (∀ r' ∈ R: presumed_last = CviewRecd_r' ∨presumed_last »CviewR
  if (| presumed_last ∩ R| ≤ (| presumed_last |/2) then { give up
}
```

Note that by requiring that presumed-last be non-null, an R of onl
with mode = waiting are prohibited from forming the master subgrou

### *4.2.3. Step 3: Attempt to Determine restart-view*

$CM_p$ divides the components in presumed_last ∩ R into non-overla
called candidate sets and denoted as $CS_v$, $v \geq 0$, based on the comp
$presumed\_last+_i$, $i \geq 1$, be an Sview[2] that is later than presumed_las

---

[2] presumed_last+ need not be unique after a g-failure; different presumed_last com
recorded different progressive Sviews, due to their VM modules concluding view agr
points. Let, for example, last = {1,2,3,4,5}. Let C5 crash and VM of C4 reach agree
{1,2,3,4}. If VMs of C1, C2, and C3 suspect C4 before they reach agreement on {1,2,3
agreement straightaway on {1,2,3}. If a g-failure occurs after CMs have recorded the
= {1,2,3,4} (say, $presumed\_last+_1$), and $Sview_1 = Sview_2 = Sview_3$ = {1,2,3} (say, presu

contains the components in presumed_last ∩ R whose Sview presumed_last+$_V$; $CS_0$ contains those components in presumed_last SviewRecd$_r$ is not later than presumed_last. The code for this t below.

```
(1)  if (∃ CS_V: v ≥1 ∧ CS_V ∈ M_SETS(presumed_last)
                ∧ presumed_last+_V ∩ R ∈M_SETS(presumed_last+_V) ) th
          {restart-view := presumed_last+_V;}
               // existence of situation (1) is deduced
(2)  else if (CS_0 ∈ M_SETS(presumed_last) then
          {restart-view := presumed_last;}
               // existence of situation (2) is deduced
(3)  else if (presumed_last ⊆ R) then
          {restart-view := presumed_last;}
               // existence of situation (3) is deduced
     else {give up on R;}
```

#### *4.2.4. Step 4: Commencing Group Operations*

Any p that is not in the restart-view becomes a spare, otherwise information in its stable store. The pseudo code is as follows:

```
{    if (p ∉ restart-view) then { write atomically:
                        {Sview_p, Cview_p := null; status_p := spare
                        mode_p := waiting; view-number_p := -1;}
                       exit; }
     write atomically:
          {view-number_p:= view-number(restart-view);
          Sview_p, Cview_p:= restart-view; status_p := member;
          mode_p := normal; }
}
```

### 4.2 Examples

We explain the working of the protocol with the help of examples a the evolution of Sviews depicted in Table 1. For simplicity, assum considered in this discussion are caused by node crashes only, a occur only when the group is being reconfigured after a g-failure.

Table 1 depicts a possible sequence of membership changes for a gr adopts the following style to represent the state of the view inst normal font indicates that it has been installed as the Cview$_p$; an be installed is written in mixed fonts: survivors (from the curr Sview$_p$) in normal font, joiners in italics and excluded components are in the current Cview$_p$ but not in the Sview$_p$) in bold. The Sview$_p$ indicates its view-number.

| Stage No | Sview of C1, C2 | Sview of C3 | Sview of C4, C5 | Sview of C6, C7 | Description |
|---|---|---|---|---|---|
| 1 | $\{1,2,3,4,5\}^0$ | $\{1,2,3,4,5\}^0$ | $\{1,2,3,4,5\}^0$ | --- | group initialised, n= C6 and C7 are spares |
| 2 | $\{1,2,3,4,5\}^1$ | $\{1,2,3,4,5\}^0$ | $\{1,2,3,4,5\}^0$ | --- | C4 and C5 crash; CM1 a CM2 record their excl first |
| 3 | $\{1,2,3,4,5\}^1$ | $\{1,2,3,4,5\}^1$ | $\{1,2,3,4,5\}^0$ | --- | Slow CM3 records Sview |
| 4 | $\{1,2,3,4,5\}^1$ | $\{1,2,3,4,5\}^1$ | $\{1,2,3,4,5\}^0$ | --- | CM1 & CM2 install Sview(1) |
| 5 | $\{1,2,3,4,5\}^1$ | $\{1,2,3,4,5\}^1$ | $\{1,2,3,4,5\}^0$ | --- | CM3 install its Sview( |
| 6 | $\{1,2,3,6,7\}^2$ | $\{1,2,3,6,7\}^2$ | $\{1,2,3,4,5\}^0$ | $\{1,2,3,6,7\}^2$ | C6 & C7 join; all act CM record Sview(2) = {1,2,3,6,7} |
| 7 | $\{1,2,3,6,7\}^2$ | $\{1,2,3,6,7\}^2$ | $\{1,2,3,4,5\}^0$ | $\{1,2,3,6,7\}^2$ | CM3, CM6, and CM7 install Sview(2) |
| 8 | $\{1,2,3,6,7\}^2$ | $\{3,6,7\}^3$ | $\{1,2,3,4,5\}^0$ | $\{3,6,7\}^3$ | C1, C2 crash before i Mview(2); CM3, CM6, CM record and then instal |

Table 1. An evolution of Sviews.

The group is initially formed with {C1, C2, C3, C4, C5}. At the e member has $\{1,2,3,4,5\}^0$ as its (initial) Sview in stable store; thi the end of stage 2, $CM_1$ and $CM_2$ have recorded Sview(1) which cann now as $\{1,2\} \notin$ M_SETS(Sview(0)). The situation changes after sta and $CM_2$ install Sview(1) in stage 4. In stage 6, the spares C6 an CM1, CM2, CM3, CM6, and CM7 record the Sview $\{1,2,3,6,7\}$. In the C3, C6, and C7 install the Sview, since all components in the ol known to have recorded $\{1,2,3,6,7\}$. But C1 and C2 crash before the recorded view. In stage 8, C3, C6, and C7 install the Cview wit According to Table 1, $\{1,2,3,6,7\} \gg \{1,2,3\}$ and $\{1,2,3,6,7\} \gg \{1$ last is $\{1,2,3,4,5\}^0$ until stage 3, $\{1,2,3\}^1$ in stages 4, 5, and 6, and $\{3,6,7\}^3$ in stage 8.

Example 0:This example shows that the protocol is safe in not allo one master subgroup to be formed after a g-failure. Let C1 and C connected after stage 8 but remain partitioned from other compone C2}. Both C1 and C2 will estimate presumed_last to be $\{1,2,3\}^1$. Si is already functioning as the group, another master subgroup shoul emerge from R even though R contains a majority subset of pr Components of R will find that they have an identical (prog $\{1,2,3,6,7\}^2 \gg$ presumed_last, and R does not contain a majo $\{1,2,3,6,7\}^2$. So, none of the conditions in Step 3 of the protoc will be given up.

Example 1:This exemplifies the behaviour of the protocol under t mentioned in section 4.1. Suppose that a g-failure occurs immedia

Here, last is $\{1,2,3\}^1$ and all of the last components have record
restart-view is $\{1,2,3,6,7\}^2$. Say, R = {C1, C2, C4, C6}. By step
each component r in R determines restart-view to be $\{1,2,3,6,7\}^2$. F
restart-view, C4 will exit the protocol and join the pool of spare
restart-view) install restart-view as their Cview and resume norr
Note that the view R is still the at head of every $ViewQ_r$, $r \in$
ViewQ not empty, CMs of ($R \cap$ restart-view) will execute the vi
protocol as members and CM of C4 as a joiner. Assuming no furth
disconnections, C1, C2, C4, and C6 will get install R as the Cview.

Example 2 considers situation 2 where a majority of presumed-last
an Sview that is later than presumed-last. Say, a g-failure occurs
last = $\{1,2,3,4,5\}^0$. Since no last component has recorded a later S
also $\{1,2,3,4,5\}^0$; further, since all last components have identi
contains any three (majority subset) of the last members will
subgroup. Say, R = {C3, C4, C5}. Assuming that R remains connected
CM of R computes presumed-last to be $\{1,2,3,4,5\}^0$, i.e., last its
of R forms $CS_0$ = R and decides in step 3.2 of the protocol the
presumed-last = $\{1,2,3,4,5\}^0$. After (re-)installing restart-view
of C3, C4, and C5 subsequently install R as their next Cview = {3,

Say, after CMs of C3, C4, and C5 have installed $\{3, 4, 5\}^1$ in the
C1 and C2 recover and reconnect with C3, C4, and C5. While CMs c
execute the reconfiguration protocol with R = {1,2,3,4,5}, CMs o
will execute view installation protocol for the delivered view {1,
C1 and C2 are regarded as joiners. This conflict gets resolved ver
C4, and C5 expect CMs of C1 and C2 to send recorded messages but
messages of configuration protocol. They would then respond by s
and Cview to $CM_1$ and $CM_2$ which would get Walked_Over exception,
spares, and then start executing the view installation protocol as
of their ViewQ (still) having R = {1,2,3,4,5}.

In example 3, we illustrate the need for R to contain all the las
circumstances. Let a g-failure occur at the end of stage 2. The
$\{1,2,3,4,5\}^0$ which is also the Cview of every member component. We
= {C1, C2, C4, C5}. The presumed_last is the same as last = {1,
component of R knows that a minority of presumed_last (i.e. two)
an Sview that is later than presumed_last; and also that on
presumed_last (i.e. two again) are known to have recorded an Sview
is later than presumed_last. When Sview of C3 is $\{1,2,3,4,5\}^0$ (a
restart-view becomes $\{1,2,3,4,5\}^0$. If Sview of C3 had been {1,2,3,4
end of Stage 3), then restart-view becomes $\{1,2,3\}^1$. Hence determi
view requires that the components of R know the Sview of C3. Here,
step 3.3 of the protocol which requires R to contain presumed_las
situation 1 nor 2 is known to prevail.

## 5. Related Work

Using the traditional, 2-Phase Commit (2PC) protocol [Gray78]
updating membership-related information, [Jajodia90] maintains
distinguished partition in a replicated database system. Our CM s

a variation of this traditional 2PC for Cview installation and the
by our requirements and efficiency. In the traditional 2PC way of
the coordinator - a deterministically chosen member in the new Cvi
the second (view-installation) phase after learning during the f
component of the new Cview has recorded the view. Note that while
is in progress, delivery of application messages is put on hold
synchrony. Since we only require that at least a majority subset,
the current Cview install the next Cview, we can speed up the vi
having the coordinator initiate the second phase as soon as a maj
Cview and all joiners (if any) in the new Cview have recorded th
soon as the installation conditions of section 3.3 are met. Furt
based execution of traditional 2PC are susceptible to co-ordin
eliminate this weakness by executing our version of 2PC in a dec
where every component checks installation conditions.

Since we use a 2PC protocol for view installations, the configurat
be non-blocking. This blocking can be removed by using a 3PC prot
The protocol of [Dolev97] employs the principles of an extende
[Keid95] and builds a unique master subgroup after a g-failure. No
architecture is remarkably similar to theirs. They differ from our
major aspect: a component can have, and may have to exchange, more
so more stable information needs to be maintained and message si
Obviously these features of [Dolev97] increase the overhead of t
advantage, on the other hand, is that a reconnected set need only
majority subset of last, never all last components as we would req
when a g-failure occurs during view update (see example 3 of sectio

The primary partition membership service in [Birman87, Ricciardi91,
the assumption that a majority of components in the Cview do not s
and that a functioning component is rarely detected as failed. Th
not hold true during periods of network instability caused for
traffic or network congestion. This instability can lead to incor
which in turn can lead to g-failures. In these circumstances, our
[Dolev97]) can provide recovery from g-failures once the network tr

[Chandra96] establishes the weakest failure detector (denoted as
the consensus problem. Using this consensus protocol, a (pri
membership service is designed [Malloth95] and implemented [Fe
membership service can construct a totally ordered sequence of vie
of each view surviving into the next view. It blocks from deliverir
the periods of g-failures (i.e., when a connected majority does
blocking is released as soon as the requirements of ◊S are realised
based membership service provides recovery from g-failures for th
requirements of ◊S? The answer appears to be no. The first view wh
membership service constructs after a g-failure, is what we call
requirement 2 in §2.5). Determining the restart-view does not nec
the new master subgroup exists to restart the group services. To s
following example. Let the current view be {p, q, r, s, t} with vi
and t crash before a g-failure occurs. With the ◊S based members
possible for R = {r, s, t} to reconnect and decide that the (k+1

Though the restart-view is now known, the group operations cannnot does not contain a majority subset of the restart-view. (Permittir form the master subgroup will lead to two concurrent master subgr operating in a seperate partition.) Group re-configuration with R for either p or q to recover/reconnect. Our approach is differen view cannot be determined until the reconnected set (R) contains a the restart view itself (see section 4.1); in other words, determ straight leads to the re-formation of the group (barring the occ failures). As a future work, we intend to compare these two appro more detailed manner.

## 6. Conclusions

Group failures can occur even in the absence of any physical fail by sudden bursts in message traffic with potentials to lead to v have designed and implemented a configuration management subsyste provide automatic recovery from group failures, once the real/ disappear and components recover. Our system employs a variatior commit protocol for view updates. Consequently, the recovery provi blocking. On the other hand, it is efficient in terms of message and use of stable store, during both normal operations and recor group failure; it costs only one extra message round to update vi failure-free periods. This low, failure-free overhead makes our suited to soft real-time systems where it can be incorporated in in [Hurfin98].

### Acknowledgements

The work described here has been supported in part by Hewl Laboratories, Bristol. Roger Flemming and Paul Harry from Hew Laboratories provided facilities and resources for implementing o system over Somersault. Discussions with Andre Schiper over ◊S-ba protocol were useful in clarifying many subtle issues.

# References

[Amir92] Y. Amir, Dolev, D., Kramer, S., and Malki, D., "Membership Algorith Communication Groups", Proc. of 6th Intl. Workshop on Dist. Algorithms, pp 29 1992.

[Babaoglu95] O.Babaoglu, R. Davoli, and A Montresor, "Group Membership and View Partitionable Asynchronous Distributed Systems: Specifications", Technical Rep Dept. of Computer Science, University of Bologna, Italy, Nov 1995.

[Babaoglu97] O.Babaoglu, A. Bartoli, and G Dini, "Enriched View Synchrony: A Paradigm for Partitionable Asynchronous Distributed Systems", IEEE ToCS, 46 pp.642-658.

[Birman87] K.Birman and T. Joseph, "Exploiting virtual synchrony in distribute 11th ACM Symposium on Operating System Principles, Austin, November 1987, pp. 1

[Black97] D. Black, P. Ezhilchelvan and S.K. Shrivastava, “Determining the La Process Group after a Total Failure”, Tech. Report No. 602, Dept. of Computing of Newcastle upon Tyne.

[Chandra96] T. D. Chandra, V. Hadzilacos, and S. Toueg, "The weakest Failure D Consensus", JACM, 43(4), pp. 685 - 722, July 1996.

[Ezhil95] P. Ezhilchelvan, R. Macedo and S. K. Shrivastava, “Newtop: a fau communication protocol”, 15th IEEE Intl. Conf. on Distributed Computing Systems 1995, pp. 296-306.

[Felber98] P Felber, R Guerraoui and A Schiper, "The implementation of CORBA Theory and Prctice of Object Systems, Vol. 4, No. 2, 1998, pp. 93-105.

[Gray78] J N Gray. Notes on Database Operating Systems. In Operating System Course, Lecture Notes In Computer Science, Vol 60, pp. 393-481. Springer Verlag

[Guerr95] R Guerraoui, M Larrea and A Schiper. Non Blocking Atomic Commitm Unreliable Failure Detector. Proceedings of IEEE Symposium on Reliable Distribut Bad Neunhar, Germany, September 1995, pp. 41-51.

[Hurfin98] M. Hurfin and M. Raynal, "Asynchronous Protocols to Meet Real-Time Really Sensible? How to Proceed?", Proc. of 1st Int. Symp. on Object-Oriented R Computing, (ISORC98) pp. 290-297, April 98.

[Jajodia90] S Jajodia and D Mutchler. Dynamic Voting Algorithms for Maintainir of a Replicated Database. ACM Transactions on Database Systems, Vol 15, No 2 230-280

[Keid95] I Keidar and D Dolev. Increasing the Resilience of Distributed and Systems. Journal of Computer and System Sciences (JCSS). 1995.

[Lotem97] E Y Lotem, I Keidar and D Dolev. Dynamic Voting for Consisten Components. Proceedings of ACM Symposium on Principles of Distributed Computin 63-71, 1997.

[Malloth95] C Malloth and A Schiper, "Virtually Synchronous Communication in I Networks", BROADCAST Third Year Report, Vol 3, Chapter 2, July 1995. (Anonymo broadcast.esprit.ec.org in directory projects/broadcast/reports)

[Melliar-Smith91] P. M. Melliar-Smith, Moser L.E., and Agarwala, V., "Membersh Asynchronous Distributed Systems", Proc. of 12th Intl. Conf. on Distributed 480-488, May 1991.

[Mishra91] S. Mishra, L. Peterson and R. Schlichting, “A membership Protocol Order”, Proc. IFIP Conf. on Dependable Computing For Critical Applications, Tus 137-145.

[Moser96] L.E. Moser, P.M. Melliar-Smith et al, “Totem: a Fault-tolerant communication system”, CACM, 39 (4), April 1996, pp. 54-63.

[Murray97] P. Murray, R. Flemming, P. Harry and P. Vickers, “Somersault softwa Hewlett-Packard Technical Report, 1997.

[Ricciardi91] A. Ricciardi and K P Birman, "Using Process Groups to Implement F
Asynchronous Environments", In Proceedings of ACM symposium on PoDC, pp. 480-48

[Schiper94] A Schiper and A Sandoz, "Primary-Partition Virtually Synchronous
Harder Than Consensus", Proc. of the 8th International Workshop on Distributed A
94), Sept. 94, LNCS 857, Springer Verlag. (Also in BROADCAST Second Year Re
October 1994).

[Skeen 81] D. Skeen, “Non-Blocking Commit Protocols”, ACM SIGMOD, pp.133 - 142,