



Teoria della Replicazione

Sistemi Distribuiti a.a. 2004/2005

Prof. Roberto Baldoni, Ing. Alessia Milani



Argomenti (1)

- Introduzione alla replicazione:
 - Modello
 - Requisiti
 - Trasparenza
 - Consistenza
- Replicazione software:
 - Tolleranza ai guasti
 - Linearizability

Argomenti (2)

- Tecniche di replicazione
 - Primary Backup
 - Active Replication
- Memorie condivise distribuite:
 - Comunicazione
 - Criteri di consistenza (specifici):
 - Sequential consistency
 - Causal consistency
 - PRAM
 - Criteri di consistenza (protocolli):
 - Protocollo sequenziale
 - Protocollo causal (Ahmad)

Modello per la Replicazione : oggetto logico

- N processi sequenziali p_1, p_2, \dots, p_n che interagiscono attraverso un insieme X di oggetti (file, variabili, code, ecc...)
- Un oggetto $x \in X$ ha uno stato (**oggetto logico**)
- I processi accedono allo stato dell'oggetto attraverso operazioni \bullet .
- Processo sequenziale: invoca \bullet e resta bloccato fino a quando ottiene la corrispondente risposta.
- Un'operazione \bullet eseguita da un processo p_i su un oggetto x è una coppia invocazione-risposta, $[x \bullet(\arg) p_i]/[ok(\text{res}) p_i]$.
- Il set di operazioni che gestiscono l'oggetto ne determinano la semantica.

Esempio: coda FIFO

- o Coda FIFO q
- o Stato: contenuto della coda in un dato momento.
Es. $q=[c\ b\ d]$ c e' in cima alla coda.
- o Operazioni:
 - Quando un processo vuole inserire un elemento nella coda, es. a , invoca $ENQ(a)$. Quando riceve $Ok()$ e' sicuro che l'elemento e' stato inserito.
Es. Stato della coda dopo l'inserimento di a :
 $q=[c\ b\ d\ a]$
 - Quando un processo vuole estrarre un elemento dalla coda invoca $DEQ()$. Se la coda non e' vuota ottiene come risposta $Ok(c)$.
Es. Stato della coda dopo l'estrazione di c :
 $q=[b\ d\ a]$

Modello per la Replicazione : repliche fisiche

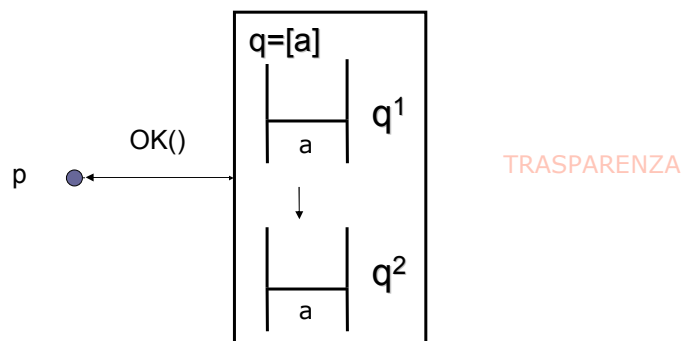
- o Ogni "oggetto logico" x corrisponde a diverse copie fisiche, "repliche", distribuite nel sistema, denotate x^1, x^2, \dots, x^k .
- o Le invocazioni sulla replica x^k collocata nel sito s sono gestite dal gestore della replica in s .

Requisiti

- **Trasparenza:**
 - Il processo deve credere di interagire con l'oggetto logico.
 - La replicazione non cambia:
 - Il modo in cui un processo invoca un'operazione. Es. `ENQ(a)`
 - Il modo in cui vengono restituite le risposte. Es. `OK()`
- **Consistenza:**
 - Le operazioni eseguite su una collezione di oggetti replicati devono produrre risultati che rispettino le specifiche di correttezza per tali oggetti.
 - Le specifiche di correttezza dipendono dalla semantica.

Esempio Trasparenza: Coda FIFO Replicata

- Implementiamo la coda FIFO q con 2 repliche q^1 e q^2 .
- Assumiamo q inizialmente vuota.



Esempio Consistenza: Coda FIFO

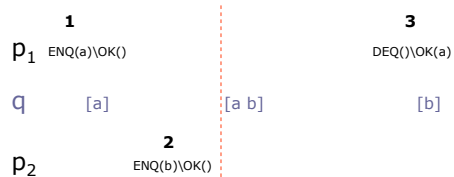
- Assumiamo che un processo p acceda ad una coda FIFO q inizialmente vuota.

Stato q	Operazione	Risultato operazione	Nuovo stato q
coda vuota	ENQ(a)	OK()	[a]
[a]	ENQ(b)	OK()	[a b]
[a b]	DEQ()	<ol style="list-style-type: none"> OK(a) risultato CONSISTENTE OK(b) risultato INCONSISTENTE 	<ol style="list-style-type: none"> [b] [a]

Consistenza e Concorrenza

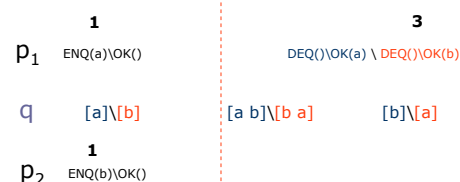
- Sistema sequenziale:

- In ogni istante di tempo un solo processo nel sistema esegue un'operazione sull'oggetto.
- Le pre e post condizioni mi definiscono il significato di un'operazione



- Sistema concorrente:

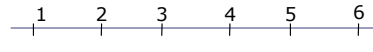
- Piu' processi concorrono per accedere all'oggetto.
- Si deve dare un significato al possibile ordinamento delle operazioni.



Relazioni temporali (real time) tra le Operazioni

Notazione:

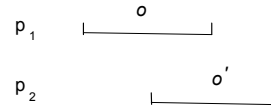
1. $t_{inv}(o)$ istante (real time) in cui p_i invoca o .
2. $t_{res}(o)$ istante (real time) in cui p_i riceve la corrispondente risposta.



- Due operazioni o e o' , sono dette **sequenziali**, denotato $o < o'$, se la risposta di o precede l'invocazione di o' . ($t_{res}(o)$ minore $t_{inv}(o')$)



- Due operazioni o e o' , sono dette **concorrenti**, denotato $o || o'$, se $\neg(o < o')$ e $\neg(o' < o)$.



Replicazione Software

11

Replicazione Software

Tolleranza ai guasti

- Assicurare un comportamento corretto del servizio a fronte di un certo numero e tipo di guasti.
- Modello di guasto: CRASH
 - Un processo è **guasto** se esegue correttamente il suo algoritmo fino ad un istante t dopo il quale si ferma prematuramente e non esegue più alcuna azione.
 - Un processo p è **corretto** se non si verifica mai un crash per p e lo stesso esegue un numero infinito di passi.

Disponibilità

- Il servizio deve essere accessibile con tempo di risposta ragionevole per una frazione di tempo prossima al 100%
- Es. Sia **O** un oggetto replicato su n processo la cui probabilità di guastarsi sia p (indipendente dalla probabilità di guasto degli altri processi).

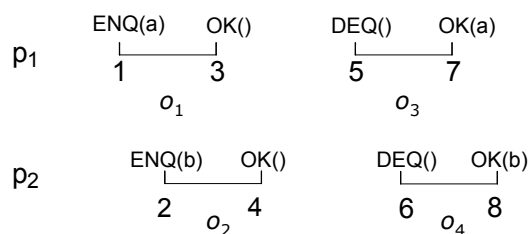
Disponibilità di **O**:

$$1 - p^n$$

Linearizability

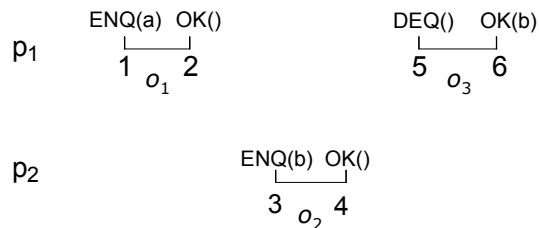
- Motivazioni:
 - l'effetto di ogni operazione deve essere lo stesso che si avrebbe qualora la stessa fosse stata istantanea;
 - l'ordine tra operazioni sequenziali deve essere preservato
- Un'esecuzione E è **linearizzabile** se esiste una sequenza S contenente tutte le operazioni in E tali che:
 - \forall due operazioni o e o' in E tali che $o < o'$, o appare prima di o' in S ;
 - La sequenza S rispetta la semantica dell'oggetto.

Linearizability: Esempio coda FIFO (1)



- Assumiamo la coda inizialmente vuota.
- E e' l'esecuzione costituita dal seguente insieme di operazioni $\{o_1, o_2, o_3, o_4\}$ e t.c. $o_1 < o_3$, $o_1 < o_4$, $o_2 < o_3$ e $o_2 < o_4$.
- E e' linearizzabile perche' esiste $S = o_2 o_1 o_4 o_3$ tale che:
 - Rispetta l'ordinamento real time,
 - S e' **legale** perche' rispetta le specifiche di una coda FIFO: b e' messo in coda prima di a , quindi la prima operazione di estrazione restituisce b e la seconda a .

Linearizability: Esempio coda FIFO (2)



- o Assumiamo la coda inizialmente vuota.
- o E e' l'esecuzione costituita dal seguente insieme di operazioni $\{o_1, o_2, o_3\}$ e t.c. $o_1 < o_2, o_1 < o_3, o_2 < o_3$.
- o E NON e' LINEARIZABILE perche' NON ESISTE S tale che:
 - Rispetta l'ordinamento real time,
 - Rispetta le specifiche di una coda FIFO: a è messo in coda prima di b, quindi la prima operazione di estrazione dovrebbe restituire a.

Assicurare Linearizability

- o Condizione sufficiente a garantire la linearizability e' che le repliche siano concordi sul set di invocazioni ricevute e sull'ordine delle stesse.
- o Formalmente:
 - **Ordine:** date due invocazioni, se due repliche eseguono entrambe le invocazioni, le eseguono in uno stesso ordine.
 - **Atomicità:** se una replica esegue un'invocazione allora tutte le repliche non guaste devono eseguire tale invocazione.

Tecniche di Replicazione che assicurano linearizability

- Due fondamentali tecniche di replicazione che assicurano linearizability sono:
 - Primary Backup
 - Active Replication

Modello di Sistema

- Insieme finito di processi sequenziali $\{p_1, p_2, \dots, p_n\}$ che invocano le operazioni sugli oggetti, detti client.
- Insieme finito di processi sequenziali $\{x^1, x^2, \dots, x^m\}$ che gestiscono le repliche dell'oggetto x , gestori delle repliche.
- Modello di comunicazione a scambio di messaggi
- **Perfect link:**
 - **Reliable delivery:** se un processo p manda un messaggio m ad un processo q e nessuno dei due si guasta, allora q prima o poi riceve m .
 - **No duplication:** nessun messaggio è consegnato da un processo più di una volta
 - **No creation:** Se un messaggio m è consegnato da un qualche processo p , allora m è stato precedentemente inviato da un qualche processo q .
- **Crash**

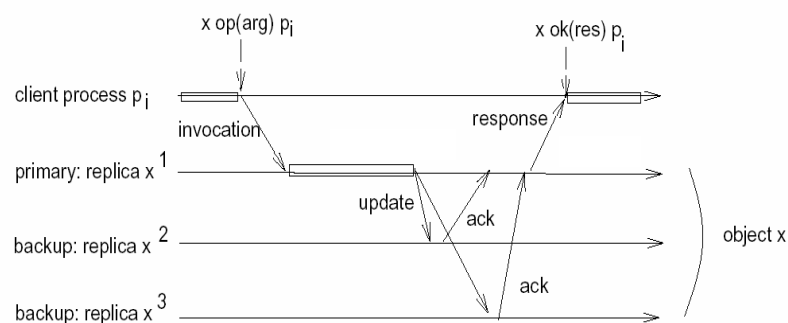
Primary Backup

- Le repliche non hanno tutte lo stesso ruolo:
 - **Primary:**
 - riceve le invocazioni dal processo client e restituisce la risposta.
 - Dato un oggetto x , il primary di x è denotata $prim(x)$.
 - **Backup:**
 - interagiscono solo con la $prim(x)$
 - servono per garantire la tolleranza ai guasti

Primary Backup

21

Primary Backup: Scenario di Funzionamento



Primary Backup

22

Primary Backup: Assenza di Crash

- Il client p_i invoca $o(\text{arg})$ su x .
- $\text{prim}(x)$ riceve l'invocazione ed esegue l'operazione.
- Dopo l'esecuzione $\text{prim}(x)$ ha aggiornato il suo stato e la risposta, res , è disponibile.
- $\text{prim}(x)$ invia dei messaggi di update (invId , res , state-update) ai backup :
 - **invId**: identificativo univoco dell'invocazione
 - **res**: risposta per il client
 - **state-update**: stato del primary dopo aver eseguito la corrente invocazione.

Primary Backup: Assenza di Crash (2)

- Quando ricevono gli update, i backup aggiornano il loro stato e inviano l'ack al $\text{prim}(x)$.
- Il $\text{prim}(x)$ aspetta gli ack di tutti i backup corretti e poi invia la risposta, res , al client.
- **Garantire Linearizability**: l'ordine in cui $\text{prim}(x)$ riceve le invocazioni determina l'ordinamento totale delle operazioni sull'oggetto.

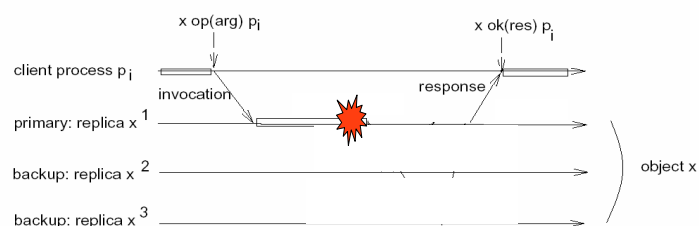
Primary Backup: Presenza di Crash

- o Distinguiamo tre scenari di guasto del *primary*:
 1. **Scenario 1**: Guasto dopo che il client ha ricevuto la risposta.
 2. **Scenario 2**: Guasto prima di inviare i messaggi di update.
 3. **Scenario 3**: Guasto dopo aver inviato i messaggi di update ma prima di aver ricevuto gli ack.
- o In tutti e tre i casi si deve eleggere un nuovo *primary*: *elezione del leader*.

Primary Backup

25

Scenario 1

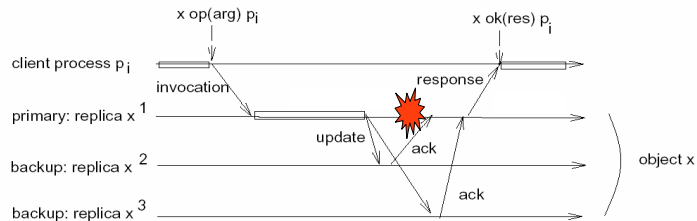


- o Guasto prima di inviare i messaggi di update
 - Il client non ottiene una risposta e quindi sospetta il guasto.
 - Il nuovo primary tratta l'invocazione come se fosse nuova

Primary Backup

26

Scenario 2



- Guasto dopo aver inviato i messaggi di update ma prima di aver ricevuto gli ack:
 - **Garantire atomicità**: l'update è ricevuto da **tutti** o da **nessuno**.

Primary Backup

27

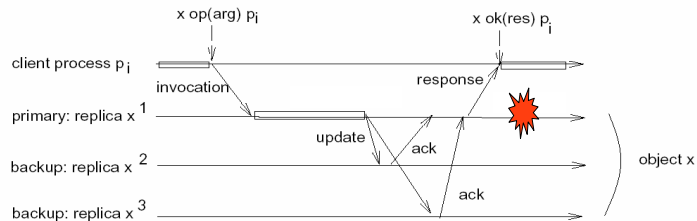
Scenario 2: possibili sottoscenari

- **Nessun** backup ha ricevuto l'update:
 - si torna allo **scenario 1**
- **Tutti** i backup hanno ricevuto l'update implica che l'operazione è stata eseguita ma il client non ha avuto risposta:
 - Lo stato dei backup è aggiornato
 - Il client invoca nuovamente la stessa operazione
 - Il nuovo primary usa le informazioni (invId,res) per capire che l'invocazione è già stata gestita e mandare al client la res precedentemente calcolata.

Primary Backup

28

Scenario 3



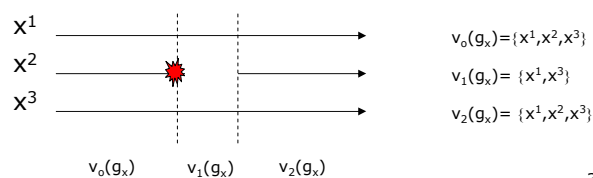
- o Guasto dopo che il client ha ricevuto la risposta:
 - Il client non percepisce il guasto

Primary Backup

29

Primary Backup: Leader Election

- o Quando il **prim(x)** si guasta è necessario eleggerne un altro tra le repliche corrette: **Leader election**
- o L'insieme di repliche corrette nel sistema può variare nel tempo a causa dei crash e dei ripristini.
- o **Gruppo g_x** : insieme di repliche che gestiscono l'oggetto x .
- o **Gruppi dinamici**: la composizione del gruppo varia durante il ciclo di vita del sistema:
 - o quando una replica x^i si guasta, viene rimossa dal gruppo,
 - o quando una replica x^i si ripristina viene riaggiunta al gruppo.
- o **View**: modella l'evolvere della composizione del gruppo.
 - $v_0(g_x)$ composizione iniziale del gruppo
 - $v_i(g_x)$ i -esima composizione del gruppo



30

Primary-Backup: Leader Election (2)

- Supponiamo che ci sia una regola R che ordina le repliche all'interno di una view.
 - Es. R=ordine crescente dell'identificativo della replica:
 - x^1, x^2, x^4 OK
 - x^2, x^1, x^4 **ERRORE**
- Il nuovo primary può essere la prima replica nella **view corrente** in accordo ad R.
 - Definizione primary:
 - $v_0(x^1, x^2, x^4)$, primary x^1
 - $v_1(x^2, x^4)$, primary x^2
- **CONSEGNA ORDINATA DELLE VIEW.**

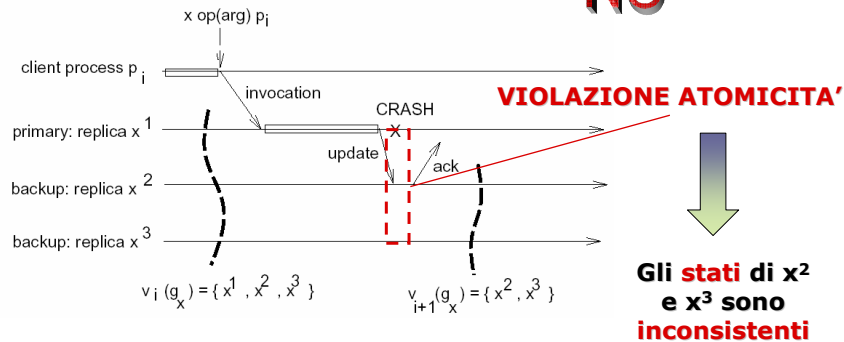
Consegna Ordinata delle view: Specifica

- L'ordinamento delle view deve soddisfare le seguenti specifiche:
 - Se un processo $p \in g_x$ si guasta, viene definita una nuova vista di cui p non fa parte
 - Se un processo p viene ripristinato e fa nuovamente parte del gruppo g_x , viene definita una nuova vista che comprende p.
 - Data una vista $v_i(g_x)$ t.c. un processo $p \in v_i(g_x)$, allora o p consegna $v_i(g_x)$ oppure \exists un $k > 0$ t.c. $p \in v_{i+k}(g_x)$.
 - $\forall p, q \in g_x$, se p e q consegnano $v_i(g_x)$ e $v_j(g_x)$ con $i \neq j$, allora le consegnano nello stesso ordine.
- **NOTA:** è **IRRILEVANTE** se una replica è stata eliminata dal gruppo perchè realmente guasta oppure perchè erroneamente sospettata.

Primary-Backup: Correttezza

- o E' sufficiente garantire ordinamento delle invocazioni e delle viste per garantire CORRETTEZZA?

NO



Primary Backup

33

Primary-Backup Garantire Atomicità

- o Il messaggio di update inviato dal primary deve essere consegnato da TUTTE o da NESSUNA delle repliche *backup* corrette.

VIEW
SYNCHRONOUS
MULTICAST

Primary Backup

34

View-Synchronous Multicast: Specifica

- o Estende la semantica del reliable multicast prendendo in considerazione i cambiamenti delle viste del gruppo.
- o Assunzioni: $v_0(g_x)$ è la view iniziale di ogni processo di g_x

DEF[View-synchronous multicast]

Dato un gruppo g_x , una view $v_i(g_x)$ e un messaggio m mandato in multicast a tutti i processi appartenenti a g_x , allora il multicast di m è view-synchronous in $v_i(g_x)$ se e solo se:
 se $\exists p \in v_i(g_x)$ che ha consegnato m in $v_i(g_x)$ e ha consegnato $v_{i+1}(g_x)$, allora tutti i processi $q \in v_i(g_x)$ che hanno consegnato $v_{i+1}(g_x)$ hanno consegnato m prima di consegnare $v_{i+1}(g_x)$.

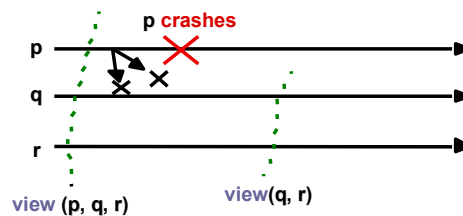
Primary Backup

35

View-Synchronous Multicast: Scenario a

- o Gruppo: p, q, r
- o p si guasta, q ed r sono corretti

a. OK

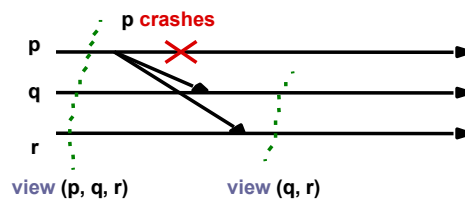


Primary Backup

36

View-Synchronous Multicast: Scenario b

b. OK



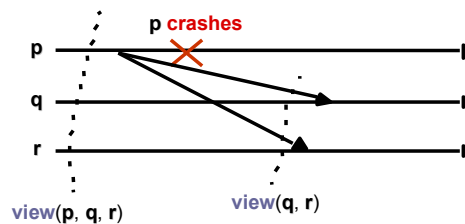
- o Almeno uno dei due processi q,r ha consegnato m quando p si guasta \Rightarrow q ed r prima consegnano m e poi la nuova view, view(q,r).

Primary Backup

37

View-Synchronous Multicast: Scenario c

c. NO



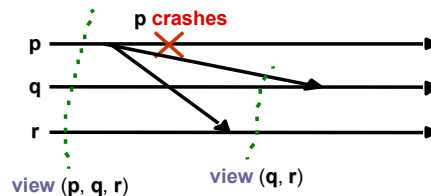
- o q ed r non possono consegnare un messaggio il cui mittente sanno essere guasto.

Primary Backup

38

View-Synchronous Multicast: Scenario d

d. NO.



- o I processi devono rispettare uno stesso ordinamento nella consegna dei messaggi e delle view.

Primary Backup

39

Active Replication

- o Non c'è un controllo centralizzato: tutte le repliche hanno uno stesso ruolo.
- o Ogni **replica** è **deterministica**: l'outcome di una operazione dipende soltanto dallo stato iniziale della replica e dalla sequenza di operazioni precedentemente eseguite dalla replica.

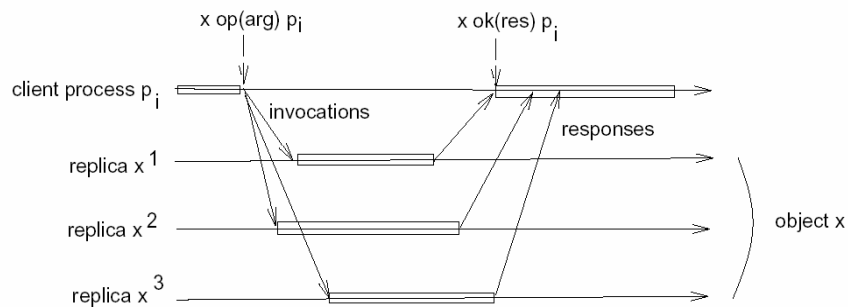
Quando un processo p_i invoca un'operazione sull'oggetto x :

- L'invocazione è spedita a tutte le repliche
- Ogni replica processa l'invocazione, aggiorna il suo stato e restituisce la risposta al client.
- Il client aspetta una sola risposta

Active Replication

40

Active Replication: Scenario di Funzionamento



Teoria della Replicazione

41

Active Replication: Garantire Linearizability

- Questa tecnica richiede:
 - **Atomicità**: se una replica gestisce un'invocazione allora ogni replica corretta la deve gestire a sua volta.
 - **Ordinamento**: date due invocazioni, se due repliche gestiscono entrambe, lo fanno secondo uno stesso ordinamento.
- Primitiva di comunicazione adeguata: **TOTAL ORDER MULTICAST**.

Active Replication

42

Active Replication: Crash

- L'active replication non richiede alcun tipo di azione nel caso di guasto di una replica.
- Si basa su **gruppi statici**:
 - la composizione del gruppo non cambia nel tempo,
 - la composizione non riflette i guasti: una replica guasta resta nel gruppo.

Active Replication: Ripristino di una replica

- Supponiamo che una replica x^k si guasti al tempo t e venga ripristinata all'istante t' ($t < t'$).
- Per l'**atomicità**, x^k dovrebbe aver consegnato tutti i messaggi spediti fino a t' .
- Quindi, quando x^k viene ripristinata, viene aggiornata da un'altra replica operativa x^j attraverso la procedura di **State Transfer**.

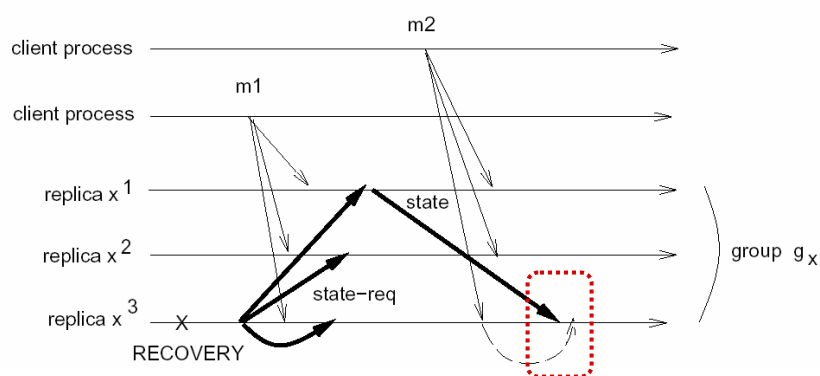
Active Replication: State Transfer

- x^k invia un messaggio di richiesta dello stato ($\text{state-req}, x^k$) attraverso una primitiva di total order multicast.
- Ogni replica dopo aver consegnato il messaggio ($\text{state-req}, x^k$) invia il suo stato a x^k .
- x^k aspetta di aver consegnato il suo messaggio di richiesta dello stato.
- x^k aspetta lo stato corrente da uno dei membri del gruppo g_x .
- Nel frattempo x^k bufferizza i messaggi ricevuti dopo aver inviato ($\text{state-req}, x^k$).
- Aggiorna il suo stato.
- Gestisce ipotetici messaggi bufferizzati.

Active Replication

45

State Transfer: Esempio



Active Replication

46

Confronto delle tecniche

	PRIMARY BACKUP	ACTIVE REPLICATION
Approccio	Centralizzato: Primary ha un ruolo di coordinatore	Completamente distribuito: Tutte le repliche hanno lo stesso ruolo
Repliche		deterministiche
Guasto di una replica	<ol style="list-style-type: none"> 1. Trasparente se la replica è un backup 2. Se la replica è il primary allora la latenza sperimentata dal client può essere eccessiva: non accettabile per applicazioni REAL-TIME 	un guasto di una replica è sempre trasparente al client
Uso di risorse	Minore dell'Active Replication	Maggiore del Primary Backup

47