ESERCITAZIONE 18/01/2023 + ALTRI ESAMI

Domanda 1 del 6/12/2021

a) **Spiegare l'approccio basato su shadow page table per la virtualizzazione della memoria, discutendone anche i possibili svantaggi o punti critici.**
b) **Con riferimento alla migrazione live delle macchine virtuali, spiegare gli approcci pre-copy, post-copy e ibrido per la migrazione live della memoria di una macchina virtuale.**
c) **In presenza di un carico applicativo di tipo read-intensive rispetto alla memoria, quale approccio tra quelli elencati al punto b) consente di ridurre il tempo totale di migrazione? E in presenza di un carico write-intensive? Motivare la risposta.**
d) **Descrivere cosa è l'immagine di un container Docker e spiegare la sua strutturazione in layer, discutendone anche i vantaggi. Considerando il seguente Dockerfile, spiegare lo scopo della corrispondente immagine di container, motivando opportunamente la risposta.**

```
FROM node:12-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "/app/src/index.js"]
```

Svolgimento

a) L'approccio basato su shadow page table prevede che venga mantenuta nell'unità di gestione di memoria, la quale sfrutta questa tabella per eseguire un mapping tra GVA – Guest Virtual Address e HPA – Host Physical Address, senza passare per Guest Physical Address (GPA). Permette di evitare un grado di indirezione, e la tabella viene modificata a seguito di una trap. La sua implementazione è complessa, richiede memoria e operazioni di mantenimento dei dati.
b) L'approccio **pre-copy** prevede di copiare in modo iterativo le pagine da VM sorgente e VM destinazione, mentre la sorgente è ancora in run. All'iterazione 'n', copio le pagine modificate durante 'n-1'. Spengo e poi copio cpu e driver.

   L'approccio **post-copy** di copiare cpu e device driver immediatamente, mentre le pagine di memoria vengo acquisite solo quando ce n'è bisogno. Ciò richiede che la VM sorgente rimanga attiva, perché devo usarla per prelevare pagine necessarie.
   Ciò comporta che, ad ogni pagina nuova non ancora copiata, si abbia un page fault. Infine il down time è ridotto.

   L'approccio ibrido prevede di copiare prima le pagine "Più usate", e solo dopo quelle meno richieste in modalità post-copy.
c) L'approccio pre-copy è ottimale in casi read-intensive (copio tutte le pagine), l'approccio post-copy è write-intensive (se scrivo di più rispetto alle letture, ho meno pagine da leggere e meno fault).
d) Brevemente, un container è un ambiente avente un proprio insieme di processi, file-system, utenti, ma sistema operativo condiviso. Docker usa una struttura a layer,

vantaggiosa perché i layer sono usabili per più container. Ogni layer è una istruzione, e ogni layer (tranne l'ultimo) è read-only. L'ultimo layer è **container layer** e viene aggiunto alla creazione del container, su cui riporto anche le modifiche al container. Non c'è persistenza (uso db o volumi). Il dockerfile ci dice che:
Dall'immagine del parent "alpine", operando nella workdir /app, copio la directory corrente in workdir, ed eseguo il comando di **RUN** yarn install in un nuovo layer, impostando con **CMD** i comandi default da eseguire all'avvio.

Domanda 1 del 19/1/2022

a) **Si definiscano la consistenza sequenziale e la consistenza causale.**
b) **Si illustrino le somiglianze e differenze tra i due modelli indicati al punto a) e si presenti un esempio di archivio di dati che soddisfa la consistenza causale ma non quella sequenziale.**
c) **Qual è il massimo grado di consistenza data centrica soddisfatto dall'archivio di dati sottostante? Si spieghi in particolare se soddisfa uno dei due tipi di consistenza definiti al punto a), motivando opportunamente la risposta.**

P1:	W(x)a	W(x)b	R(x)c	W(x)d
P2:	R(x)a	W(x)c	R(x)d	
P3:	R(x)b	R(x)b	R(x)c	

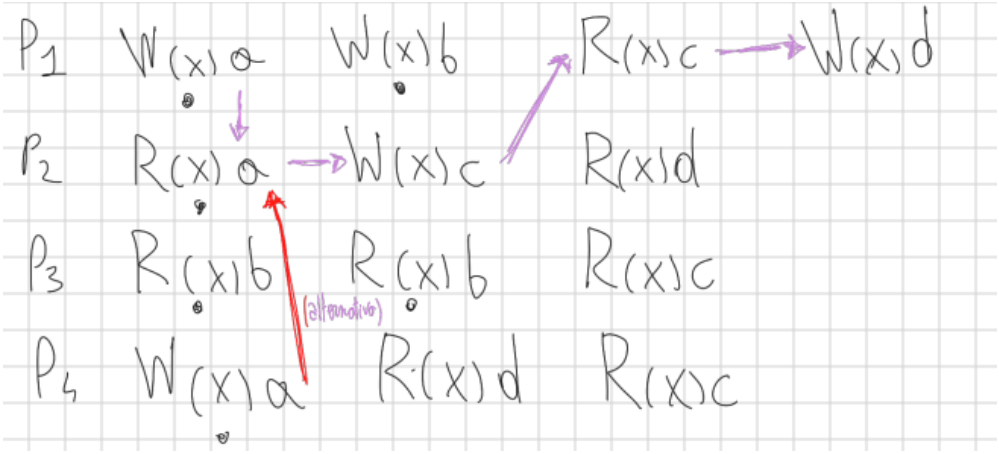
La risposta cambia ed eventualmente come aggiungendo il processo P4?

P1:	W(x)a	W(x)b	R(x)c	W(x)d
P2:	R(x)a	W(x)c	R(x)d	
P3:	R(x)b	R(x)b	R(x)c	
P4:	W(x)a	R(x)d	R(x)c	

d) **Tra i protocolli di consistenza di tipo replicated-write esaminati a lezione, se ne descriva uno a scelta in grado di garantire la consistenza sequenziale, discutendone anche vantaggi e possibili svantaggi**

Svolgimento

a) Nella consistenza **sequenziale** abbiamo l'illusione di un'unica copia e in unico ordine di programma (non per forza coincide con quello temporale).
Nella consistenza **causale** non abbiamo tale illusione, e l'ordinamento è limitato alle operazioni di causa-effetto.
b) Un esempio causale ma non sequenziale prevede che due Processi P1 P2 leggano due dati 'a' e 'b' in ordine inverso, cosicché non sia possibile trovare un ordine di programma.
c) C'è consistenza stretta? Le read devono tornare il valore della write più recente.
Nel processo P1 scrivo 'b' e leggo 'c', basta questo per violare. Non c'è.
C'è consistenza linearizzabile? Dovrebbe esistere unico interleaving, e ordinamento in base al timestamp.
C'è consistenza sequenziale? Devo trovare ordine di programma:
W(1,a) – R(2,a) – W(1,b) – R(3,b) – R(3,b) – W(2,c) – R(1,c) – R(3,c) – W(1,d) – R(2,d)
Di conseguenza valgono tutte le consistenze sottostanti, quindi anche la causale.
Aggiungiamo P4:



Soddisfa la sequenziale? (nb: non guardare le frecce viola, servono dopo), quindi
W(1,a) – R(2,a) – W(4,a) – W(1,b) – R(3,b) – R(3,b) ...
Non posso continuare! Mancano **c** e **d,** ma in P1 ho c,d e in P2 ho c,d. Non c'è sequenziale.
Ora sono "obbligato" a vedere le cause-effetto:
Se cerchiamo coppie causa-effetto: (nb: l'ordine è fondamentale, RW diverso da WR) le ho se:

- R-W su stesso processo (anche variabili diverse)
- W-R su processi diversi (stessa variabile)
- Transitività

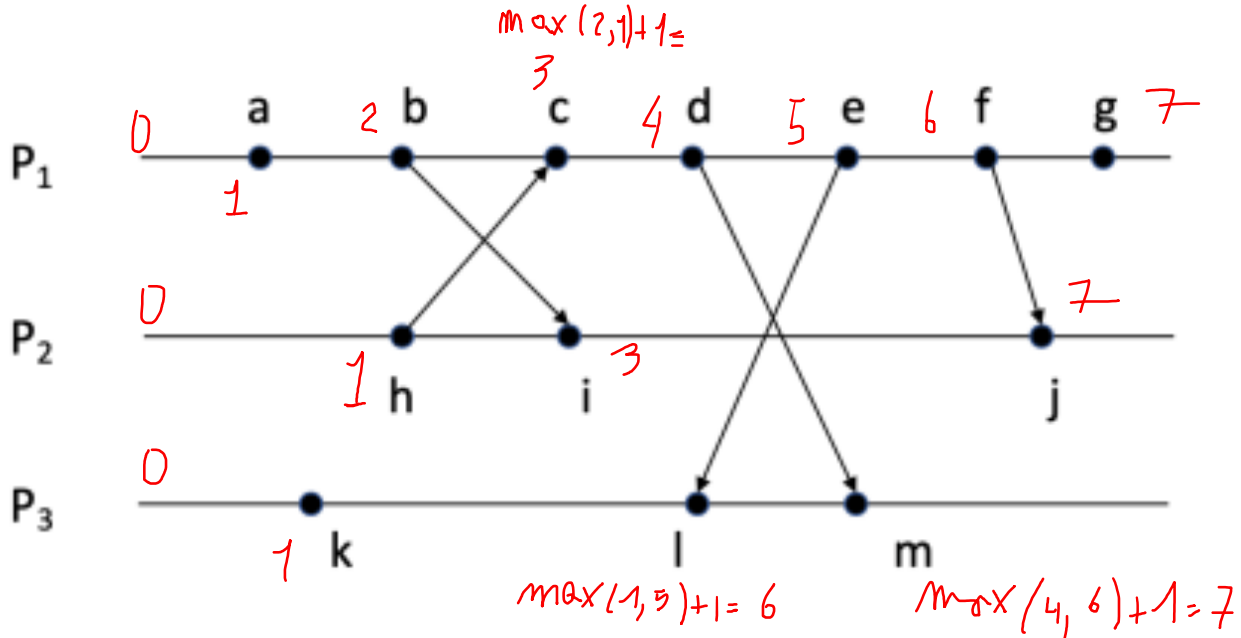
Evidenziamo la causalità **a** -> **c** -> **d**, ovvero, qualora ci siano questi dati, devono avere questo ordine (posso avere anche altro in mezzo, ma queste variabili sono ordinate tra loro).
In P4, abbiamo **a** -> **d** -> **c**, NON VA BENE. Se avessi invertito le due read di P4, andava bene.

Domanda 2 del 19/1/2022

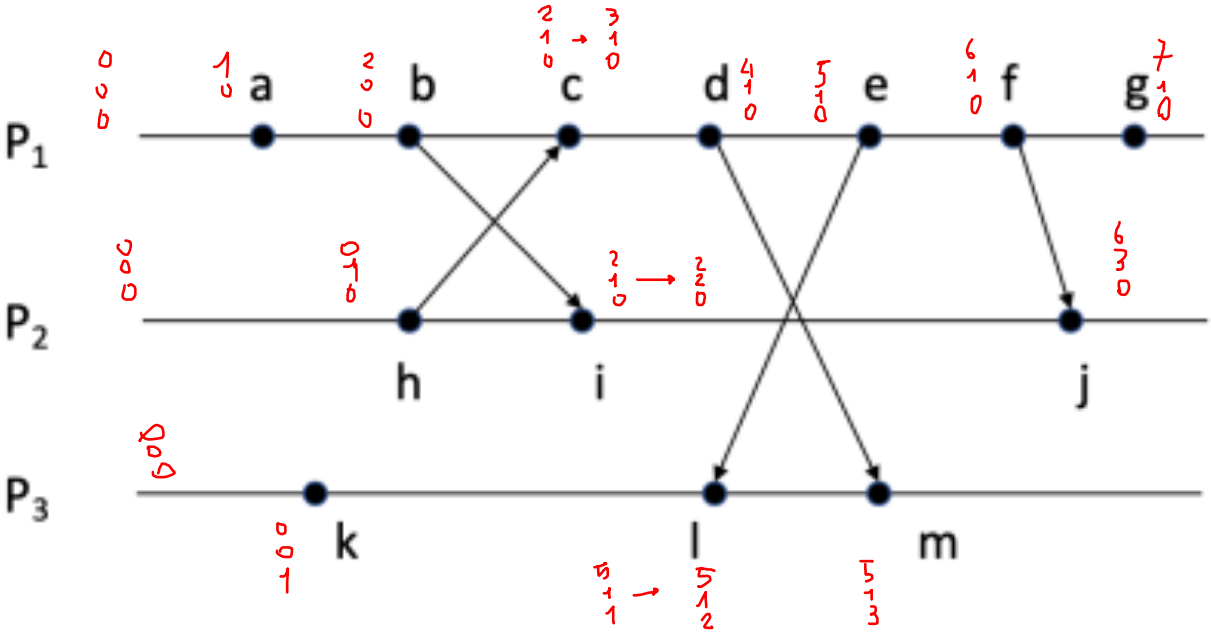
a) Esercizio sul clock di Cristian: per massimizzare l'accuratezza <= Tround/2 – min, dobbiamo prendere alfa il più piccolo possibile. Quindi cerco Tround più piccolo (min uguale per tutti i valori della tabella, poiché non specificato diversamente). La terza riga è da scegliere, RTT/2 = 10, il più piccolo, e il client setterà il suo clock al valore in tabella + 10 ms.

b) Limitazione: prendo il clock da un server.

c) Si determinino i valori del clock vettoriale di tutti gli eventi nel sottostante diagramma temporale. Quale conoscenza sullo stato del sistema rappresenta il valore del clock vettoriale associato all'evento m? Cosa rappresenta la somma dei valori di tale clock decrementata di 1.



Questo era caso scalare. **f** e **m** concorrenti, ma il vettore scalare non lo cattura. Introduciamo il vettoriale.



Attenzione: **m** è definito sapendo i vettori di **d** (freccia) e di **l** (stesso processo).
Servono entrambi. La somma degli elementi dei vettori indicano gli eventi avvenuti su sé stessi e conosciuti di altri (quelli che conosco, <= quelli effettivi).
La componente j di un vettore del processo j è informazione certa degli eventi avvenuti su di me.
L'uso del clock scalare porta una grana meno fine, poiché fa il "max" su unica componente, e quindi perdo info.
Un vettore di P3 [5 1 3] mi dice: sono a conoscenza di 5 eventi su P1, di 1 su P2, e sono certo che su di me sono occorsi 3 eventi. Su m, prima del decremento, ho [5 1 2], gli eventi occorsi.

d) i -> m?
Se prendiamo i vettori abbiamo: [2 2 0] e [5 1 3].
"i" viene prima di "m" almeno un componente di "i" è minore stretto del corrispettivo in "m", e tutti gli altri sono minori uguali. Qui non abbiamo ciò: 2<5, 2>1,0<3.
i -> j?
[2 2 0] vs [6 3 0]: $2 < 6$, $2 < 3$, $0 = 0$, quindi sì.
k -> d?
[0 0 1] vs [4 1 0]: 0<4, 0<1, 1>0, quindi no.
g -> h?
[7 1 0] vs [0 1 0]: 7>0, 1=1, 0=0, va bene; infatti, c'è un percorso da 'h' verso 'g' (seguendo le frecce!).

Nota: nel caso di vettori **scalari** non posso fare comparazione, perché L(e) < L(e') non implica che **e<e'**, ma posso solo dire che **e' non viene prima di e.**

Domanda 3 del 19/1/2022

a) **Si spieghi a cosa serve Kubernetes e si presenti la sua architettura distribuita.**
b) **In quali casi lo sviluppatore può scegliere di usare Docker Compose anziché Kubernetes? Con quali vantaggi e svantaggi?**
c) **Si presenti un pattern, a scelta tra quelli esaminati a lezione, che può essere usato nella realizzazione di un'applicazione a microservizi.**
d) **Si descrivano le caratteristiche del serverless computing e si discuta se è una soluzione adatta per realizzare applicazioni distribuiti scalabili.**

Svolgimento

a) Kubernetes è un tool che permette il deployment di una applicazione multi-container su molteplici host. La sua architettura è di tipo **master worker**, dove il master è **Kubernetes Control Plane**, i worker sono i nodi.
Il Control plane prevede:
+Kube-apiserver: Front end del control plane, espone le API.
+ Etcd: storage <key,value> distribuito.
+ Kube-scheduler: decide come posizionare i pod (cioè l'unità di schedulazione più piccola e computabile) nel nodo.
I nodi worker hanno:
+ Kubelet: si assicura che i pod siano in esecuzione
+Kube-proxy: gestisce il routing.Docker compose orchestra container su singolo host, mentre Kubernetes su molteplici host. La scelta dipende quindi dal numero di host con cui si vuole lavorare.

b) Docker Compose è uno strumento di Docker, e permette quindi al developer un uso semplificato, ma questo rende Docker Compose dipendente da Docker, mentre Kubernetes si presta a più casi di uso. DockerCompose non scala elasticamente, ma ha funzionalità di restart come Kubernetes, il quale non ha distribuzione geografica dei nodi, ovvero questo fattore non viene considerato nella scelta dello scheduling.

c) Abbiamo vari pattern nell'ambito dei microservizi: Circuit Breaker, Database per Service, Saga, Log Aggregation e Request Tracing.
Un pattern interessante è SAGA, che si pone di affrontare il problema della consistenza dei dati tra più servizi, definendo una sequenza di transizioni locale (Saga) in cui, a catena, una transazione locale aggiorna il rispetto database, e triggera la transazione locale successiva. Nel caso di un fail, si adotta una logica "all-or-nothing", facendo undo di tutto. Implementabile sia tramite orchestrazione che coreografia.

d) Il serverless computing è un modello di cloud computing il developer può testare le proprie funzioni, senza preoccuparsi dell'infrastruttura sottostante e di aspetti come l'elasticità. Le risorse sono effimere, allocate quando necessarie, e basate su event-driven. L'approccio è pay-per-use. Anche se il costo per il singolo uso del modello è irrisorio e teoricamente potrebbe essere usata in un ambito di applicazioni distribuite scalabili, l'approccio serverless, in ambiente distribuito, non è una valida alternativa, sia in termini di costi che di flessibilità.

Domanda 4 del 19/1/2022

a) **Si presenti un algoritmo di mutua esclusione distribuita per un sistema distribuito a scelta tra quelli esaminati a lezione. Nell'algoritmo considerato, cosa accade quando due processi richiedono in modo concorrente l'accesso in sezione critica?**
b) **Si discuta quali problemi possono insorgere durante l'esecuzione dell'algoritmo scelto al punto a), presentandone, se esiste, una possibile soluzione.**
c) **Si descriva un algoritmo di consenso distribuito a scelta tra Paxos e Raft e si discuta come l'algoritmo gestisce la tolleranza a guasti di tipo non bizantino.**
d) **Con riferimento all'algoritmo di consenso Raft, si spieghi se può essere utilizzato come algoritmo di mutua esclusione distribuita e, in caso affermativo, con quali vantaggi rispetto all'algoritmo scelto al punto a)**

Svolgimento

a) Algoritmi di mutua esclusione sono basati su **autorizzazioni (Lamport e Ricart-Agrawala**), **Token** e **Quorum (Maekawa).**
Per completezza, diamo una rapida occhiata a tutti.
Lamport distribuito: ogni processo ha clock logico scalare (per la relazione di ordine) e coda locale per memorizzare richieste quando sto in sezione critica.
Sostanzialmente un processo **i**, per entrare in sezione critica, invia un messaggio a tutti i processi (anche a sè stesso), che la memorizzano in coda. Tutti questi processi possono rispondere con un ack (a meno che loro stessi non siano in sezione critica). Il processo **i** entra in sezione critica se il suo messaggio è in testa alla coda (quindi timestamp minore) e ha ricevuto da tutti gli altri processi un messaggio di release (il processo **i**, per entrare, deve essere sicuro che tocca a lui, e può esserlo solo se ha visto tutti i messaggi degli altri e quindi ha sicurezza di essere il prossimo). Quando processo esce, invia msg di release a tutti, che cancellano msg dalla sua coda.
Se c'è un contenzioso per l'accesso in sezione critica:
P2 vuole accedere, manda i msg a tutti, incluso P1. Se P1 vuole accedere, confronta grazie al clock il messaggio con timestamp minore: se è quello di P2, P1 manda il suo ack, altrimenti non risponde e mette in coda. Se processo si rompe, rip.
Soddisfa Safety, Liveness e Fairness.

Ricart-Agrawala:
Come Lamport, ma ha una differenza: **non prevede alcun messaggio di release**.

Token decentralizzato
Anello unidirezionale, il token viaggia tra processo e processo. Solo chi ha il token può entrare in sezione critica, altrimenti passa il token. Carico bilanciato (non c'è gestore del token) e fairness (tutti possono entrare in sezione critica).
Problema: trasmissione del token, e se ho errori sul token può essere molto complicata la sua gestione, devo essere sicuro che non venga duplicato.

Maekawa-quorum

Basato su insieme di votazione Vi. Un elemento di Vi invia richiesta ai membri di Vi per accedere in CS, e attende reply. Solo così può entrare, e lasciare un msg di release quando esce. D'altro canto, se un processo di Vi è già in CS o ha già risposto dopo l'ultima release, non risponde e accoda la richiesta. Quando si riceve una release, estraggo dalla coda e invio reply. Abbiamo regole precise su come comporre i gruppi (ad esempio due gruppi devono avere un elemento in comune per garantire mutua esclusione, o anche che ogni processo deve apparire lo stesso numero di volte in insiemi di votazione). Il vantaggio risiede nei pochi messaggi, ma un grande svantaggio è che posso avere deadlock: se due processi si contendono l'accesso e sono in stesso insieme di votazione: voteranno per sé stessi, e quindi deadlock. Posso risolvere con messaggi aggiuntivi.

b) Insieme al punto a.

c) Esaminiamoli tutti e due, in quanto non li ho mai visti.

Paxos: N processi eseguono l'algoritmo, Settando N >= 2k+1 posso gestire fino a k processi guasti. Infatti, se N = 11, allora fino a 5 fault va bene, perché ho 6 processi > 5 fault che quindi hanno la maggioranza. Ha molteplici implementazioni. Sfrutta assunzioni realistiche e deboli, come sistemi **parzialmente sincroni**, posso avere **ritardi**, msg persi etc, velocità variabile, processi non devono "barare". E' di tipo quorum, e devo raggiungere il consenso su **un solo valore.**

Sostanzialmente ci sono tre ruoli: **Proposer**, che propone un valore per conto del client, **Acceptor** che vota quale scegliere, e **Learner**, che "impara" questo valore dopo che è stato effettivamente scelto (non prima!) e riporta la decisione al client.

Paxos opera in più round, ognuno avente **fasi di prepare** e **accept**.

Esempio: 3 processi, 3 acceptor (gruppi A1,A2 per P1 e A2,A3 per P2).

Nel primo round:

- Prepare: mando solo il numero di proposta "1", non il valore. I due acceptor accettano, "promettendogli di non accettare richieste con numero proposta <1". Nei round successivi dovranno aggiungere anche il valore associato, che ancora non c'è. P2 manda prepare con numero "2", che anche qui fanno stessa promessa. Poiché A2 è in comune a entrambi, allora egli non accetterà più le proposte di P1, per via di questa nuova promessa.
- Accept: P1 manda accept <n proposta, valore proposto1). Solo A1 accetta. P2 fa lo stesso, ma a lui accettano entrambi. A2,A3 maggioranza nei 3 acceptor, quindi vince P2 con il suo valore proposto "2".. A1 ha ultima richiesta accettata "1", A2 e A3 ultima richiesta "2".

Nei round successivi?

- Prepare: p1 invia request con id = 3 ad A1, A2. I due acceptor promettono di rifiutare proposte sotto a id = 3, inviando anche il precedente valore accettato, quindi A1 gli invia <1,valore1> e A2 <2,valore2>. P1 proporrà quindi ad A1 e A2 il messaggio <2,valore2> poiché avente id più alto.
- Accept: p1 invia accept <3, valore2>, gli acceptor accettano.

La versione studiata di Paxos non gestisce guasti bizantini, ma esiste apposita versione che lo fa.

Raft: L'idea alla base è quella di decomporre ciascun problema in sottoproblemi indipendenti. E' implementato su cluster di server, ciascuno con macchina a stati, log

(input per la VM) e protocollo Raft. C'è un leader e vari follower. L'obiettivo è mantenere la consistenza della macchina a stati replicata su vari server. C'è anche il ruolo di candidate nel caso ci sia un crash di un leader. Il leader manda periodici heartbeat, se viene a mancare per un follower, si indice una votazione in cui ci si propone, e si richiede voti agli altri. Altri follower votano se non hanno già votato (magari follower1 è il primo a chiedere la votazione, e poi la nota follower2). Potrebbe però verificarsi una elezione in contemporanea, per questo si può introdurre un timeout ulteriore nella votazione o randomizzare leggermente il periodo di mancanza di heartbeat (quindi per P1 c'è un problema se non ho heartbeat tra 3 secondi, per P2 se non li ho tra 3.1). Solo il leader riceve comandi dal client, e fa append sul log, che viene distribuito ai follower, i quali confermano. Quando riceve maggioranza delle conferme, la append sul log è considerata "committed", inviando questa conferma ai follower, che la eseguono a loro volta. (Quindi è assai improbabile non avere consistenza, perché se un follower perde il primo msg, dovrebbe perdere anche il secondo!). Non è tollerante ai guasti bizantini, perché mi devo fidare del leader.

Domanda 1 del 5/2/2020

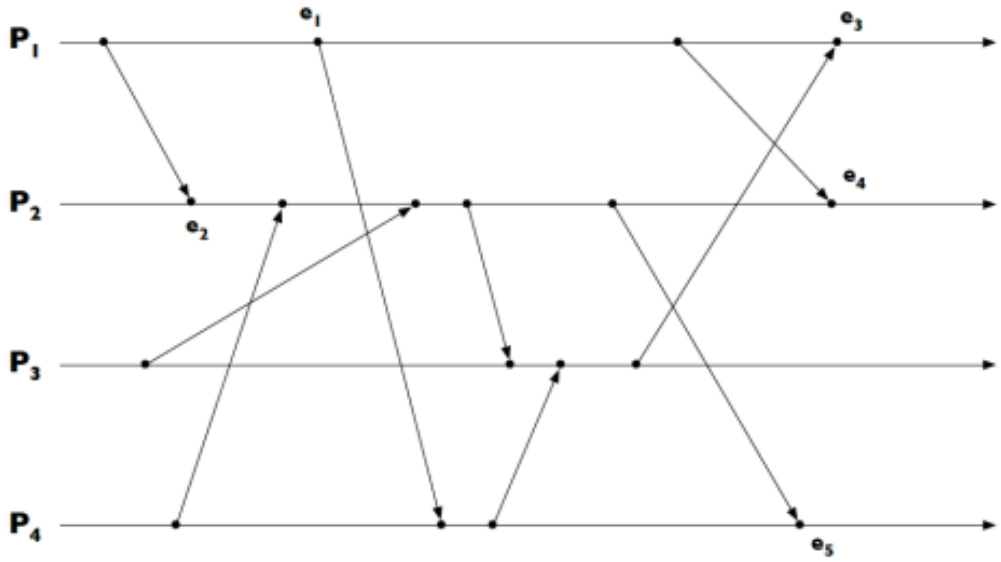
a) **Si spieghi il significato delle componenti del clock logico vettoriale di un processo Pi e si descriva il protocollo di aggiornamento del clock logico vettoriale.**

b) **b) Si spieghi qual è l'obiettivo del multicasting causalmente ordinato e perché si applica il clock logico vettoriale in questo algoritmo e non quello scalare.**

c) **Si determinino i valori del clock scalare e del clock vettoriale di tutti gli eventi nel sottostante diagramma temporale.**

d) **Si discuta se, dato il valore del clock sia scalare che vettoriale determinato al punto c), quali delle seguenti affermazioni è falsa, motivando opportunamente la risposta:**

1. $e_1 \rightarrow e_3$
2. $e_1 \rightarrow e_5$
3. $e_2 \rightarrow e_3$
4. $e_1 \rightarrow e_4$

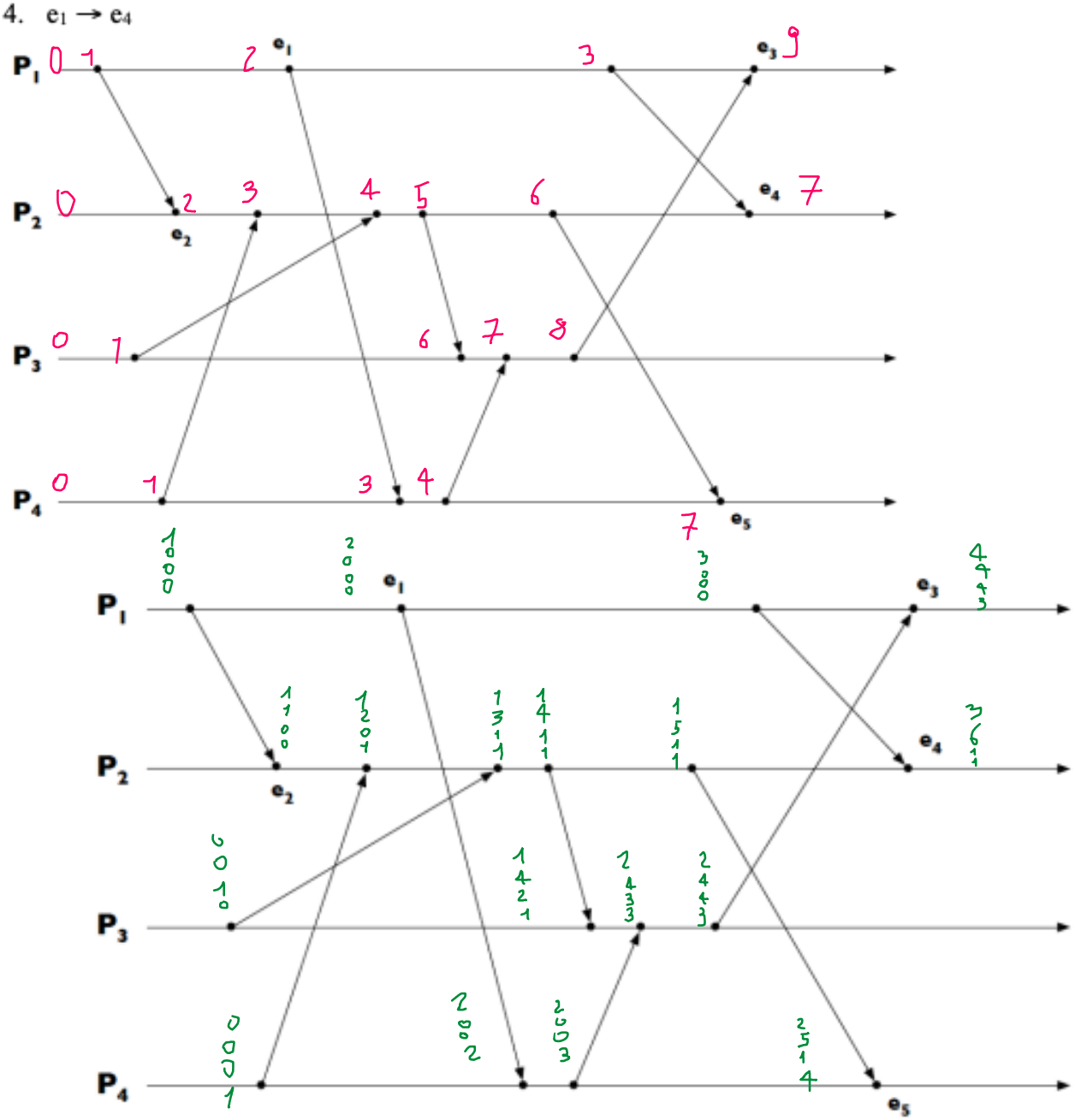


SVOLGIMENTO

a) Dati N processi, ogni processo ha un vettore pari ad N elementi.
Per il processo "i" identifichiamo l'elemento i-esimo del vettore, ed altri elementi j != i. L'elemento i-esimo rappresenta gli eventi cerrti avvenuti sul processo i, sono certi perché il processo "i" sa cosa avviene su sé stesso. Gli altri elementi j-esimi rappresentano gli eventi occorsi sul processo j di cui "i" ha conoscenza, che non corrispondono per forza agli eventi effettivamente avvenuti su j.
Se P1 [1 2] vuol dire che su P1 è avvenuto un evento certo, mentre P1 sa che P2 ha visto almeno due eventi, ma potrebbero essere di più.
L'aggiornamento del clock vettoriale avviene secondo due condizioni:
- Pi ha un evento (interno/esterno), questo comporta l'incremento di 1 della componente i-esima del vettore.
- Pi riceve un messaggio da Pj, allora Pi aggiorna il suo vettore in questo modo: il vettore Pi sarà il massimo, per ogni componente, tra il vettore di Pj e il precedente vettore di Pi, cioè il suo timestamp.

b) L'obiettivo del multicast è garantire che scritture concorrenti su un db replicato siano viste nello stesso ordine da ogni replica. Abbiamo due approcci (la domanda ne chiede uno, ma per un ripasso li espongo entrambi):
    a. **Multicast totalmente ordinato**: voglio avere medesimo ordine su tutte le repliche, non deve per forza essere quello corretto.
    Come suggerisce il nome, dovremo inviare messaggi a tutti i processi.
    Supponendo FIFO e comunicazione affidabile abbiamo due algoritmi:
        a. **Centralizzato**:
        C'è un sequencer, che riceve update dai processi, li ordina assegnando un sequence number e poi li invia ordinati a tutti i processi. Problemi di scalabilità e SPOF.
        b. **Distribuito:**
        Si usa clock logico scalare per ogni messaggio. Un processo Pi invia msg a tutti (incluso sé stesso), ciascun processo lo ordina secondo timestamp e invia in multicast un ack di conferma. Pi inoltra all'applicazione il messaggio se tutti hanno inviato un loro update, e il messaggio di Pi risulta avere timestamp minore di tutti. (Molto simile a Lamport distribuito). Molti messaggi.
    b. **Multicast casualmente ordinato:** non voglio avere ordine di tutto, ma solo sulla relazione causa-effetto, quindi ipotesi più debole rispetto a quella proposta prima. Qui usiamo un clock logico vettoriale, con una piccola differenza: Per un processo "i", la componente "j" indica i messaggi scambiati tra i e j, non quelli che i ha osservato di j. La novità è che Pj ritarda la consegna del messaggio di Pi finché non sono entrambe vere le condizioni:
        a. t(m)[i] = Vj[i] + 1, cioè se P[j] ha ricevuto ad esempio 2 messaggi da Pi, e quindi ha Vj[i] = 2, il prossimo deve essere 3, non può essere 4.
        b. Per tutte le altre componenti, il clock di Pi deve essere <= delle componenti di Pj. Stiamo dicendo che, se Pi invia a Pj, Pi non può aver più scambiato messaggi con altri processi rispetto Pj.

1. $e_1 \rightarrow e_3$
2. $e_1 \rightarrow e_5$
3. $e_2 \rightarrow e_3$
4. $e_1 \rightarrow e_4$



Nel caso scalare non posso dire nulla: L(e) < L(e') non implica e<e', ma solo che e' non viene prima di e.
Vediamo le altre relazioni nel clock vettoriale:

- e1 -> e3: [2 0 0 0] vs [4 4 4 3] 2<4, 0<4, 0<4, 0<3 si, infatti avvengono su P1, quindi effettivamente l'operazione è corretta.
- e1 -> e5: [2 0 0 0] vs [2 5 1 4]: 2 = 2, 0<5, 0<1, 0<4 lo è, infatti possiamo disegnare un percorso da e1 a e5.
- e2 -> e3: [1 1 0 0] vs [4 4 4 3], anche qui ok, infatti c'è percorso.
- e1 -> e4: c'è percorso, vediamo la conferma.
  [3 0 0 0] vs [3 6 1 1]: ok anche qui, c'è percorso.

Domanda 2 del 5/2/2020

a) Si definiscano la consistenza linearizzabile e la consistenza causale e se ne evidenzino le differenze.

b) Qual è il massimo grado di consistenza data centrica soddisfatto dall'archivio di dati sottostante?

P1:	W(y)0	W(x)1	R(x)1	R(y)0
P2:		W(y)1	R(y)1	R(x)1
P3:		R(x)1	R(y)0	
P4:	W(x)0	R(y)0	R(x)0	

c) Qual è il massimo grado di consistenza data centrica soddisfatto dall'archivio di dati sottostante? Motivare la risposta.

P1:	W(x)0	W(x)1	R(x)2	W(x)3
P2:	R(x)0	W(x)2	R(x)3	
P3:	R(x)1	R(x)1	R(x)2	

SVOLGIMENTO

a) Nella consistenza l'obiettivo è che tutte i processi vedano lo stesso ordine di operazioni. Ciò che cambia è l'interpretazione dell'ordine. Nella consistenza linearizzabile necessitiamo di un timestamp globale (dato da un clock sincronizzato), e abbiamo tale tipo di consistenza solo in presenza di un unico interleaving, ovvero tutte le operazioni appaiono nello stesso ordine per tutti i processi. Unica eccezione è la possibilità di avere operazioni sovrapposte (cioè stesso timestamp). In questo caso abbiamo flessibilità su quale operazione mettere per prima, ma tutti i processi dovranno vedere l'ordine scelto. Nella consistenza causale rilassiamo queste ipotesi, e diciamo che questo tipo di consistenza è realizzata se rispettiamo l'ordine per le relazioni di causa-effetto. (R-W su stesso processo, W-R stesso dato su processi diversi, transitività). Non c'è più illusione singola copia.

b) Non c'è consistenza linearizzabile, in quanto non riusciamo a trovare unico interleaving. (Nella colonna 3 P3 e P4 leggo x = 0 e y = 0, ma sono state sovrascritte nella colonna 2 da x = 1 e y = 1).
Consistenza sequenziale:
w1(y,0) w4(x,0) r4(y,0) r4(x,0) w1(x,1) r1(x,1) r3(x,1) r1(y,0) r3(y,0) w2(y,1) r2(y,1) r2(x,1)

c) w1(x,0) r2(x,0) w1(x,1) r3(x,1) r3(x,1) w2(x,2) r1(x,2) r3(x,2) w1(x,3) r2(x,2)

Domanda 3 del 5/2/2020

a) Si descriva un algoritmo di elezione a scelta tra quelli esaminati a lezione, indicando anche quali sono le assunzioni sul modello del sistema.

b) Si discuta cosa accade se durante l'elezione condotta in base all'algoritmo descritto al punto a):
- due processi partecipanti avviano contemporaneamente l'elezione;
- uno dei processi partecipanti subisce un crash;
- avviene una partizione di rete.

c) Perché l'algoritmo di Paxos può essere usato per l'elezione di un leader in un sistema distribuito (ad es. in Zookeeper)? Quali eventuali vantaggi presenta rispetto all'algoritmo descritto al punto a)?
d) Si può usare l'algoritmo dei generali bizantini per l'elezione di un leader ed eventualmente con quali vantaggi e svantaggi rispetto all'algoritmo descritto al punto a)?

Svolgimento

a) Due principali algoritmi di elezione sono: **Bully** e **Fredkinson/Lynch**
Entrambi richiedono: comunicazione affidabile, un'elezione per volta e un id univoco, eleggendo il processo con id più alto non guasto.
a) Bully
In questo algoritmo, quando un processo "i" nota l'assenza del coordinatore, indice una nuova elezione, inviando questo messaggio a tutti i processi aventi id superiori.
Se nessuno risponde, "i" è il vincitore, e manda un messaggio a tutti i processi con id inferiore, comunicando la vincita.
Se un processo con id j>i riceve il messaggio di elezione, e risponde con "OK", allora "i" non è più candidabile, e "j" indice una nuova elezione con i processi di id superiore al suo (nello stesso modo descritto finora). Viene inoltre richiesto che il sistema sia sincrono, perché devo aspettare l'arrivo dei messaggi.
b) - Se due processi avviano elezione in contemporanea, il processo avente id superiore "bloccherà" l'altro.
- Se un processo subisce un crash, e questo processo era leader, si indice una nuova elezione.
- Se avviene una partizione di rete, i processi continuano a funzionare internamente alla rete, non all'esterno.
c) L'algoritmo di Paxos può essere usato per l'elezione di un leader in quanto tale algoritmo ha l'obiettivo di far raggiungere un consenso, in un sistema di N = 2k+1 processi.Nulla vieta che tale consenso possa essere riferito all'elezione di un leader.
I vantaggi di tale applicazione risiedono nelle assunzioni fatti alla base del suo funzionamento, che sono più rilassate rispetto Bully. In particolare, il sistema deve essere parzialmente sincrono e può tollerare fino a k guasti non bizantini. Esiste anche una particolare implementazione di Paxos per affrontare il caso dei guasti bizantini. Di contro, la sua implementazione, anche nella versione base, risulta abbastanza complessa.
d) non trattato.

Domanda 4 del 5/2/2020

a) Si presenti un algoritmo di mutua esclusione distribuita a scelta tra Ricart-Agrawala e Maekawa.
b) Quali sono le differenze tra i due algoritmi in termini di distribuzione delle decisioni, prestazioni, safety, liveness e ordinamento delle richieste di accesso in sezione critica?
c) Si può usare l'algoritmo di Raft per la mutua esclusione distribuita ed eventualmente con quali vantaggi e svantaggi rispetto all'algoritmo considerato al punto a)?

Svolgimento

a) Esaminiamo Ricart-Agrawala, una ottimizzazione di Lamport distribuito (in termini di messaggi scambiati). Dato un processo Pi, che vuole entrare in sezione critica, esso manda un messaggio di richiesta a **tutti** i processi, incluso sé stesso, includendo il proprio <id,timestamp scalare>.
   Attenzione: Lamport usava solo timestamp, non id!
   Un processo Pj che non è intenzionato ad entrare in sezione critica, invia a Pi un reply, quindi una "conferma da parte sua per accedere in CS". Se Pj è in sezione critica, accoda il messaggio, e lo gestirà successivamente.
   Se Pj vuole anch'egli entrare in CS, confronterà il suo timestamp con quello di Pi, se vince Pi, allora Pj manderà il suo reply, se vince Pj non risponde accoda il messaggio.
   Pi può entrare in sezione critica se ha ricevuto reply da tutti.
b) Ricordiamo che Maekawa è basato su quorum, in cui si organizzano dei gruppi con particolari condizioni.
   Safety: Solo un processo entra in sezione critica. Sia Agrawala che Maekawa lo fanno.
   Liveness: Non devo bloccarmi (no deadlock). Agrawala non si blocca (discrimino sempre chi deve entrare), Maekawa si (ognuno vota sé stesso).
   In termini di "distribuzione", con Maekawa il consenso avviene nell'insieme di votazione Vi, mentre in Agrawala c'è uno scambio di messaggio con tutti i processi in esame.
   In termini di "ordinamento delle richieste in sezione critica", Agrawala si basa su timestamp e code, mentre Maekawa usa "id" e code.
c) Raft, come Paxos, è un algoritmo di consenso, ma nulla viete che tale consenso sia relativo ad una elezione. Qui abbiamo cluster di server, in ognuno c'è una macchina a stati, l'algoritmo di Paxos e un log. C'è leader che guida i follower, con una variabile locale che creste nel tempo, e il leader invia heartbeat. I vantaggi di questa implementazione risiedono nelle assunzioni rilassate (sistemi parzialmente asincroni, posso avere perdite di messaggi etc), si ha tolleranza alle partizioni di rete, ma di contro ho un sistema basato su leader-follower e progettato per consistenza.

Domanda 1 del 14/01/2019

a) Si definisca la consistenza sequenziale e la consistenza causale e se ne evidenzino le differenze.

b) L'archivio di dati sottostante soddisfa la consistenza sequenziale? E la consistenza causale? Motivare la risposta

P1:	W(x)1	R(x)1	R(y)0
P2:	W(y)1	R(y)1	R(x)1
P3:	R(x)1	R(y)0	
P4:	R(y)0	R(x)0	

c) L'archivio di dati sottostante soddisfa la consistenza sequenziale? E la consistenza causale? Motivare la risposta.

P1:		W(x)1				
P2:			W(x)2			
P3:	R(x)2		R(x)1		W(y)3	
P4:				R(x)2	W(x)1	R(y)3

d) Con riferimento ai protocolli di consistenza quorum-based, si indichi quali modelli di consistenza sono garantiti dalle seguenti configurazioni, motivando la risposta.

$N_R=N_W=N$

$N_R=N_W=N/2+1$

SVOLGIMENTO

a) Nell'ambito della consistenza, parliamo di un contratto tra datastore e processi, in cui vorremmo che risultati di lettura e scrittura si verifichino secondo un determinato ordine. La definizione di "ordine" dipende dalle ipotesi e dal tipo di consistenza messa in atto. Nella consistenza sequenziale l'ordine da rispettare è l'ordine di programma, possiamo avere vari interleaving e abbiamo l'illusione di una singola copia.
Nella causale perdiamo l'illusione della singola copia, e ordiniamo solo su relazione causa-effetto.

b) Consistenza sequenziale:
r4(y,0) r4(x,0) w1(x,1) r1(x,1) r3(x,1) r3(y,0) r1(y,0) w1(y,1) r1(y,1) r1(x,1)

c) Consistenza sequenziale:
w2(x,2) – r3(x,2) – r4(x,2) – w1(x,1) – w4(x,1) – w3(y,3) - r4(y,3)

d) Nei quorum-based, per fare read e write, servono quorum. Sia Nr il quorum per la lettura e Nw quello per la scrittura:
Se voglio impedire conflitti r-w -> $Nr + Nw > N$
Se voglio impedire conflitti w-w -> $Nw > N/2$
Se le ho entrambe -> consistenza sequenziale.
Se $Nr = Nw = N$ allora $Nr+Nw = 2N > N$ e $Nw=N>N/2$ -> consistenza sequenziale.
Se $Nr = Nw = N/2 +1$ allora $Nr+Nw = N + 2 > N$, ho anche $Nw = N/2 + 1 > N/2$, consistenza sequenziale.

Domanda 2 del 14/01/2019

a) Si spieghi il teorema CAP e si discutano le implicazioni che comporta nella progettazione di un sistema distribuito a larga scala.

b) Si fornisca un esempio di un servizio, software o piattaforma per la memorizzazione di dati per almeno due combinazioni di C, A e P (i.e., CA, AP, CP). Si discuta brevemente come l'esempio considerato supporta le due proprietà e non la terza.

c) Si supponga che un'università di grandi dimensioni intenda progettare alcuni servizi online, elencati di seguito, per i propri studenti e docenti. In ciascun caso, si discuta quali proprietà del teorema CAP il servizio dovrebbe supportare, motivando opportunamente la risposta.
   - Servizio di editing collaborativo di documenti;
   - Servizio di notifica di situazioni di emergenza;
   - Servizio di discussione per i corsi erogati;
   - Servizio di iscrizione per gli studenti;
   - Servizio di hosting dei video delle lezioni.

Svolgimento

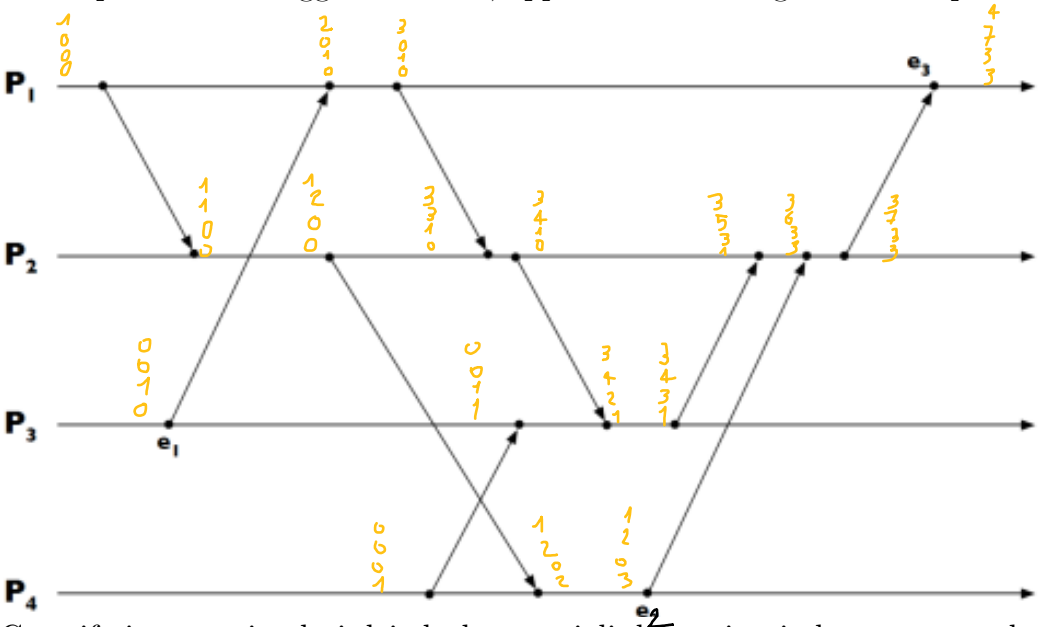
a) Nei vari algoritmi esaminati nell'ambito dei sistemi distribuiti, ci siamo resi conto come il partizionamento della rete possa essere un grave problema da gestire in larga scala. Nella consistenza finale (il grado più basso) rendiamo copie subito disponibili, con la convinzione che prima o poi "convergeranno" al valore corretto. Tale analisi è alla base del teorema CAP, che ci dice che in qualunque data-store possiamo scegliere solo due aspetti dei tre seguenti: partizionamento rete, consistenza, disponibilità. Normalmente, il partizionamento è sempre considerato, e per questo si sceglie tra consistenza e disponibilità. A seconda del servizio erogato, un approccio CP può essere ideale rispetto un AP, e viceversa.

b) AP: non ho consistenza, ma ho continua disponibilità e tolleranza al partizionamento: un esempio è un archivio di file online condiviso, e due utenti in Europa e America. In questo contesto, tollero la partizione tra i due server, i file sono disponibili sempre, ma se un utente facesse una modifica, questa potrebbe non essere subito disponibile.
CP: Prenotazione di un concerto di una star mondiale. Ho partizionamento perché i biglietti possono essere presi in tutto il mondo tramite replicazione server, devo avere consistenza per evitare "biglietti doppi" effettivamente acquistati, ma se non fossi abbastanza veloce potrei vedere un biglietto disponibile quando non lo è.
1) Editing: CA, tutte le facoltà devono accedervi (P), e se collaborano devono avervi accesso (A).
2) Emergenza: AP, tutte le facoltà devono accedervi, (P), tutti devono averle disponibili (A), anche se questo potrebbe creare dei falsi positivi, ma se fosse CP potrei non vedere proprio l'allarme.
3) Corsi: AC, ogni facoltà ha i propri corsi, non c'è bisogno di interazione.
4) Iscrizione: CP, tutti devono accedervi(P), e tutti devono completare l'iscrizione (C).
5) Hosting: AC, perché ogni facoltà ha i propri corsi.

c) non trattato nel dettaglio.

Domanda 4 del 14/01/2019

a) Si presenti un algoritmo di mutua esclusione distribuita a scelta tra Ricart-Agrawala, Maekawa e basato su token centralizzato.
b) Quali sono le differenze tra i tre algoritmi in termini di: distribuzione dell'algoritmo, prestazioni e liveness?
c) Si spieghi come si rappresenta il clock logico vettoriale e si descriva il relativo protocollo di aggiornamento, applicandolo al diagramma temporale sottostante.



Con riferimento ai valori dei clock vettoriali determinati al punto precedente, si discuta quale delle seguenti affermazioni è falsa e perché

1. $e_1 \rightarrow e_2$
2. $e_1 \rightarrow e_3$
3. $e_2 \rightarrow e_3$

Svolgimento

a) Maekawa è un algoritmo basato su quorum. Particolare attenzione è data alla creazione dei gruppi di votazione Vi; infatti, ogni gruppo di votazione deve avere almeno un elemento in comune (per garantire la safety), e tutti i processi hanno stessa responsabilità (fanno parte dello stesso numero di insiemi di votazione).
In questo algoritmo, un processo che vuole entrare in CS invia un msg ai membri del gruppo, attende le repliche, e se le ha tutte (del suo gruppo) può accedere in CS.
Un Processo Pj che riceve questa richiesta, se non è interessato alla CS o non ha mai votato rilascia subito il suo reply, altrimenti se è in CS o ha già votato accoda la richiesta, per poi esaudirla successivamente. Quando processo esce dalla CS, estrare una richiesta dalla coda e invia reply.
b) Ricart-Agrawala garantisce liveness (no deadlock), prestazioni migliori di Lamport distribuito ma peggiori di Maekawa, poiché scambia messaggi con tutti gli altri processi. Maekawa non garantisce liveness (se ogni processo vota sé stesso c'è un deadlock), richiede comunicazione solo con l'insieme di votazione, e questo porta a un invio minore di messaggi.
c) Nel clock logico vettoriale, per ogni processo definiamo un vettore di N elementi (dove N è il numero di processi in esame). Per un processo "i", la componente i-esima sono gli

eventi occorsi sul processo "i", e sono eventi certi proprio perché avvenuti su sé stessi. Le altre componenti j!=i rappresentano gli eventi di j di cui "i" è a conoscenza, e non necessariamente coincidono con quelli effettivamente avvenuti.

d) Vedendo la figura mi aspetto che sia la seconda successione a non valere, perché non esiste percorso da e1 a e3. Vediamo:
e1 vs e2: [0 0 1 0] vs [1 2 0 3], come vediamo, in e1 solo il terzo elemento è maggiore di quello in e2, mentre gli altri sono maggiori in e2, quindi è lei.

DOMANDE VIRTUALIZZAZIONE SPARSE

a) Caratteristiche virtualizzazione s.o, realizzazione e isolamento container?

Con la virtualizzazione a livello di sistema operativo, l'obiettivo è creare molteplici ambienti isolati all'interno di un unico sistema operativo. Tali ambienti sono i **container**. Ogni container ha propri processi, nomi, utenti, tabelle, regole etc.. ma tutti condividono lo stesso **sistema operativo**.
Esistono vari approcci per creare un container, tra i quali **chroot** (cambio della root directory per il processo in esecuzione) **Namespaces** (isolo la vista di un insieme di processi) o **Cgroups** (isolo, limito le risorse per un insieme di processi).
Rispetto ad una virtualizzazione di sistema, i container sono più leggeri, invocano le system call senza indirezione, sono facili da migrare, e possono condividere pagine di memoria. Tuttavia, mi limitano all'uso di un singolo s.o (e quindi kernel), e le applicazioni devono essere native per il mio s.o, oltre ad un minor isolamento. La superficie di attacco è il s.o. guest, più ampia rispetto al vmm.

b) Rispetto all'unikernel, quali vantaggi e svantaggi presentano i container?

L'uso degli unikernel comporta una velocità di utilizzo maggiore, poiché il kernel è di dimensione ridotta, tuttavia questa compressione rende difficile progettare app per unikernel, poiché ad esempio posso usare solo un linguaggio di programmazione.

c) Spiegazione approcci pre-copy e post copy per migrazione live di una macchina virtuale, pro e contro.

Nella migrazione live di una vm possiamo sfruttare due tecniche, la loro applicazione può dipendere dal tipo di carico a cui è sottoposta la memoria.
Approccio pre-copy: Qui vengono inizialmente copiate tutte le pagine dalla vm sorgente a quella di destinazione, in modo iterativo, mentre la vm sorgente è ancora in run. Terminata questa fase di copia, la vm sorgente viene stoppata, e nella vm destinazione vengono copiate informazioni come stato della cpu e driver. Infine, la vm sorgente viene rimossa, la vm destinazione riprende l'esecuzione.
Questo approccio è ideale per carichi read-intensive, poiché abbiamo già tutte le pagine pronte per la lettura.
L'approccio post-copy prevede lo spostamento dello stato stato cpu e driver subito, mentre le pagine vengono acquisite man mano che ce ne sia bisogno. Questo richiede che la vm sorgente rimanga attiva più tempo, e ad ogni "primo accesso" ad una pagina non copiata, ci sia un page fault.

OSSERVAZIONE:
E' importante ricordare che, quando facciamo queste migrazioni, migriamo sostanzialmente due cose: **pagine** e **stati(cpu,driver)**. Con pre copy faccio prima pagine iterative e poi stati, con post-copy prima stati e poi le pagine.

d) Problema del ring deprivileging, come viene risolto nella virtualizzazione a livello di sistema con supporto hw e nella paravirtualizzazione.
Normalmente, un'architettura presenta dei livelli detti ring, che esprimono livelli di privilegio, e vanno dall'user al superuser (privilegi massimi). Nel caso della virtualizzazione a livello di sistema, si verifica un problema detto "ring deprivileging", in cui sostanzialmente il sistema virtualizzato non ha i privilegi per le chiamate di sistema. Questo problema viene risolto nella virtualizzazione con supporto hw introducendo due modalità operative: root mode e non root mode. Il vmm/hypervisor gira in root mode (ring 0 root), mentre il sistema operativo guest in ring 0 non root, risolvendo anche il ring compression (proteggiamo lo spazio del sist.op.).
Nota: nella virtualizzazione a livello di sistema, possiamo non avere supporto hw, e quindi usiamo **Fast Binary Translation**, in cui si scansionano e traducono blocchi di codice. Qui, quando ho istruzione privilegiata, la mando al vmm che poi le esegue sul computer host. Virtualizzazione completa: espongo interfacce hw funzionalmente uguali
Nella paravirtualizzazione, non viene emulato l'hardware, ma creato uno strato minimale di software, e quindi modificando il kernel del sistema operativo guest per interagire, introducendo trap software dette **hypercall**. Poiché il s.o. guesto opera su ring 0, risolvo ring deprivileging. Virtualizzazione completa: espongo interfacce hw funzionalmente simili, non uguali.

e) La virtualizzazione a livello di sistema operativo (aka container) presenta problemi di ring deprivileging? come si realizza?
Nell'ambito dei container, non si ha il problema del ring deprivileging, perché il container condivide il kernel del sistema operativo; quindi, non vado ad intaccare la struttura sottostante. Il container è infatti un ambiente isolato in termini di processi, gruppi, etc ma che condivide stesso sistema operativo (e quindi stesso kernel) con altri container. Posso realizzarlo mediante chroot (cambio root directory del processo corrente), cgroups (isolo le risorse per un gruppo di processi) e namespace (isolo la vista per un insieme di processi).

f) Come viene realizzata la migrazione live della memoria di una vm?
pre-copy e post copy, già visti.
pre-copy: prima copio le pagine in modo iterativo (step x: copio pagine modificate allo step precedente), poi fermo vm sorgente e copio stato cpu e driver, e poi la spengo in favore di quella nuova.
post copy: copio stato cpu e driver, e poi quando mi server una pagina (si crea page fault perché non ce l'ho) la copio dalla vm sorgente, che quindi rimane accesa più tempo.

g) Perché i container i Docker sono abilitanti ai microservizi?
Con i microservizi trattiamo componenti quanto più possibile indipendenti, sono servizi

debolmente accoppiati, autocontenuti, e comunico tramite chiamate. Date le loro dimensioni ridotte, normalmente si associa un container ad ogni microservizio, questo produce quindi un sistema multi-container che deve essere orchestrato. Normalmente gli approcci sono mediante Docker Compose (se ho container su singolo host) o Docker Swarm (se ho container su host multipli).

h) Perché è difficile mantenere clock fisici sincronizzati su un sistema distribuito a larga scala?

In un sistema distribuito a larga scala, è fondamentale risalire all'ordinamento degli eventi. La difficoltà nel mantenere un clock fisico nasce dall'impossibilità di averne uno comune, e nel caso di clock fisici associati a più nodi bisogna considerare ritardi, drift rate, etc.. Lamport dimostrerà come è più importante mantenere solo l'ordinamento degli eventi, piuttosto che il tempo in cui avvengono.

i) **Descrizione di Cristian e Berkley, pro e contro.**

L'algoritmo di Cristian prevede la sincronizzazione con un Time Server S centralizzato, sincronizzazione esterna.
Un processo p richiede il tempo tramite messaggio, e lo riceve con un messaggio di risposta. p imposterà il suo clock a t + tround/2, ovvero = tempo del Server + tempo per far arrivare il messaggio da S a p.
L'accuratezza è data da a <= Tround/2 – min, dove min è il tempo minimo di trasmissione.
L'algoritmo di Berkley è invece di sincronizzazione interna, centralizzato.
Il time server (Master) richiede in broadcast i valori dei worker (incluso sé stesso), calcola la differenza tra il suo tempo e quello dei worker, ne fa una media e invia il valore correttivo (media + clock di M).

j) **Clock: 16:31:54:00, più avanti di 4 secondi. Perché non posso metterlo direttamente corretto? Come devo regolarlo per averlo corretto dopo otto secondi?**

In un ambito di clock non posso mandare indietro il tempo, creerei problema di consistenza e sincronizzazione. Posso solo rallentarlo, perché il tempo è monotono crescente. Il tempo corretto è 16:31:50, tra 8 secondi sarà 16:31:58.
Il nostro tempo è 16:31:54, quindi se rallentiamo il clock, facendo in modo che ogni secondo corrisponda ad un aumento di mezzo secondo di clock, otteniamo l'orario corretto. (secondi che mancano per essere corretti/secondi a disposizione per fare ciò = 4/8 = 0.5)

ESERCIZI VARI SULLA CONSISTENZA

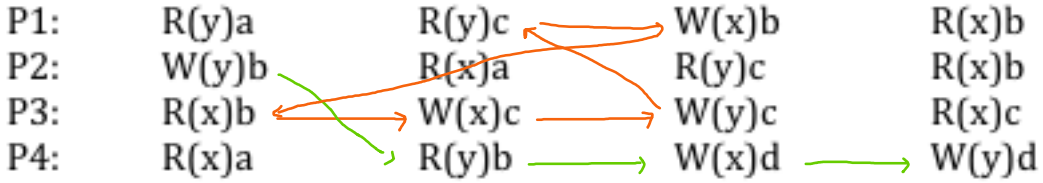
1) Si consideri il seguente archivio di dati:

P1:	W(x)a	R(x)b	W(x)c
P2:	R(x)a	W(x)b	R(x)c
P3:	R(x)X	R(x)?	

L'incognita "X" può assumere valori "a", "b" o "c". Per ogni caso vedere se è realizzabile la consistenza **sequenziale** (ordine di programma).

- Caso X = a,
  - ? = a
    W1(x,a) – R2(x,a) – R3(x,a) – R3(x,a) – W2(x,b) – R1(x,b) – W1(x,c) – R2(x,c)
  - ? = b
    W1(x,a) – R2(x,a) – R3(x,a) – W2(x,b) – R1(x,b) – R3(x,b) - W1(x,c) – R2(x,c)
  - ? = c
    W1(x,a) – R2(x,a) – R3(x,a) – W2(x,b) – R1(x,b) - W1(x,c) – R2(x,c) – R3(x,c)
- Caso X = b
  - ? = a
    W1(x,a) – R2(x,a) – *R3(x,a)* – W2(x,b) – R1(x,b) – *R3(x,b)* – W1(x,c) – R2(x,c). ATTENZIONE: non va bene! Quando concludo la sequenza, una verifica che possiamo fare è la seguente: per ogni processo dobbiamo trovare, sull'interleaving, l'ordine esatto delle letture. Notiamo come P3 prima legga "b" e poi "a", ma nell'interleaving avviene il contrario e non va bene.
  - ? = b
    W1(x,a) – R2(x,a) – W2(x,b) – R1(x,b) – *R3(x,b)* – *R3(x,b)* - W1(x,c) – R2(x,c). Va bene.
  - ? = c
    W1(x,a) – R2(x,a) – W2(x,b) – R1(x,b) – *R3(x,b)* - W1(x,c) – R2(x,c) - – *R3(x,c)*. Va bene.
- Caso X = c
  - ? = a
    Non realizzabile, perché P2 legge prima "a" e poi "c", mentre P3 "c" e poi "a".
  - ? = b
    Non realizzabile, infatti P3 dovrei avere prima "c" e poi "b", ma vedendo P1 e P2 dovrei prima mettere "b" e poi "c".
  - ? = c
    W1(x,a) – R2(x,a) – W2(x,b) – R1(x,b) - W1(x,c) – R2(x,c) – *R3(x,c)* - *R3(x,c)*. Va bene.

2)



C'è consistenza causale? C'è ciclo, quindi no.
Se tolgo R1(y,c)? Si spezza il ciclo, e quindi c'è.
NB: Su stesso processo seguo R-W su dati diversi, variabile indifferente.
Su processi diversi, devo avere W-R su stesso dato, e **forse stessa variabile?** (non specificato nelle slide, parla solo di dato).

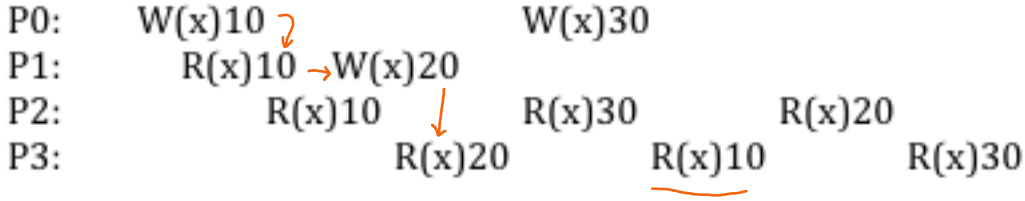
3) Massimo grado di consistenza?

P1:	W(x)b	R(x)c	R(y)b	W(x)d
P2:	R(y)a	W(y)b	R(x)b	R(y)d
P3:	R(x)b	W(x)c	R(y)c	R(x)d
P4:	R(x)a	R(x)b	W(y)c	W(y)d

- Stretta? Non ce l'ho mai, è quasi irrealizzabile.
- Linearizzabile? No, ad esempio in P3 leggo R3(y,c), che mai ho scritto.
- Sequenziale?

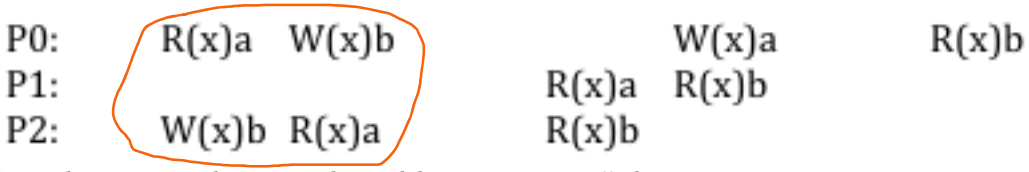
R2(y,a) – R4(x,a) – W1(x,b) – R3(x,b) – R4(x,b) – W3(x,c) – R1(x,c) – W2(y,b) – R1(y,b) – R2(x,b) – W4(y,c) – R3(y,c) – W1(x,d) - R3(x,d) - W4(y,d) - R2(y,d)
Va bene, perché per ogni processo, le sue operazioni sono ordinate nell'interleaving.

4) Massimo grado di consistenza?



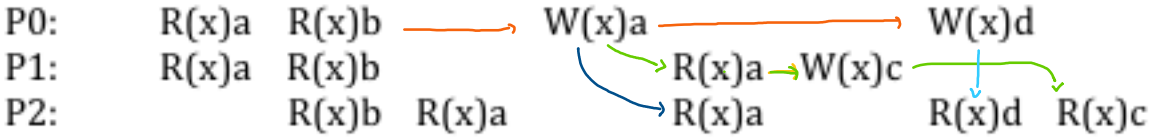
partiamo con la sequenziale: No, perché P2 legge prima 30 e poi 20, mentre P3 legge 20 e poi 30.
Causale: Trovo le relazioni: W0(x,10) -> R1(x,10) -> W1(x,20) - > R3(x,20)
quindi questo ci dice che 10 è la causa e 20 è l'effetto.
In P3 leggo prima l'effetto e poi la causa, quindi violo!

5) Soddisfo la sequenziale?



No: il mio interleaving dovrebbe "rispettare" di tutte **le read e write dei vari processi**. Assumiamo che il dato di 'a' sia già stato inserito. P0 mi imporrebbe di leggere prima 'a' e poi scrivere 'b', mentre P2 mi dice il contrario. Appena trovo su due processi, due operazioni su stesse variabili, invertite, non ho sequenziale.

6) Soddisfo la causale?



b – a – d, b – a – c? "c" e "d" concorrenti, perché non c'è causa effetto.
P2 non dà problemi, perché ho "b->a" e poi "d->c" ma non avendo alcun vincolo su quest'ultimo, non mi da problemi.

7) Es.1a seconda prova intermedia 18/19
Ho consistenza sequenziale?

P1:	W(x)1	R(x)1	R(y)0
P2:	W(y)1	R(y)1	R(x)1
P3:	R(x)1	R(y)0	
P4:	R(y)0	R(x)0	

Devo trovare ordine programma.
R4(y,0) - R4(x,0) - W1(x,1) – R1(x,1) – R1(y,0) – R3(x,1) – R3(y,0) – W2(y,1) – R2(y,1) – R2(x,1), stessa soluzione della prof.

8) Es. 1b seconda prova intermedia 18/19
Ho consistenza sequenziale? e Causale?

P1:		W(x)1				
P2:			W(x)2			
P3:	R(x)2		R(x)1		W(y)3	
P4:				R(x)2		W(x)1 R(y)3

Cerco la sequenziale:
W2(x,2)- R3(x,2) – R4(x,2) – W1(x,1) – R3(x,1) – W4(x,1) – W3(y,3) – R4(y,3)
allora anche la causale è inclusa.

9) K assume {b,d,e}, che valori deve avere "?" per avere consistenza sequenziale o causale?

P1:	R(x)b	W(y)c	W(x)d	
P2:	W(x)a	W(x)b	W(x)e	R(y)c
P3:	R(x)b	R(x)e	R(y)c	R(x)d
P4:	R(x)K	R(y)c	R(x)?	

Svolgimento

a) K = b,? = b
W2(x,a) – W2(x,b) – R1(x,b) – R3(x,b) – R4(x,b) –W1(y,c) – R4(y,c) – R4(x,b) - W2(x,e) – R3(x,e) – R2(y,c) – R3(y,c) – W1(x,d) – R3(x,d).

b) K = b, ? = d
W2(x,a) – W2(x,b) – R1(x,b) – R3(x,b) – R4(x,b) –W1(y,c) – R4(y,c) –- W2(x,e) – R3(x,e) – R2(y,c) – R3(y,c) – W1(x,d) – R3(x,d) - R4(x,d).

c) K = b, ? = e
W2(x,a) – W2(x,b) – R1(x,b) – R3(x,b) – R4(x,b) –W1(y,c) – R4(y,c) –- W2(x,e) – R3(x,e) – R4(x,e) – R2(y,c) – R3(y,c) – W1(x,d) – R3(x,d)

d) K = d, ? = b
W2(x,a) – w2(x,b) – attenzione. P4 mi dice di leggere "d" e poi "b", P3 il contrario, quindi non c'è interleaving. **(Sulla stessa variabile!)**
C'è causale? No, perché abbiamo ordine "b-c-d", ma p4 ha ordine d ->b

e) K = d, ? = d
w2(x,a) – w2(x,b) – r1(x,b) – r3(x,b) – w1(y,c) – w2(x,e) - r3(x,e) - r2(y,c) - r3(y,c) – w1(x,d) – r3(x,d) – r4(x,d)

f) K = d, ? = e
Non è sequenziale, perché p3 legge prima "e" e poi "c", mentre p4 prima "c" e poi "e".
E' causale? P3 legge prima "e" e poi "c", mentre p4 fa il contrario. E' un problema se c'è causa effetto tra "e" e "c", che però non c'è.

g) K = e, ? = b
Non c'è interleaving, perché P3 legge prima "b" e poi "e", mentre P4 legge prima "e" e poi "b".
E' causale? da P1 c'è prima "b" e poi "c", ma P4 fa il contrario. quindi no.

h) K = e, ? = d
w2(x,a) w2(x,b) r1(x,b) r3(x,b) w2(x,e) r3(x,e) r4(x,e) w1(y,c) r2(y,c) r3(y,c) r4(x,c) w1(x,d) r3(x,d) r4(x,d)

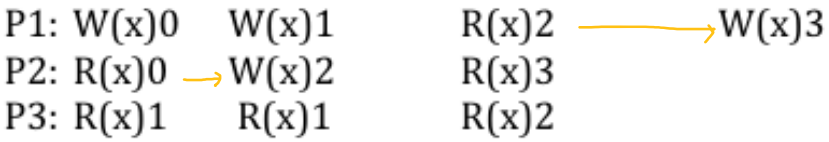
i) K = e, ? = e
w2(x,a) w2(x,b) r1(x,b) r3(x,b) w2(x,e) r4(x,e) r3(x,e) w1(y,c) r3(y,c) r4(y,c) r4(x,e) w1(x,d) r3(x,d)

10) esercizio 2 19/20
Che consistenza si ha?

P1:	W(y)0	W(x)1	R(x)1	R(y)0
P2:		W(y)1	R(y)1	R(x)1
P3:		R(x)1	R(y)0	
P4:	W(x)0	R(y)0	R(x)0	

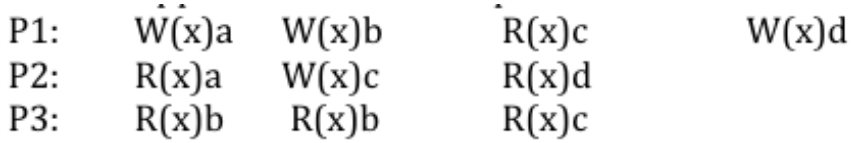
Stretta? no.
Linearizzabile? no, perché in colonna 3 P3 legge y,0 e P4 x,0 che però sono già state sovrascritte nella colonna 1 (in cui scrivo 1 su x ed y).
Sequenziale?
w1(y,0) w4(x,0) r4(y,0) r4(x,0) w1(x,1) r1(x,1) r3(x,1) r1(y,0) r3(y,0) w2(y,1) r2(y,1) r2(x,1)

11) Che consistenza si ha?

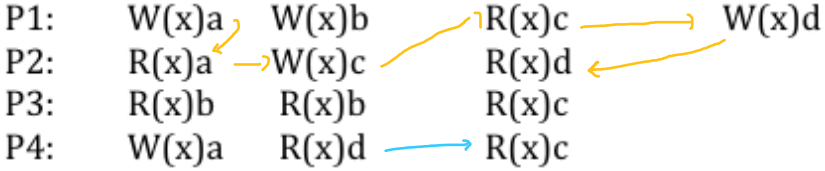


Vedo direttamente la sequenziale:
w1(x,0) r2(x,0) w1(x,1) r3(x,1) r3(x,1) w2(x,2) r1(x,2) r3(x,2) w1(x,3) r2(x,3)

12) seconda prova 21/22, es 1



Stretta? no.
Linearizzabile? no, P3 legge b prima che venga scritto.
Sequenziale?
w1(x,a) r2(x,a) w1(x,b) r3(x,b) r3(x,b) w2(x,c) r1(x,c) r3(x,c) w1(x,d) r2(x,d)
eccola.



Dalla sequenza, notiamo che l'ordine è : a -> c ->d
ma p4 presenta ordine d->c, quindi non rispetto la sequenziale.
Causale? Neanche, perché prima ho ragionato sulla causalità!
Ho ragionato in questo modo: se vale sequenziale -> vale causale.
Se dimostro che causale non vale -> non vale sequenziale.