

Κατανομημένα Συστήματα I

Μάθημα Βασικής Επιλογής,

Χειμερινού Εξαμήνου

Τομέας Εφαρμογών και Θεμελιώσεων

Ιωάννης Χατζηγιαννάκης

Δευτέρα, 26 Οκτωβρίου, 2009
Αίθουσα Β3



Προηγούμενο Μάθημα

- ▶ Σύγχρονα Κατανομημένα Συστήματα
- ▶ Πρόβλημα Εκλογής Αρχηγού
 - ▶ Δίκτυα Δακτυλίου
 - ▶ Αλγόριθμος TimeSlice
 - ▶ Αλγόριθμος των Itai και Rodeh
 - ▶ Γενικά Δίκτυα
 - ▶ Αλγόριθμος FloodMax
- ▶ Αναζήτηση κατά Εύρος
 - ▶ Αλγόριθμος SyncBFS
- ▶ Συντομότερα Μονοπάτια
 - ▶ Αλγόριθμος των Bellman και Ford



Μοντέλο Σύγχρονου Δικτύου

- ▶ Μία συλλογή υπολογιστικών μονάδων ή 'επεξεργαστές'
 - ▶ κάθε επεξεργαστής εκτελεί μόνο μία διεργασία
- ▶ Οι μονάδες του συστήματος είναι συνδεδεμένες με ένα σύγχρονο δίκτυο
 - ▶ Ορίζουμε το σύγχρονο δίκτυο ως ένα **κατευθυνόμενο γράφημα** $G = (V, E)$
 - ▶ αποτελείται από $n = |V|$ **κορυφές** και $m = |E|$ **ακμές**
- ▶ Υποθέτουμε ότι κάθε κανάλι επικοινωνίας μπορεί να δεχτεί μόνο ένα μήνυμα τη φορά
 - ▶ τα κανάλια είναι οι ακμές του γραφήματος
- ▶ Θεωρούμε ότι υπάρχει ένα δεδομένο αλφάβητο M μηνυμάτων



Οι Καταστάσεις των Διεργασιών

- ▶ Κάθε διεργασία $u \in V$ χαρακτηρίζεται από ένα σύνολο καταστάσεων $states_u$
 - ▶ Ορισμένες τις ονομάζουμε **αρχικές καταστάσεις** $start_u$
 - ▶ Ορισμένες τις ονομάζουμε **καταστάσεις τερματισμού** $halt_u$
- ▶ Διαθέτει μια γεννήτρια εξερχόμενων μηνυμάτων $msgs_u : states_u \times nbrs_u^{out} \rightarrow M \cup \{null\}$
 - ▶ δεδομένης της τρέχουσας κατάστασης
 - ▶ δημιουργεί κάποια μηνύματα για τις γειτονικές διεργασίες
- ▶ Διαθέτει μία συνάρτηση αλλαγής κατάστασης $trans_u : states_u \times (M \cup \{null\})^{nbrs_u} \rightarrow states_u$
 - ▶ δεδομένης της τρέχουσας κατάστασης
 - ▶ τα μηνύματα που παραλήφθηκαν
 - ▶ υπολογίζει την επόμενη κατάσταση της διεργασίας



Έναρξη εκτέλεσης, Βήματα και Γύρος

- ▶ Αρχικά
 - ▶ όλες οι διεργασίες βρίσκονται σε κάποια αρχική κατάσταση
 - ▶ όλα τα κανάλια είναι άδεια
- ▶ Όλες οι διεργασίες, επαναλαμβάνουν 'συντονισμένα' τα ακόλουθα δύο βήματα:
 - 1^ο Βήμα
 1. Εφαρμογή της γεννήτριας μηνυμάτων
 2. Παραγωγή μηνυμάτων για τους εξερχόμενους γείτονες
 3. Αποστολή μηνυμάτων μέσω των αντίστοιχων καναλιών
 - 2^ο Βήμα
 1. Εφαρμογή της συνάρτησης αλλαγής κατάσταση
 2. Διαγραφή όλων των μηνυμάτων από τα κανάλια.
- ▶ Ο συνδυασμός των δύο βημάτων ονομάζεται **γύρος**



Μέτρηση πολυπλοκότητας

- ▶ Σχεδιασμός Συστήματος
 - ▶ Ορισμός Ελάχιστων Απαιτήσεων
 - ▶ Επιλογή κατάλληλου κατανεμημένου αλγόριθμου
 - ▶ Πως μπορούμε να μετρήσουμε την απόδοση;

Χρονική πολυπλοκότητα

Το πλήθος των γύρων που απαιτούνται για να παραχθούν όλες οι ζητούμενες έξοδοι, ή μέχρι να τερματιστούν όλες οι διεργασίες (δηλ. να βρεθούν σε μια τερματική κατάσταση).

Πολυπλοκότητα επικοινωνίας

Ο συνολικός αριθμός μη μηδενικών μηνυμάτων (δηλ. δεν προσαμετρώνται τα null μηνύματα) που αποστέλλονται.



Σύνοψη 4^{ης} Διάλεξης

Προηγούμενο Μάθημα

Προηγούμενο Μάθημα

Σύγχρονα Κατανεμημένα Συστήματα

Πρόβλημα Συναίνεσης

Ορισμός Προβλήματος

Σφάλματα Επικοινωνίας

Σφάλματα Τερματισμού

Σύνοψη Μαθήματος

Σύνοψη Μαθήματος

2^η Άσκηση

Επόμενο Μάθημα



Γενικά

Πρόβλημα Συναίνεσης

Σε ένα σύγχρονο δίκτυο G , η συναίνεση απαιτεί την κοινή επιλογή μιας μοναδικής τιμής από όλες τις διεργασίες του συστήματος. Όταν οι διεργασίες καταλήξουν σε μια κοινά αποδεκτή απόφαση, όλες οι διεργασίες τερματίζουν.

Οι διεργασίες

- ▶ ξεκινούν με μία αρχική τιμή, την επεξεργάζονται και εξάγουν μια τελική τιμή
- ▶ εκτελούν έναν κατανεμημένο αλγόριθμο συναίνεσης
- ▶ αποφασίζουν από κοινού μια μοναδική τιμή
 - ▶ Η απόφαση είναι 'κοινή' – π.χ. μια διεργασία με τιμή 'ναί', πως μπορεί να δεχτεί μια 'κοινή' απόφαση 'όχι';



Ενοποίηση Βάσεων Δεδομένων

Για λόγους ανεκτικότητας ασφαλιμάτων και δυνατότητα κλιμάκωσης, το σύστημα χρησιμοποιεί μια φάρμα διακομιστών (με n βάσεις δεδομένων). Σε τακτά χρονικά διαστήματα, οι διακομιστές ενοποιούν τα δεδομένα. Στην συνέχεια, ελέγχουν τα δεδομένα και αποφασίζουν κατά πόσο η ενοποίηση ήταν επιτυχής.

- ▶ Κάθε μονάδα χειρίζεται μια βάση δεδομένων – ορισμένες χρονικές στιγμές δεν έχουν κοινά δεδομένα
- ▶ **Στόχος Αυξημένης Αξιοπιστίας:** Οι μονάδες που λειτουργούν σωστά αναλάβουν το έργο αυτών που βγήκαν εκτός λειτουργίας
- ▶ **Στόχος Δυνατότητα Κλιμάκωσης:** Οι ρυθμοί εκτέλεσης των διεργασιών διατηρούνται στα ίδια επίπεδα ανεξάρτητα από πιθανές αυξήσεις των απαιτήσεων των χρηστών



Υποθέτουμε ένα δίκτυο από n διεργασίες, συνδεδεμένες από ένα μη-κατευθυνόμενο γράφημα, όπου κάθε διεργασία γνωρίζει τη δομή του γραφήματος.

Κάθε διεργασία u δέχεται ως είσοδο μια τιμή i_u από το σύνολο S , δηλ. $i_u \in S$

Ένας αλγόριθμος λύνει το πρόβλημα συναίνεσης εφόσον πληροί τις παρακάτω προδιαγραφές:

1. **Συμφωνία:** Κανένα ζεύγος διεργασιών δεν αποφασίζει διαφορετικές τιμές εξόδου, δηλ. $\exists u, v : o_u \neq o_v$
2. **Εγκυρότητα:** Αν όλες οι διεργασίες αρχίζουν με την ίδια τιμή $i \in S$ δηλ. $\forall u \in [1, n] : i_u = i$, τότε η τιμή i είναι η μόνη πιθανή τελική απόφαση, δηλ. $\forall u \in [1, n] : o_u = i$
3. **Τερματισμός:** Όλες οι διαδικασίες τελικά αποφασίζουν



Μια απλή λύση

Αλγόριθμος Συναίνεσης SimpleConsensus

Κάθε διεργασία $u \in [1, n]$ διατηρεί μια λίστα l_u με ζεύγη από **ταυτότητες** και **τιμές εισόδου**, η οποία αρχικά περιέχει ένα μόνο ζεύγος, την ταυτότητα της u και την τιμή εισόδου $i_u \in S$. Σε κάθε γύρο, οι διεργασίες εκπέμπουν την λίστα l σε όλους τους γείτονες. Μόλις λάβουν μία λίστα l_v από κάποιον γείτονα v , την ενοποιούν με την δικιά τους. Μετά από $\delta + 1$ γύρους, όλες οι διεργασίες διατηρούν μια λίστα που περιέχει ένα ζεύγος (u, i_u) για κάθε διεργασία του συστήματος. Εφαρμόζουν τους κανόνες συναίνεσης και τερματίζουν επιστρέφοντας την κοινή τιμή εξόδου $o \in S$.

- ▶ Κάθε διεργασία γνωρίζει τη δομή του γραφήματος G
- ▶ Ο αλγόριθμος λύνει το πρόβλημα της συναίνεσης

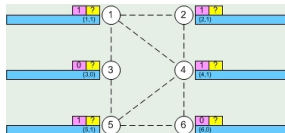


Παράδειγμα Εκτέλεσης Αλγόριθμου SimpleConsensus

Έστω ένα σύγχρονο γενικό δίκτυο όπου $n = 6$ και $\delta = 2$.

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)
- ▶ Ο κανόνας συναίνεσης βασίζεται σε απλή πλειοψηφία

Γενικό Δίκτυο

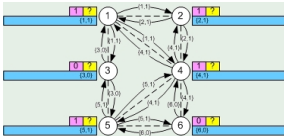


Παράδειγμα Εκτέλεσης Αλγόριθμου SimpleConsensus

Έστω ένα σύγχρονο γενικό δίκτυο όπου $n = 6$ και $\delta = 2$.

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)
- ▶ Ο κανόνας συναίνεσης βασίζεται σε απλή πλειοψηφία

1ος Γύρος – αποστολή μηνυμάτων

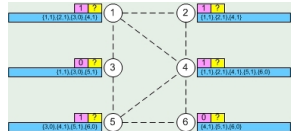


Παράδειγμα Εκτέλεσης Αλγόριθμου SimpleConsensus

Έστω ένα σύγχρονο γενικό δίκτυο όπου $n = 6$ και $\delta = 2$.

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)
- ▶ Ο κανόνας συναίνεσης βασίζεται σε απλή πλειοψηφία

1ος Γύρος – επεξεργασία

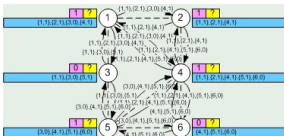


Παράδειγμα Εκτέλεσης Αλγόριθμου SimpleConsensus

Έστω ένα σύγχρονο γενικό δίκτυο όπου $n = 6$ και $\delta = 2$.

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)
- ▶ Ο κανόνας συναίνεσης βασίζεται σε απλή πλειοψηφία

2ος Γύρος – αποστολή μηνυμάτων

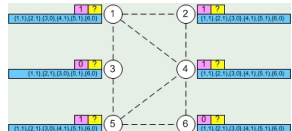


Παράδειγμα Εκτέλεσης Αλγόριθμου SimpleConsensus

Έστω ένα σύγχρονο γενικό δίκτυο όπου $n = 6$ και $\delta = 2$.

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)
- ▶ Ο κανόνας συναίνεσης βασίζεται σε απλή πλειοψηφία

2ος Γύρος – επεξεργασία

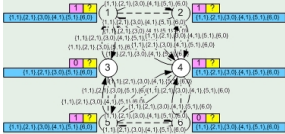


Παράδειγμα Εκτέλεσης Αλγόριθμου SimpleConsensus

Έστω ένα σύγχρονο γενικό δίκτυο όπου $n = 6$ και $\delta = 2$.

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)
- ▶ Ο κανόνας συναίνεσης βασίζεται σε απλή πλειοψηφία

3ος Γύρος – αποστολή μηνυμάτων

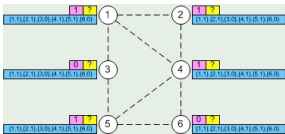


Παράδειγμα Εκτέλεσης Αλγόριθμου SimpleConsensus

Έστω ένα σύγχρονο γενικό δίκτυο όπου $n = 6$ και $\delta = 2$.

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)
- ▶ Ο κανόνας συναίνεσης βασίζεται σε απλή πλειοψηφία

3ος Γύρος – επεξεργασία

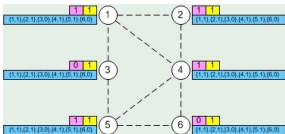


Παράδειγμα Εκτέλεσης Αλγόριθμου SimpleConsensus

Έστω ένα σύγχρονο γενικό δίκτυο όπου $n = 6$ και $\delta = 2$.

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)
- ▶ Ο κανόνας συναίνεσης βασίζεται σε απλή πλειοψηφία

3ος Γύρος – απόφαση



Χαρακτηριστικά του Αλγόριθμου SimpleConsensus

Σε ένα σύγχρονο δίκτυο G με n διεργασίες και m κανάλια

- ▶ Στο τέλος του γύρου δ κάθε διεργασία $u \in [1, n]$ διατηρεί μια λίστα $l_u = \{(1, i_1), (2, b_2), \dots, (n, i_n)\}$
- ▶ Όλες οι διεργασίες έχουν κοινές λίστες, δηλ. $\forall u \in [1, n] : l_u = l$
- ▶ Η χρονική πολυπλοκότητα είναι $\mathcal{O}(\text{diam}(G))$
- ▶ Η πολυπλοκότητα επικοινωνίας είναι $\mathcal{O}(\text{diam}(G) \cdot m)$
- ▶ Η πολυπλοκότητα επικοινωνίας σε bit είναι $\mathcal{O}(\text{diam}(G) \cdot n \cdot m)$



Παρουσία σφαλμάτων

Τι συμβαίνει όταν κατά την εκτέλεση ενός κατανεμημένου αλγορίθμου

1. παρουσιάζονται σφάλματα κατά την αποστολή μηνυμάτων
2. σφάλματα που παρουσιάζονται στις υπολογιστικές μονάδες (στους επεξεργαστές)

Υπό την παρουσία σφαλμάτων μπορούμε

- ▶ να εγγυηθούμε την ορθότητα του αλγορίθμου SimpleConsensus ;
- ▶ να εντοπίσουμε μια αποτυχία;
- ▶ να αντιμετωπίσουμε μια αποτυχία;



Μοντέλο Σφαλμάτων Επικοινωνίας

Εξετάζουμε την περίπτωση όπου

- ▶ κατά την εκτέλεση ενός κατανεμημένου αλγορίθμου
- ▶ παρουσιάζονται σφάλματα κατά την αποστολή μηνυμάτων

Σφάλμα Επικοινωνίας

Το δίκτυο επικοινωνίας που συνδέει τις μονάδες ενός κατανεμημένου συστήματος μπορεί να αποτύχει κατά την αποστολή ενός μηνύματος μέσω ενός (ελαττωματικού) καναλιού. Η παράδοση των μηνυμάτων που έχουν σταλεί δεν είναι εγγυημένη. Υποθέτουμε ότι ένας αριθμός μηνυμάτων που θα αποσταλούν κατά την εκτέλεση ενός κατανεμημένου αλγορίθμου, δεν θα παραδοθούν με επιτυχία.



Παράδειγμα

Συντονισμένη Επίθεση

Δύο στρατηγοί διοικούν τον **κόκκινο στρατό** και τον **πράσινο στρατό**. Θέλουν να συντονίσουν την επίθεση κατά του **μυλε στρατού**. Εάν επιτεθούν μόνοι τους, ο **μυλε στρατός** θα τους νικήσει. Οι στρατηγοί πρέπει να συμφωνήσουν αν θα κάνουν μαζί επίθεση, βασισμένοι στις τοπικές τους εκτιμήσεις για την άποψη του άλλου στρατηγού

- ▶ Πρόβλημα συναίνεσης σε ένα σύστημα με $n = 2$ διεργασίες
- ▶ Οι πιθανές τιμές εισόδου/εξόδου μπορεί να είναι 'ναι' και 'όχι' - δηλ. $S = \{ \text{"ναι"}, \text{"όχι"} \}$



Συντονισμένη Επίθεση (1)

Θεωρούμε ότι οι δύο στρατηγοί συντονίζουν την επίθεση όταν ικανοποιούνται οι τρεις συνθήκες:

1. **Συμφωνία:** Οι στρατηγοί u , v παίρνουν την ίδια απόφαση, δηλ. $\alpha_u = \alpha_v$
2. **Εγκυρότητα:**
 - ▶ Αν η αρχική άποψη **κάθε** στρατηγού είναι 'όχι', τότε η μοναδική αποδεκτή απόφαση είναι το 'όχι'
 - ▶ Αν η αρχική άποψη **κάθε** στρατηγού είναι 'ναι', και δεν χαθεί κανένα μήνυμα, τότε η μοναδική αποδεκτή απόφαση είναι το 'ναι'
3. **Τερματισμός:** Οι δύο στρατηγοί τελικά αποφασίζουν



Συνθήκη Εγκυρότητας

- ▶ Αν η αρχική άποψη **όλων** των στρατηγών είναι 'όχι', τότε η μόνη δυνατή απόφαση είναι το 'όχι'
- ▶ Αν η αρχική άποψη **όλων** των στρατηγών είναι 'ναι', και δεν χαθεί κανένα μήνυμα, τότε η μόνη δυνατή απόφαση είναι το 'ναι'

Η συνθήκη της **εγκυρότητας** είναι 'εύκολη'

- ▶ Οι διεργασίες μπορούν να αποφασίσουν 'ναι' ακόμα και αν μόνο μια διεργασία ξεκινήσει με 'ναι'
- ▶ Οι διεργασίες μπορούν να αποφασίσουν 'όχι' ακόμα και αν όλες οι διεργασίες ξεκινήσουν με 'ναι' αλλά χαθεί ένα μήνυμα



Έστω ότι υπάρχει ένας αλγόριθμος \mathcal{A}

- ▶ λύνει το πρόβλημα της Συντονισμένης Επίθεσης
- ▶ υπό την παρουσία σφαλμάτων επικοινωνίας
- ▶ έχει 2 αρχικές καταστάσεις για κάθε τιμή εισόδου, δηλ. $\forall u \in [1, n], |start_u| = 2$
- ▶ για μια συγκεκριμένη ανάθεση αρχικών καταστάσεων και επιτυχής ανταλλαγή συγκεκριμένων μηνυμάτων, υπάρχει μόνο μια δυνατή εκτέλεση
- ▶ σε κάθε γύρο, όλες οι διεργασίες στέλνουν ένα μήνυμα – μπορούν να στείλουν null



Εξέταση αλγόριθμου \mathcal{A} (1)

Έστω μια εκτέλεση ϵ του \mathcal{A}

- ▶ οι δύο διεργασίες ξεκινάνε με αρχική τιμή 'ναι', δηλ. $i_1 = i_2 = \text{"ναι"}$
- ▶ όλα τα μηνύματα παραδίδονται

Σύμφωνα με την **συνθήκη τερματισμού**

- ▶ υπάρχει κάποιος γύρος γ όπου και οι δύο διεργασίες θα έχουν αποφασίσει

Σύμφωνα με την **συνθήκη εγκυρότητας**

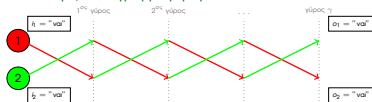
- ▶ οι δύο διεργασίες αποφασίζουν 'ναι', δηλ. $o_1 = o_2 = \text{"ναι"}$



Εξέταση αλγόριθμου \mathcal{A} (2)

Έστω η εκτέλεση ϵ_1 , που προκύπτει από την ϵ του \mathcal{A} , όπου μετά τον γύρο γ όλα τα μηνύματα χάνονται

Εκτέλεση ϵ_1 – **διάγραμμα μηνυμάτων**



Σε ένα διάγραμμα μηνυμάτων

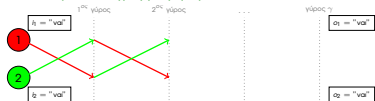
- ▶ τα τόξα αντιπροσωπεύουν αποστολές μηνυμάτων που ολοκληρώνονται επιτυχώς
- ▶ οι αποστολές που αντιμετώπισαν σφάλμα επικοινωνίας δεν εμφανίζονται



Εξέταση αλγόριθμου \mathcal{A} (5)

Συνεχίζοντας με τον ίδιο τρόπο, καταλήγουμε στην εκτέλεση ϵ' , όπου τελικά δεν παραδίδεται κανένα μήνυμα

Εκτέλεση ϵ' – διάγραμμα μηνυμάτων



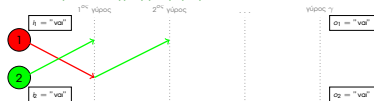
- ▶ Αφού ο γύρος γ είναι ένας πεπερασμένος αριθμός, σε πεπερασμένο αριθμό βημάτων θα έχουμε την εκτέλεση ϵ'
- ▶ Οι δύο στρατηγικοί ξεκινούν με αρχικές απόψεις 'να'
- ▶ Αποφασίζουν και οι δύο 'ναι' χωρίς να ανταλλάξουν κανένα μήνυμα



Εξέταση αλγόριθμου \mathcal{A} (5)

Συνεχίζοντας με τον ίδιο τρόπο, καταλήγουμε στην εκτέλεση ϵ' , όπου τελικά δεν παραδίδεται κανένα μήνυμα

Εκτέλεση ϵ' – διάγραμμα μηνυμάτων



- ▶ Αφού ο γύρος γ είναι ένας πεπερασμένος αριθμός, σε πεπερασμένο αριθμό βημάτων θα έχουμε την εκτέλεση ϵ'
- ▶ Οι δύο στρατηγικοί ξεκινούν με αρχικές απόψεις 'να'
- ▶ Αποφασίζουν και οι δύο 'ναι' χωρίς να ανταλλάξουν κανένα μήνυμα



Εξέταση αλγόριθμου \mathcal{A} (5)

Συνεχίζοντας με τον ίδιο τρόπο, καταλήγουμε στην εκτέλεση ϵ' , όπου τελικά δεν παραδίδεται κανένα μήνυμα

Εκτέλεση ϵ' – διάγραμμα μηνυμάτων



- ▶ Αφού ο γύρος γ είναι ένας πεπερασμένος αριθμός, σε πεπερασμένο αριθμό βημάτων θα έχουμε την εκτέλεση ϵ'
- ▶ Οι δύο στρατηγικοί ξεκινούν με αρχικές απόψεις 'να'
- ▶ Αποφασίζουν και οι δύο 'ναι' χωρίς να ανταλλάξουν κανένα μήνυμα



Εξέταση αλγόριθμου \mathcal{A} (5)

Συνεχίζοντας με τον ίδιο τρόπο, καταλήγουμε στην εκτέλεση ϵ' , όπου τελικά δεν παραδίδεται κανένα μήνυμα

Εκτέλεση ϵ' – διάγραμμα μηνυμάτων



- ▶ Αφού ο γύρος γ είναι ένας πεπερασμένος αριθμός, σε πεπερασμένο αριθμό βημάτων θα έχουμε την εκτέλεση ϵ'
- ▶ Οι δύο στρατηγικοί ξεκινούν με αρχικές απόψεις 'να'
- ▶ Αποφασίζουν και οι δύο 'ναι' χωρίς να ανταλλάξουν κανένα μήνυμα



Εξέταση αλγόριθμου \mathcal{A} (5)

Συνεχίζοντας με τον ίδιο τρόπο, καταλήγουμε στην εκτέλεση ϵ' , όπου τελικά δεν παραδίδεται κανένα μήνυμα

Εκτέλεση ϵ' – διάγραμμα μηνυμάτων



- Αφού ο γύρος γ είναι ένας πεπερασμένος αριθμός, σε πεπερασμένο αριθμό βημάτων θα έχουμε την εκτέλεση ϵ'
- Οι δύο στρατηγοί ξεκινούν με αρχικές απόψεις 'ναί'
- Αποφασίζουν και οι δύο 'ναί' χωρίς να ανταλλάξουν κανένα μήνυμα



Εξέταση αλγόριθμου \mathcal{A} (6)

Έστω η εκτέλεση ϵ'' που προκύπτει από την ϵ' , όταν ο **πράσινος στρατηγός** έχει αρχική άποψη 'όχι'

Εκτέλεση ϵ'' – διάγραμμα μηνυμάτων



- Η απόφαση του **πράσινου στρατηγού** στο τέλος του γύρου γ μπορεί να είναι διαφορετική στην ϵ'' από την ϵ'
- Όμως ο **κόκκινος στρατηγός** δεν το γνωρίζει – η κατάσταση της 1 είναι η ίδια στην ϵ'' από την ϵ'
- Λόγω της **συνθήκης συμφωνίας**: η 2 αποφασίζει το ίδιο με την 1



Εξέταση αλγόριθμου \mathcal{A} (7)

Έστω η εκτέλεση ϵ''' που προκύπτει από την ϵ'' , όταν ο **κόκκινος στρατηγός** έχει αρχική άποψη 'όχι'

Εκτέλεση ϵ''' – διάγραμμα μηνυμάτων



- Η απόφαση του **κόκκινου στρατηγού** στο τέλος του γύρου γ μπορεί να είναι διαφορετική στην ϵ''' από την ϵ''
- Όμως ο **πράσινος στρατηγός** δεν το γνωρίζει – η κατάσταση της 2 είναι η ίδια στην ϵ''' από την ϵ''
- Λόγω της **συνθήκης συμφωνίας**: η 1 αποφασίζει το ίδιο με την 2

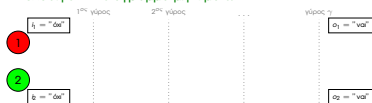


Εξέταση αλγόριθμου \mathcal{A} (8)

Άποιο

- οι δύο στρατηγοί έχουν την ίδια αρχική άποψη ('όχι')
- σύμφωνα με την **συνθήκη εγκυρότητας** η μοναδική αποδεκτή απόφαση είναι το 'όχι'

Εκτέλεση ϵ''' – διάγραμμα μηνυμάτων



Άρα ο αλγόριθμος \mathcal{A} δεν λύνει το πρόβλημα της Συντονισμένης Επίθεσης



Θεώρημα

Έστω σύγχρονο δίκτυο G που αποτελείται από δύο διεργασίες $V = \{u, v\}$ συνδεδεμένες από μια μη-κατευθυνόμενη ακμή uv . Δεν υπάρχει αλγόριθμος που να λύνει το πρόβλημα της συντονισμένης επίθεσης υπό την παρουσία σφαλμάτων επικοινωνίας.

- ▶ Βασικός περιορισμός στις δυνατότητες των κατανεμημένων δικτύων
- ▶ Για να ξεπεραστεί πρέπει να κάνουμε πιο ισχυρό το μοντέλο
 - ▶ Υποθέτουμε ότι ένας φραγμένος αριθμός μηνύματα χάνονται
 - ▶ Υποθέτουμε ότι τα μηνύματα χάνονται με πιθανότητα p
- ▶ Για να ξεπεραστεί πρέπει να κάνουμε πιο ασθενές το πρόβλημα
 - ▶ Επιτρέπουμε την παραβίαση της συνθήκης της συμφωνίας υπό ορισμένες συνθήκες
 - ▶ Επιτρέπουμε την παραβίαση της συνθήκης της εγκρότητας υπό ορισμένες συνθήκες



Εξετάζουμε την περίπτωση όπου

- ▶ κατά την εκτέλεση ενός κατανεμημένου αλγορίθμου
- ▶ σφάλματα που παρουσιάζονται στις υπολογιστικές μονάδες (στους επεξεργαστές)
- ▶ εμφανίζονται το πολύ σ σφάλματα

Σφάλμα Τερματισμού

Κάποια (ελαττωματική) μονάδα του κατανεμημένου συστήματος μπορεί να αποτύχει κατά την εκτέλεση μια διεργασίας. Ένα σφάλμα τερματισμού μπορεί να παρουσιαστεί σε οποιοδήποτε σημείο της εκτέλεσης μιας διεργασίας. Η διεργασία μπορεί να τερματίσει ξαφνικά κατά την παραγωγή μηνυμάτων, οπότε να σταλεί μόνο ένα μέρος των εξερχόμενων μηνυμάτων.



Αλγόριθμος Συνάιησης FloodSet

Κάθε διεργασία $u \in [1, n]$ διατηρεί μια λίστα l_u με **πμέ εισόδου**, η οποία αρχικά περιέχει ένα μόνο την τιμή εισόδου $l_u \in S$ της u , $l_u = \{l_u\}$. Σε κάθε γύρο, οι διεργασίες εκπέμπουν την λίστα l σε όλους τους γείτονες. Μόλις λάβουν μία λίστα l , από κάποιον γείτονα v , την ενοποιούν με την δικιά τους. Μετά από $\sigma + 1$ γύρους, αν η λίστα l_u περιέχει μία μόνο τιμή ($|l_u| = 1$) η διεργασία u αποφασίζει την τιμή που περιέχει η λίστα - αλλιώς αποφασίζει την προκαθορισμένη τιμή $p \in S$.

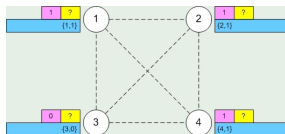
- ▶ Το γράφημα G είναι πλήρες
- ▶ Γνωρίζουν το συνολικό αριθμό σφαλμάτων σ
- ▶ Συμβολίζουμε με $l_u(\gamma)$ την τιμή της λίστας l_u που διατηρεί η διεργασία u τον γύρο γ



Έστω ένα σύγχρονο πλήρες δίκτυο όπου $n = 4$ και $\sigma = 2$

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)

Πλήρως Συνδεδεμένο Δίκτυο

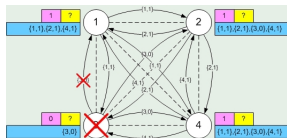


Παράδειγμα Εκτέλεσης Αλγόριθμου FloodSet

Έστω ένα σύγχρονο πλήρες δίκτυο όπου $n = 4$ και $\sigma = 2$

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)

1ος Γύρος – σφάλμα στην διεργασία 3

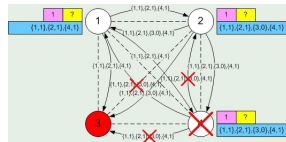


Παράδειγμα Εκτέλεσης Αλγόριθμου FloodSet

Έστω ένα σύγχρονο πλήρες δίκτυο όπου $n = 4$ και $\sigma = 2$

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)

2ος Γύρος – σφάλμα στην διεργασία 4

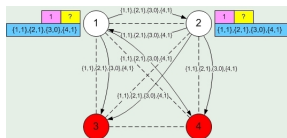


Παράδειγμα Εκτέλεσης Αλγόριθμου FloodSet

Έστω ένα σύγχρονο πλήρες δίκτυο όπου $n = 4$ και $\sigma = 2$

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)

3ος Γύρος – κανένα σφάλμα

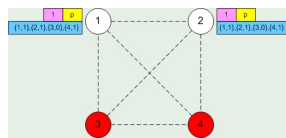


Παράδειγμα Εκτέλεσης Αλγόριθμου FloodSet

Έστω ένα σύγχρονο πλήρες δίκτυο όπου $n = 4$ και $\sigma = 2$

- ▶ Οι διεργασίες έχουν μια τιμή εισόδου (ροζ κουτί)
- ▶ Οι διεργασίες διατηρούν μια λίστα (μπλέ κουτί)
- ▶ Οι διεργασίες έχουν μια τιμή εξόδου (κίτρινο κουτί)

3ος Γύρος – απόφαση



Χαρακτηριστικά του Αλγόριθμου FloodSet (1)

Λήμμα (FloodSet.1)

Αν καμία διεργασία δεν αντιμετωπίσει ένα σφάλμα τερματισμού κατά τον γύρο γ , $1 \leq \gamma \leq \sigma + 1$, τότε για κάθε ζεύγος διεργασιών u, v , που δεν έχει αντιμετωπίσει κάποιο σφάλμα μετά από γ γύρους, $l_u(\gamma) = l_v(\gamma)$.

Απόδειξη: Έστω C το σύνολο των διεργασιών που δεν έχουν αντιμετωπίσει σφάλμα μετά από $\gamma - 1$ γύρους.

Έστω ότι κανένα σφάλμα δεν εμφανίζεται τον γύρο γ .

Τότε $\forall u \in C$ θα στείλει την $l_u(\gamma - 1)$ σε όλες τις άλλες.

Άρα στον γύρο γ ,

$$\forall u \in C, l_u(\gamma) = \cup_{v \in C} l_v(\gamma - 1)$$



Χαρακτηριστικά του Αλγόριθμου FloodSet (2)

Λήμμα (FloodSet.2)

Έστω ότι για κάθε ζεύγος διεργασιών u, v , που δεν έχει αντιμετωπίσει κάποιο σφάλμα μετά από γ γύρους, $l_u(\gamma) = l_v(\gamma)$. Τότε σε κάθε γύρο $\gamma', \gamma \leq \gamma' \leq \sigma + 1$, για κάθε ζεύγος διεργασιών u, v που δεν έχει αντιμετωπίσει κάποιο σφάλμα μετά από γ' γύρους, $l_u(\gamma') = l_v(\gamma')$.

Απόδειξη: Όλες οι διεργασίες που δεν έχει αντιμετωπίσει κάποιο σφάλμα μετά από γ γύρους έχουν στείλει την τιμή τους

Οι διεργασίες που δεν έχουν αντιμετωπίσει κάποιο σφάλμα μετά από γ γύρους έχουν ίδιες λίστες

Καμία άλλη ενεργή διεργασία δεν υπάρχει – μετά τον γύρο γ δεν θα εμφανιστεί κάποια νέα τιμή

Άρα η τιμή της $l_u, \forall u \in C$ δεν θα αλλάξει σε επόμενους γύρους



Χαρακτηριστικά του Αλγόριθμου FloodSet (3)

Λήμμα (FloodSet.3)

Έστω ένα ζεύγος διεργασιών u, v , που δεν έχει αντιμετωπίσει κάποιο σφάλμα μετά από $\sigma + 1$ γύρους, τότε $l_u(\sigma + 1) = l_v(\sigma + 1)$ στο τέλος του γύρου $\sigma + 1$.

Απόδειξη: Εφόσον παρουσιάζονται το πολύ σ σφάλματα

- ▶ υπάρχει κάποιος γύρος γ , $1 \leq \gamma \leq \sigma + 1$ όπου καμία διεργασία δεν αντιμετωπίζει σφάλμα

Σύμφωνα με το Λήμμα FloodSet.1 $l_u(\gamma) = l_v(\gamma)$ για κάθε u, v που δεν έχουν αντιμετωπίσει σφάλμα μετά από γ γύρους

Σύμφωνα με το Λήμμα FloodSet.2 $l_u(\sigma + 1) = l_v(\sigma + 1)$ για κάθε u, v που δεν έχουν αντιμετωπίσει σφάλμα μετά από $\sigma + 1$ γύρους



Χαρακτηριστικά του Αλγόριθμου FloodSet (4)

Θεώρημα

Ο αλγόριθμος FloodSet λύνει το πρόβλημα της συναίνεσης

Απόδειξη:

Η **συνθήκη τερματισμού** ικανοποιείται – όλες οι διεργασίες που δεν αντιμετωπίσαν σφάλμα μέχρι το τέλος του γύρου $\sigma + 1$ τερματίζουν

Η **συνθήκη εγκυρότητας** ικανοποιείται –

- ▶ Αν όλες οι διεργασίες έχουν αρχική τιμή τ τότε η μοναδική τιμή που στέλνεται είναι η $\{\tau\}$
- ▶ Η λίστα l_u δεν θα αλλάξει μέχρι το τέλος του γύρου $\sigma + 1$

Η **συνθήκη συμφωνίας** ικανοποιείται –

- ▶ Σύμφωνα με Λήμμα FloodSet.3



Χαρακτηριστικά του Αλγόριθμου FloodSet (5)

- ▶ Η χρονική πολυπλοκότητα είναι $\sigma + 1$ γύροι
- ▶ Η πολυπλοκότητα επικοινωνίας είναι $\mathcal{O}((\sigma + 1) \cdot n^2)$
- ▶ Το κάθε μήνυμα μπορεί να έχει μέγεθος $\mathcal{O}(n)$ bit
- ▶ Η πολυπλοκότητα επικοινωνίας είναι $\mathcal{O}((\sigma + 1) \cdot n^3)$ bit

Παραλλαγές

- ▶ Αντί για την προκαθορισμένη τιμή $p \in S$, επιλέγετε η $\min(S)$
- ▶ Οι διεργασίες στέλνουν μόνο όταν υπάρχουν αλλαγές στην λίστα, μετά την επεξεργασία των μηνυμάτων (OptFloodSet)



Επικύρωση Δοσοληψιών (1)

Πρόβλημα Επικύρωσης Δοσοληψιών

Οι διεργασίες του συστήματος, συμμετέχουν στην διεκπεραίωση μιας δοσοληψίας. Η κάθε διεργασία, σύμφωνα με τις τοπικές πληροφορίες εκφέρει μια άποψη για το αν η δοσοληψία πρέπει να επικυρωθεί – δηλαδή η διεκπεραίωση της ήταν επιτυχής. Στην συνέχεια οι διεργασίες επικοινωνούν μεταξύ τους για να πάρουν μια κοινή απόφαση – αν όλες συναινούν στην επικύρωση, τότε η δοσοληψία μπορεί να επικυρωθεί, αλλιώς όχι

- ▶ Οι πιθανές τιμές εισόδου/εξόδου μπορεί να είναι 'ναι' και 'όχι' – δηλ. $S = \{ \text{"ναι"}, \text{"όχι"} \}$
- ▶ Σε ένα σύστημα με πολλούς διακομιστές βάσεων δεδομένων, σε τακτά χρονικά διαστήματα τα δεδομένα ενοποιούνται – οι διακομιστές πρέπει να επικυρώσουν την δοσοληψία ενοποίησης



Επικύρωση Δοσοληψιών (2)

Θεωρούμε ότι οι διεργασίες λύνουν το πρόβλημα της επικύρωσης δοσοληψιών όταν ικανοποιούνται οι τρεις συνθήκες:

- 1. Συμφωνία:** Κανένα ζεύγος διεργασιών δεν αποφασίζει διαφορετικές τιμές εξόδου, δηλ. $\nexists u, v : o_u \neq o_v$
- 2. Εγκυρότητα:**
 - ▶ Αν υπάρχει μια διεργασία u με τιμή $l_u = \text{"όχι"}$ τότε η μοναδική αποδεκτή απόφαση είναι το 'όχι'
 - ▶ Αν $\forall u : l_u = \text{"ναι"}$ και δεν παρουσιαστούν σφάλματα, τότε η μοναδική αποδεκτή απόφαση είναι το 'ναι'
- 3. Τερματισμός:**
 - ▶ **χαλαρός** – εάν δεν παρουσιαστούν σφάλματα, όλες οι διεργασίες αποφασίζουν
 - ▶ **ισχυρός** – όλες οι διεργασίες που δεν αντιμετώπισαν κάποιο σφάλμα αποφασίζουν



Ο Αλγόριθμος TwoPhaseCommit (1)

Αλγόριθμος TwoPhaseCommit

Η εκτέλεση του αλγορίθμου διεξάγεται σε δύο φάσεις:

1^η φάση – Κάθε διεργασία $u_i, j \in (1, n]$, στέλνει στην διεργασία u_1 την αρχική τιμή l_i . Στη συνέχεια, αν $l_i = \text{"όχι"}$, αποφασίζει 'όχι'. Η διεργασία u_1 συλλέγει όλα τα μηνύματα, συνοπολογίζει την δικιά της αρχική τιμή l_1 και αποφασίζει $o_1 = \text{"ναι"}$ μόνο όταν λάβει την τιμή 'ναι' από όλες τις διεργασίες, αλλιώς αποφασίζει $o_1 = \text{"όχι"}$

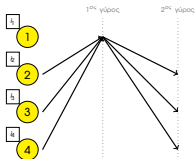
2^η φάση – Η διεργασία u_1 στέλνει την τιμή o_1 σε όλες τις διεργασίες. Όποιες από αυτές λάβουν το μήνυμα και δεν έχουν ήδη αποφασίσει απο το γύρο 1, αποφασίζουν το o_1

- ▶ Υπάρχει μια προ-επιλεγμένη διεργασία (u_1)



Ο Αλγόριθμος TwoPhaseCommit (2)

Εκτέλεση του TwoPhaseCommit



- ▶ Διάγραμμα μηνυμάτων
- ▶ Δεν παρουσιάζεται σφάλμα τερματισμού στην εκτέλεση του παραδείγματος
- ▶ Συνήθως πριν από τον 1ο γύρο – η διεργασία 1 'ζητάει' τις απόψεις των άλλων διεργασιών



Χαρακτηριστικά του Αλγόριθμου TwoPhaseCommit

- ▶ Ο αλγόριθμος TwoPhaseCommit λύνει το πρόβλημα της επικύρωσης δεσολησιών υπό την χαλαρή συνθήκη τερματισμού
- ▶ Αν παρουσιαστεί σφάλμα στην u_0 λίγο πριν πάρει την απόφαση στο τέλος της 1^{ης} φάσης – κανείς δεν θα μάθει την απόφαση της u_0
- ▶ Η χρονική πολυπλοκότητα είναι σταθερή – 2 γύροι
- ▶ Η πολυπλοκότητα επικοινωνίας είναι $O(n)$



Ο Αλγόριθμος ThreePhaseCommit (1)

Αλγόριθμος ThreePhaseCommit

Η εκτέλεση του αλγορίθμου ξεκινάει με τρεις φάσεις:

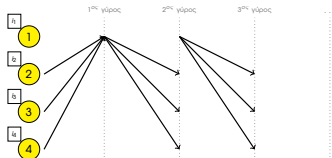
- 1^η φάση** – Κάθε διεργασία $u_j, j \in (1, n]$, στέλνει στην διεργασία u_1 την αρχική τιμή l_j . Στη συνέχεια, αν $l_j = "$ όχι", αποφασίζει 'όχι'. Η διεργασία u_1 συλλέγει τα μηνύματα, συνοπολογίζει την l_1 και γίνεται 'έτοιμη' μόνο όταν λάβει 'ναι' από όλες τις διεργασίες, αλλιώς αποφασίζει $o_1 = "$ όχι"
- 2^η φάση** – Αν η διεργασία u_1 αποφάσισε $o_1 = "$ όχι" στέλνει στις άλλες το μήνυμα 'άκυρη', αλλιώς στέλνει 'έτοιμη'. Όποια διεργασία λάβει το μήνυμα 'άκυρη', αποφασίζει $o_u = "$ όχι". Όποια διεργασία λάβει το μήνυμα 'έτοιμη', γίνεται 'έτοιμη'. Αν η u_1 δεν έχει αποφασίσει έως τώρα, αποφασίζει $o_1 = "$ ναι".
- 3^η φάση** – Αν η διεργασία u_1 αποφάσισε $o_1 = "$ ναι" στέλνει σε όλες τις διεργασίες το μήνυμα 'ναι'. Όποια διεργασία λάβει το μήνυμα 'ναι', αποφασίζει $o_u = "$ ναι".

- ▶ Υπάρχει μια προ-επιλεγμένη διεργασία (u_1)



Ο Αλγόριθμος ThreePhaseCommit (2)

Εκτέλεση του ThreePhaseCommit – διάγραμμα μηνυμάτων



Κάθε διεργασία βρίσκεται σε μια και μόνο μια κατάσταση:

1. $κ_0$ – έχει αποφασίσει $o_u = "$ όχι"
2. $κ_1$ – έχει αποφασίσει $o_u = "$ ναι"
3. 'έτοιμη' – δεν έχει αποφασίσει αλλά είναι 'έτοιμη'
4. 'άγνωστη' – δεν έχει αποφασίσει και δεν είναι 'έτοιμη'



Ο Αλγόριθμος ThreePhaseCommit (3)

Λήμμα (ThreePhaseCommit.1)

Μετά απο τρεις γύρους του αλγόριθμου ThreePhaseCommit τα ακόλουθα ισχύουν

1. Αν κάποια διεργασία βρίσκεται στην κατάσταση κ_1 ή 'έτοιμη', τότε οι αρχικές τιμές όλων των διεργασιών είναι 'ναι'
2. Αν κάποια διεργασία βρίσκεται στην κατάσταση κ_0 , τότε καμία διεργασία δεν είναι στην κατάσταση κ_1 και καμία ενεργή διεργασία δεν είναι στην κατάσταση 'έτοιμη'
3. Αν κάποια διεργασία βρίσκεται στην κατάσταση κ_1 , τότε καμία διεργασία δεν είναι στην κατάσταση κ_0 , και καμία ενεργή διεργασία δεν είναι στην κατάσταση 'άγνωστη'.



Ο Αλγόριθμος ThreePhaseCommit (4)

Λήμμα (ThreePhaseCommit.2)

Μετά απο τρεις γύρους του αλγόριθμου ThreePhaseCommit τα ακόλουθα ισχύουν

1. Η **συνθήκη συμφωνίας**
2. Η **συνθήκη ορθότητας**
3. Αν η διεργασία u_1 δεν παρουσίασε σφάλμα, τότε όλες οι ενεργές διεργασίες έχουν αποφασίσει.

Απόδειξη: Η συνθήκη συμφωνίας προκύπτει από το Λήμμα ThreePhaseCommit.1.

Η συνθήκη ορθότητας προκύπτει

- ▶ μερικώς απο το Λήμμα ThreePhaseCommit.1 -- αν κάποια διεργασία αρχίσει με 'όχι', τότε όλες αποφασίζουν 'όχι'
- ▶ εξέταση των υπόλοιπων περιπτώσεων



Ο Αλγόριθμος ThreePhaseCommit (5)

Αν η διεργασία u_1 δεν παρουσιάσει σφάλμα, τότε όλες οι ενεργές διεργασίες έχουν αποφασίσει.

- ▶ Η διεργασία u_1 αποφασίζει εφόσον δεν παρουσίασε σφάλμα
 - ▶ Μεταδίδει την απόφαση της στις άλλες διεργασίες
 - ▶ Όσες λάβουν το μήνυμα και δεν παρουσιάσουν σφάλμα, αποφασίζουν
-
- ▶ Οι τρεις πρώτες φάσεις δεν είναι αρκετές για να λυθεί το πρόβλημα υπό την **ισχυρή συνθήκη τερματισμού**.
 - ▶ Αν η διεργασία u_1 παρουσιάσει σφάλμα, μπορεί κάποιες άλλες διεργασίες να παραμείνουν στην κατάσταση 'άγνωστη'.
 - ▶ Οι διεργασίες εκτελούν ένα **'πρωτόκολλο τερματισμού'**.



Ο Αλγόριθμος ThreePhaseCommit (6)

Αλγόριθμος ThreePhaseCommit

4^η φάση -- Οι διεργασίες στέλνουν την κατάσταση τους (κ_0 , κ_1 , 'έτοιμη', 'άγνωστη') στην διεργασία u_2 η οποία τα τοποθετεί σε μία συλλογή. Αν η συλλογή:

1. περιέχει έστω μία τιμή κ_0 και η u_2 δεν έχει ακόμα αποφασίσει, αποφασίζει 'όχι'
2. περιέχει μία τιμή κ_1 και η u_2 δεν έχει ακόμα αποφασίσει, αποφασίζει 'ναι'
3. όλες οι τιμές είναι 'άγνωστη' τότε η u_2 αποφασίζει 'όχι'.
4. όλες οι τιμές είναι 'άγνωστη' ή 'έτοιμη', αλλά περιέχει τουλάχιστον μία τιμή 'έτοιμη' τότε η u_2 γίνεται 'έτοιμη'



Αλγόριθμος ThreePhaseCommit

5^η φάση – Αν η διεργασία u_2 αποφάσισε $o_2 = \text{"όχι"}$ στέλνει στις άλλες το μήνυμα **'άκυρη'**, ενώ αν $o_2 = \text{"ναι"}$ στέλνει στις άλλες το μήνυμα **'έγκυρη'**. Αν η u_2 δεν έχει αποφασίσει στέλνει **'έτοιμη'**. Όποια διεργασία λάβει το μήνυμα **'άκυρη'** ή **'έγκυρη'**, αποφασίζει αντίστοιχα – όποια λάβει το μήνυμα **'έτοιμη'** γίνεται **'έτοιμη'**. Αν η u_2 δεν έχει αποφασίσει έως τώρα, αποφασίζει $o_2 = \text{"ναι"}$.

6^η φάση – Αν η διεργασία u_2 αποφάσισε $o_2 = \text{"ναι"}$ στέλνει σε όλες τις διεργασίες το μήνυμα **'έγκυρη'**. Όποια διεργασία λάβει το μήνυμα **'έγκυρη'**, και δεν έχει αποφασίσει ακόμα, θέτει $o_u = \text{"ναι"}$.

Στην συναίεση το πρωτόκολλο ακολουθεί τρεις αντίστοιχες φάσεις συντονισμού για κάθε διεργασία $u \in [3, n]$.



- ▶ Ο αλγόριθμος ThreePhaseCommit λύνει το πρόβλημα επικύρωσης δοσοληψιών και ικανοποιεί την **ισχυρή συνθήκη τερματισμού**
 - ▶ Με επαγωγή στον αριθμό των γύρων
 - ▶ Σύμφωνα με τα Λήμματα ThreePhaseCommit.1, ThreePhaseCommit.2
- ▶ Η χρονική πολυπλοκότητα είναι $3n$ γύροι $- O(n)$
- ▶ Η πολυπλοκότητα επικοινωνίας είναι $O(n^2)$



Σύνοψη 4^{ης} Διάλεξης

Προηγούμενο Μάθημα

Προηγούμενο Μάθημα

Σύγχρονα Κατανεμημένα Συστήματα

Πρόβλημα Συναίεσης

Ορισμός Προβλήματος

Σφάλματα Επικοινωνίας

Σφάλματα Τερματισμού

Σύνοψη Μαθήματος

Σύνοψη Μαθήματος

2^η Άσκηση

Επόμενο Μάθημα



Σύνοψη Μαθήματος

- ▶ Σύγχρονα Κατανεμημένα Συστήματα
- ▶ Συναίεση Υπό Την Παρουσία Σφαλμάτων
 - ▶ Σφάλματα Επικοινωνίας
 - ▶ Σφάλματα Τερματισμού
- ▶ Συντονισμός Στρατηγών
- ▶ Επικύρωση Δοσοληψιών
- ▶ Μελέτη Ορισμένων Κατανεμημένων Αλγόριθμων – Γενικά Δίκτυα



Βιβλιογραφία (1)

- ▶ Βιβλίο "Distributed Algorithms" (N.Lynch)
 1. Κεφάλαιο 5: Distributed Consensus with Link Failures
 2. Κεφάλαιο 6: Distributed Consensus with Process Failures
 3. Κεφάλαιο 7.3: The Commit Problem
- ▶ Βιβλίο "Distributed Computing Fundamentals, Simulations, and Advanced Topics" (H.Attiya, J.Welch)
 1. Κεφάλαιο 5: Fault-Tolerant Consensus -- μόνο 5.1
- ▶ Βιβλίο "Introduction to Distributed Algorithms" (G.Tel)
 1. Κεφάλαιο 13: Fault Tolerance In Distributed Systems
- ▶ Βιβλίο "Distributed Systems, Concepts and Design" (G.Coulouris, J.Dollimore, T.Kindberg)
 1. Κεφάλαιο 13: Distributed Transactions
- ▶ Βιβλίο "Distributed Systems: Principles and Paradigms" (A.Tanenbaum, M.Steen)
 1. Κεφάλαιο 7: Fault Tolerance -- Μόνο 7.1, 7.2, 7.5



1^ο Ερώτημα

Αποδείξτε την ορθότητα του αλγόριθμου SyncBFS.



2^ο Ερώτημα

Θεωρείστε ένα σύγχρονο κατανεμημένο σύστημα με n διεργασίες συνδεδεμένες μέσω ενός γενικού, μη-κατευθυνόμενου δικτύου με m κανάλια επικοινωνίας, όπου κάθε διεργασία έχει μια μοναδική ταυτότητα αλλά δεν γνωρίζει το σύνολο των διεργασιών, ούτε την τοπολογία του δικτύου. Οι διεργασίες διατηρούν μια μεταβλητή *parent* που είτε έχει την τιμή της ταυτότητας μιας γειτονικής διεργασίας, είτε είναι *null*. Σχεδιάστε έναν κατανεμημένο αλγόριθμο που να ελέγχει αν οι μεταβλητές *parent* όλων των διεργασιών σχηματίζουν ένα επικαλυπτικό δέντρο του δικτύου με ρίζα την u_0 . Οι τιμές των μεταβλητών *parent* δεν μεταβάλλονται κατά την εκτέλεση του αλγορίθμου. Περιγράψτε τον αλγόριθμό σας (δώστε ψευδο-κώδικα), αναλύστε την ορθότητα του αλγορίθμου, την χρονική πολυπλοκότητα και πολυπλοκότητα μηνυμάτων. Αποδείξτε τους ισχυρισμούς σας.



3^ο Ερώτημα

Θεωρείστε ένα σύγχρονο κατανεμημένο σύστημα με n διεργασίες συνδεδεμένες μέσω ενός γενικού δικτύου. Κάθε διεργασία έχει μια μοναδική ταυτότητα και δεν γνωρίζει το σύνολο των διεργασιών ή την τοπολογία του δικτύου. Σχεδιάστε έναν όσο το δυνατόν αποδοτικότερο αλγόριθμο για την κατασκευή ενός επικαλυπτικού δέντρου ελαχίστου-ύψους. Ορίστε τις ιδιότητες του αλγορίθμου σας και αναλύστε την ορθότητά του, καθώς και την χρονική πολυπλοκότητα και πολυπλοκότητα μηνυμάτων. Αποδείξτε τους ισχυρισμούς σας.



- ▶ Παράδοση σε 2 Δευτέρες
 - ▶ Δευτέρα, 9 Νοεμβρίου
- ▶ Ατομική Άσκηση
- ▶ Συνεισφέρει 2 βαθμούς στον τελικό βαθμό

- ▶ Βυζαντινά Σφάλματα σε Σύγχρονα Συστήματα
- ▶ Ασύγχρονα Κατανεμημένα Συστήματα
- ▶ Μοντελοποίηση Συστήματος
- ▶ Κατανεμημένοι Αλγόριθμοι σε Ασύγχρονα Συστήματα