Άλκης Γεωργόπουλος A.M. 39
Αναστάσιος Κοντογιώργης A.M. 43

Άσκηση 1.

Λόγω της σύγχυσης που είχε προκληθεί τη προηγούμενη φορά σχετικά με ένα λάθος στην εκφώνηση, σας στέλνουμε και τα δύο ερωτήματα της προηγούμενης άσκησης.

α) Δείξτε ότι $V_j[i] \leq V_i[i]$

Το $V_i[i]$ είναι ο αριθμός των γεγονότων που έχουν συμβεί μέχρι μια δεδομένη στιγμή στη διεργασία P_i. Το $V_j[i]$ είναι ο αριθμός των γεγονότων που η διεργασία P_j ξέρει ότι έχουν συμβεί στη P_i .
Όταν η P_i στείλει ένα μήνυμα στη P_j , στέλνει μαζί και ένα διάνυσμα με timestamps και αυξάνει το $V_i[i]$ κατά 1. Όταν η P_j λάβει το μήνυμα θα αυξήσει κατά 1 το $V_j[i]$, **μόνο** εάν στο vector που περιείχε το μήνυμα υπάρχουν timestamps με μεγαλύτερες τιμές για την P_i .
Πιο απλά, το $V_j[i]$ είναι ο αριθμός των γεγονότων που συνέβησαν στην P_i και η P_j σχετίζεται (ή επηρεάζεται) από αυτά.
Οπότε συνεχώς το $V_j[i]$ θα παραμένει μικρότερο ή ίσο από το $V_i[i]$.

β) Show that if events *e* and e' are concurrent then neither $V(e) \leq V(e')$ nor $V(e') \leq V(e)$.
Hence show that if then $V(e) < V(e')$, e --> e'.

Θεωρούμε ότι το γεγονός e έγινε σε μια διεργασία P_i και το e' σε μια P_j .
Αφού τα δύο γεγονότα είναι concurrent ξέρουμε ότι κανένα μήνυμα δεν ανταλλάχτηκε μεταξύ των P_i και P_j , μετά την αποστολή των timestamps για τα δύο γεγονότα. Όποτε από το α) ερώτημα συμπεραίνουμε ότι $V_j[i] < V_i[i]$ και $V_i[j] \leq V_j[j]$, που σημαίνει ότι δεν ισχύει ούτε $V(e) \leq V(e')$, ούτε $V(e') \leq V(e)$.

Αν τώρα ξέρουμε ότι ισχύει $V(e) < V(e')$ σημαίνει ότι τα γεγονότα δεν είναι concurrent, δηλαδή υπάρχει μεταξύ τους η happens before σχέση. Και προφανώς ισχύει e --> e'.

Άσκηση 2

9. In Fig. 6-7, is 000000 a legal output for a distributed shared memory that is only FIFO consistent? Explain your answer.

Ναι, είναι, αν κάθε διεργασία «δει» τις αναθέσεις των άλλων διεργασιών μετά την δική της print. Αυτό μπορεί να γίνει αν οι διεργασίες δουν αντίστοιχα τις ακόλουθες σειρές εκτέλεσης:

x = 1;	y = 1;	z = 1;
print(y, z);	print(x, z);	print(x, y);
y = 1;	x = 1;	x = 1;

print(x, z);	print(y, z);	print(y, z);
z = 1;	z = 1;	y = 1;
print(x, y);	print(x, y);	print(x, z);
Prints: 00	Prints: 00	Prints: 00
(a)	(b)	(c)

10. In Fig. 6-8, is 001110 a legal output for a sequentially consistent memory? Explain your answer.

Όχι, γιατί το τελευταίο «0» του output σημαίνει η τελευταία print που εκτελέστηκε (δεν μας ενδιαφέρει από ποια διεργασία) **δεν είχε δει κάποια ανάθεση τιμής**. Όμως όλες οι print έχουν διεκπεραιωθεί, άρα και όλες οι αναθέσεις τιμής. Επομένως όλα τα νόμιμα αποτελέσματα στην ακολουθιακή συνέπεια πρέπει να τελειώνουν σε «11».

Παρατήρηση: στο βιβλίο, κατά το σχολιασμό της Fig. 6-8, χρησιμοποιείται η υπογραφή (signature) και όχι το output. Αν η εκφώνηση της άσκησης εννοούσε signature, τότε η απάντηση θα ήταν θετική, εάν οι διεργασίες τρέχανε με τη σειρά a), c), b), οπότε η διεργασία b) θα έκανε την τελευταία print.

12. In Fig. 6-13, a sequentially consistent memory allows six possible statement interleavings. List them all.

x = 1;	x = 1;	x = 1;	y = 1;	y = 1;	y = 1;
if (y == 0)	y = 1;	y = 1;	if (x == 0)	x = 1;	x = 1;
kill(P2);	if (y == 0)	if (x == 0)	kill(P1);	if (x == 0)	if (y == 0)
y = 1;	kill(P2);	kill(P1);	x = 1;	kill(P1);	kill(P2);
if (x == 0)	if (x == 0)	if (y == 0)	if (y == 0)	if (y == 0)	if (x == 0)
kill(P1);	kill(P1);	kill(P2);	kill(P2);	kill(P2);	kill(P1);
(12ab)	(1a2b)	(1ab2)	(ab12)	(a1b2)	(a12b)

Οι συμβολοσειρές στις παρενθέσεις δίνουν μία οπτικοποίηση των μεταθέσεων, αν θεωρήσουμε ότι (1,2) είναι οι εντολές της διεργασίας P1 και (ab) οι εντολές της P2.

27. Consider causally-consistent lazy replication. When exactly can an operation be removed from a write queue?

Μια αίτηση για write λειτουργία κρατείται στην ουρά ενός αντιγράφου, μέχρι η τοπική βάση δεδομένων να έχει ενημερωθεί (update) πλήρως, σε σχέση με όλες τις προηγούμενες write λειτουργίες από τις οποίες εξαρτάται η write λειτουργία στην ουρά.
Για να εκτελεστεί μια write λειτουργία που βρίσκεται στην ουρά του αντιγράφου L_i απαιτείται να ισχύουν οι ακόλουθες δύο συνθήκες (σύμφωνα με τους συμβολισμούς του βιβλίου):
1. ts(W)[i] = VAL(i)[i] + 1

2. $ts(W)[j] \leq VAL(i)[j] \quad \forall\, j \neq i$

Μετά την εκτέλεση της η write λειτουργία παραμένει στην ουρά μέχρι να εκτελεστούν όλα τα απαραίτητα updates στα υπόλοιπα αντίγραφα. Μόλις γίνει γνωστό ότι έχουν ενημερωθεί όλα τα υπόλοιπα αντίγραφα για τη write λειτουργία που υπάρχει στην ουρά, τότε αυτή μπορεί με ασφάλεια να διαγραφεί.
Ουσιαστικά μετά την εκτέλεση της μια write λειτουργία στην ουρά ενός αντιγράφου i, αποτελεί ένα update record, έστω u. Ένα αντίγραφο i, ξέρει ότι ένα update record u είναι γνωστό παντού, όταν έχει λάβει μηνύματα (gossip messages) που περιέχουν το u, από όλα τα υπόλοιπα αντίγραφα. Κάθε gossip μήνυμα περιέχει αρκετές πληροφορίες από την ουρά του αποστολέα, ώστε να μπορεί να εγγυηθεί, ότι όταν ο παραλήπτης παραλαμβάνει ένα record u από ένα αντίγραφο i, έχει ήδη παραλάβει (είτε στο ίδιο gossip message, είτε σε προηγούμενο) όλες τις εγγραφές που έχουν εκτελεστεί στο i πριν από το u.
Για την υλοποίηση αυτού του μηχανισμού ενημέρωσης, απαιτείται η χρήση ενός επιπλέον πίνακα ts_table σε κάθε αντίγραφο. Ο πίνακας αυτός χρησιμοποιείται για να καθορίσει πότε μια εγγραφή έχει γνωστοποιηθεί παντού. Κάθε gossip message περιέχει ένα timestamp από τον αποστολέα. Το ts_table(k) περιέχει το μεγαλύτερο timestamp που έχει παραλάβει αυτό το αντίγραφο, από το αντίγραφο k. Προφανώς το τρέχων timestamp του αντιγράφου k θα πρέπει να είναι μεγαλύτερο ή ίσο με αυτό που έχει αποθηκευτεί στο ts_table οποιουδήποτε άλλου αντιγράφου.
Επιπλέον κάθε αντίγραφο γνωρίζει τα ts_table όλων των υπολοίπων αντιγράφων. Αυτό είναι εύκολο να γίνει, εάν κάθε φορά που αλλάζει ένα ts_table, ενός αντιγράφου j, στέλνεται ένα broadcast μήνυμα σε όλα τα υπόλοιπα αντίγραφα με το νέο ts_table_j. Αν $ts\_table(k)_j = t$ στο αντίγραφο i, τότε το αντίγραφο i, ξέρει ότι το αντίγραφο k έχει ενημερωθεί για τα πρώτα t updates που έγιναν από το αντίγραφο j.
Κάθε εγγραφή r περιέχει τη ταυτότητα του αντιγράφου που τη δημιούργησε (δηλαδή του αντιγράφου που στο οποίο στάλθηκε αρχικά η αίτηση για write λειτουργία) στο πεδίο r.node.

Τελικά το αντίγραφο i μπορεί να διαγράψει μία update εγγραφή r από την ουρά του, όταν γνωρίζει ότι η r έχει παραληφθεί από κάθε άλλο αντίγραφο. Δηλαδή όταν στο αντίγραφο i ισχύει:

$\forall$ replica j, $ts\_table(j)_{r.node} \geq r.ts_{r.node}$.

Άσκηση 3

Στα data centric μοντέλα έχουμε πολλές διεργασίες οι οποίες αλληλεπιδρούν με ένα data store. Αντίθετα, στα client centric μοντέλα θεωρούμε ότι έχουμε **μία μόνο** διεργασία με δικαιώματα εγγραφής σε κάποιο data item x, ώστε να αποφεύγονται τα write-write conflicts (σελ. 319).

Έτσι για να έχει νόημα η εκφώνηση θα πρέπει να θεωρήσουμε ως διεργασίες τις αιτήσεις που γίνονται σε κάθε replica και όχι π.χ. την μοναδική διεργασία του φορητού υπολογιστή στο σχήμα 6-19. Δηλαδή αν ο φορητός υπολογιστής κάνει ένα write, στη συνέχεια αλλάξει location και κάνει άλλο write, θεωρούμε ότι αυτά τα δύο write προήρθαν από διαφορετικές «διεργασίες».

Sequential Consistency and Monotonic Reads

Η απάντηση είναι ναι, με την προϋπόθεση ότι μία read operation που θα κάνει ένας client θα πρέπει να διεκπεραιωθεί πριν αυτός αλλάξει location. Δηλαδή η sequential consistency εγγυάται ότι **εφόσον** ο client έκανε μία read operation και **περίμενε** το σύστημα να την ταξινομήσει ακολουθιακά σε σχέση με τα όποια τρέχοντα write operations, στη συνέχεια όλες αυτές οι προσπελάσεις φαίνονται με την ίδια σειρά σε όλα τα replicas, και επομένως αν συνδεθεί στη **συνέχεια** με κάποια άλλη replica η επόμενη read operation που θα κάνει θα θεωρηθεί από το σύστημα ότι **έπεται** της προηγούμενης, και άρα παρέχονται monotonic reads.

Sequential Consistency and Monotonic Writes

Η απάντηση είναι όχι. Το sequential consistency μοντέλο δεν εγγυάται πάντα Monotonic Writes.
Ο λόγος είναι ότι ένας client μπορεί να κάνει μία write operation και αυτή να μην γίνει propagate σε όλες τις replicas πριν αυτός αλλάξει location. Έτσι αν κάνει μία ακόμα write operation από διαφορετική replica, είναι δυνατόν να «ταξινομηθεί» ακολουθιακά **πριν** από αυτήν που έκανε στην προηγούμενη replica.

Sequential Consistency and Read your Writes

Η απάντηση είναι όχι. Για τον ίδιο λόγο, αν ένας client κάνει μία write operation, αλλάξει τοποθεσία και κάνει μία read operation, υπάρχει η πιθανότητα να θεωρηθεί ότι η read έγινε πριν από την write, και έτσι βλέπουμε ότι η sequential consistency δεν συμβαδίζει πάντα με το read your writes μοντέλο.

Sequential Consistency and Writes follow reads

Η απάντηση είναι ναι, πάλι με την προϋπόθεση ότι μία read operation ενός client πρέπει να διεκπεραιωθεί πριν αυτός αλλάξει location.
Αν ένας client κάνει μία read, πριν αυτός μπορέσει να αλλάξει replica, θα πρέπει το σύστημα να ταξινομήσει ακολουθιακά τα τρέχοντα operations. Έτσι όταν στη συνέχεια κάνει μία write σε διαφορετική replica, αυτή είναι εγγυημένη να συμβεί στην ίδια ή σε νεότερη έκδοση του data store. Τελικά βλέπουμε ότι η sequential consistency εγγυάται ότι τα writes follow reads.