Ηλίας Λεοντιάδης

Πέτσιος Στέφανος

Άσκηση 1

Το $V_i[i]$ είναι ο αριθμός που ο p_i των timestamp που έχει κάνει. Κάθε άλλος κόμβος αυξάνει το $V_j[i]$ μόνο όταν είναι μεγαλύτερο από αυτό που είχε, έτσι ο κόμβος p_i έχει το μεγαλύτερο timestamp άρα ισχύει $V_j[i] \leq V_i[i]$.

Αφού τα e και e' είναι concurrent τοτε το e' θα έχει μικρότερο timestamp $V_{e'}[e]$ από το $V_e[e]$ και αυτό γιατί δεν έχει μεταδοθεί κανένα μήνυμα μεταξύ τους (concurrent). Φυσικά το ίσιο ισχύει και αντίστροφα
δηλαδή:
$V_{e'}[e] < V_e[e]$
$V_e[e'] < V_{e'}[e']$

Άρα δεν ισχύει κανένα απο τα παρακάτω.
$V(e) \leq V(e')$
$V(e') \leq V(e)$

Εάν ισχύει $V(e)<V(e')$ τότε για τα δύο γεγονότα e και e' ίσχύει το "happened before" (δεν είναι concurrent) δηλαδή ισχύει το e→ e'

Άσκηση 2

Άσκηση 9 βιβλίου

Ναι είναι πιθανό. Μία τέτοια ακολουθία που θα δώσει 000000 είναι η εξής:

Εκτελείται το α και εκτυπώνει 00.

Εκτελείται το β αλλά δεν έχει διαδοθεί ακόμα η αλλαγή του α οπότε τυπώνει 00.

Ομοίως στο γ δεν φτάνουν οι αλλαγές από τα α και β και τυπώνει 00...

Άσκηση 10 βιβλίου

p1	x = 1	
p1	print(y,z)	εκτυπώνει 00
p3	z = 1	
p3	print(x,y)	εκτυπώνει 10
p2	y = 1	
p2	print(x,z)	εκτυπώνει 11

Το signature σε αυτή την περίπτωση είναι 001110

Άσκηση 12 βιβλίου

```
x = 1
y = 1
if(y = = 0) kill(p2)
if(x = =0 ) kill(p1)
```

```
x = 1
y = 1
if(x = =0 ) kill(p1)
if(y = = 0) kill(p2)
```

```
y = 1
x = 1
if(y = = 0) kill(p2)
if(x = =0 ) kill(p1)
```

```
y = 1
x = 1
if(x = =0 ) kill(p1)
if(y = = 0) kill(p2)
```

```
x = 1
if(y = = 0) kill(p2)
y = 1
if(x = =0 ) kill(p1)
```

```
y = 1
if(x = =0 ) kill(p1)
x = 1
if(y == 0) kill(p2)
```

Άσκηση 27 βιβλίου

Μια διαδικασία write μένει στην ουρά μέχρι η τοπική βάση δεδομένων να ενημερωθεί για όλες τις διαδικασίας που προηγούνται causally και από τις οποίες εξαρτάται η συγκεκριμένη εγγραφή. Κρατάει δηλαδή write operations που εξαρτώνται από άλλες operations που προηγούνται και δεν έχουν φτάσει ακόμα στην ουρά.

Για να εξυπηρετήσουμε δηλαδή την αίτηση πρέπει:
Όλες οι διεργασίες που στάλθηκαν κατευθείαν στο L_i από άλλους clients και προηγούνται να έχουν επεξεργασθεί. Δηλαδή $ts(W)[i] = VAL(i)[i] + 1$.
Και δεύτερον όλα τα updates στα οποία βασίζεται το W πρέπει να έχουν επίσης επεξεργασθεί από το L_i. Δηλαδή: $ts(W)[j] \leq VAL(i)[j] \quad \forall\, j \neq i$

Άσκηση 3

Monotonic Reads

Υποστηρίζει monotonic reads.

Αν κάποιος διαβάσει ένα δεδομένο και μετά αλλάξει θέση δεν μπορεί να διαβάσει παλιότερη τιμή γιατί αυτό είναι αντίθετο με το sequential consistency. Δεν υπάρχει δηλαδή δυνατή διάταση (σειριακή) που να επιτρέπει σε ένα δεδομένο να διαβάσει ξαφνικά παλιότερη τιμή.

Πχ άν η αρχική τιμή είναι a τότε

P_1 $W(x_1)\beta$

P_2 $R(x)\ \alpha$

$R(x)\beta$

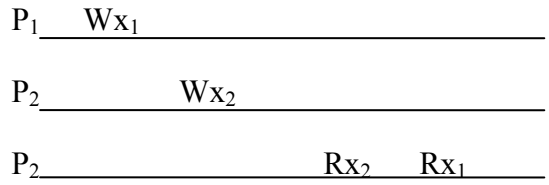
η μοναδική valid sequentially consistent ακολουθία είναι Rxα Wxβ Rxβ και όχι το αντίστροφο. Έτσι υποστηρίζει monotonic reads.

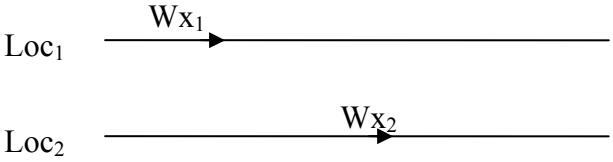
Monotonic Writes

Δεν υποστηρίζει monotonic writes. Γιατί για παράδειγμα μπορεί κάποιος να γράψει μια τιμή και να αλλάξει θέση. Όταν φτάσει στην άλλη θέση γράφει την νέα τιμή. Το σύστημα όμως και στις δύο θέσεις λαμβάνει αντίθετα την σειρά που γράφτηκαν τα δεδομένα κάτι που είναι συμβατό με το sequential consistency.

Αυτό σημαίνει όμως ότι στην νέα θέση το write θα τελειώσει πριν ολοκληρωθεί το προηγούμενο write.

Σχηματικά:





Παραπάνω βλέπουμε μια sequential consistency διαδικασία. Όμως αν η διαδικασία p1 και p2 είναι ουσιαστικά ο ίδιο client σε διαφορετικές Locations (όπως φαίνεται στο παρακάτω σχήμα) τότε όταν κάνει το Wx_2 δεν θα έχει γραφεί το Wx_1. Κάτι που δέν είναι monotonic write.

Read your writes

Με το ίδιο σκεπτικό μπορεί κάποιος να γράψει μια τιμή μετά να μετακινηθεί. Στο sequential consistency μπορεί να γίνει εναλλαγή του read με το write. Έτσι στην νέα θέση μπορεί να μην βρει την τιμή που έγραψε (ή πιο καινούργια).

Άρα δεν υποστηρίζει read your writes.

Write follow reads

Ισχύει. Γιατί δεν μπορεί να γίνει αντιμετάθεση R με R και W στο sequential consistency.