HY486: Αρχές Κατανεμημένου Υπολογισμού

Χειμερινό Εξάμηνο 2020-2021

Δεύτερη Προγραμματιστική Εργασία

Προθεσμία Παράδοσης: 22 Δεκεμβρίου 2020

1. Γενική Περιγραφή

Στη δεύτερη προγραμματιστική εργασία καλείστε να υλοποιήσετε ένα κατανεμημένο σύστημα για αυτοματοποιημένες αποθήκες (automated warehouses), δηλαδή ένα σύστημα που διαχειρίζεται και οργανώνει τις αποθήκες μιας εταιρείας σε μια συγκεκριμένη περιοχή ή χώρα. Το σύστημα θα διαχειρίζεται λειτουργίες όπως εξυπηρέτηση παραγγελιών και διαχείριση αποθεμάτων των εμπορευμάτων. Η προγραμματιστική εργασία θα πρέπει να υλοποιηθεί στην γλώσσα C με τη χρήση του Message Passing Interface (MPI). Το MPI υπάρχει εγκατεστημένο στα μηχανήματα της σχολής.

2. Υλοποίηση

Στην εργασία αυτή θα πρέπει να υλοποιήσετε ένα σύστημα διαχείρισης παραγγελιών και αποθεμάτων κάποιας εταιρείας η οποία διαθέτει (πολλές) αυτοματοποιημένες αποθήκες. Το σύστημα αποθήκης μιας εταιρείας αποτελείται από διεργασίες εξυπηρετητές (server), μια για κάθε αποθήκη της εταιρείας και διεργασίες πελάτες (clients). Κάθε διεργασία εξυπηρετητή χειρίζεται αιτήματα παραγγελιών που αποστέλλονται από συγκεκριμένες διεργασίες πελάτες που έχουν συσχετισθεί με αυτή τη διεργασία εξυπηρετητή. Η διεργασία εξυπηρετητή είναι υπεύθυνη για το απόθεμα της αποθήκης που διαχειρίζεται. Για απλοποίηση του συστήματος θεωρούμε ότι υπάρχει μόνο ένα προϊόν, το Α.

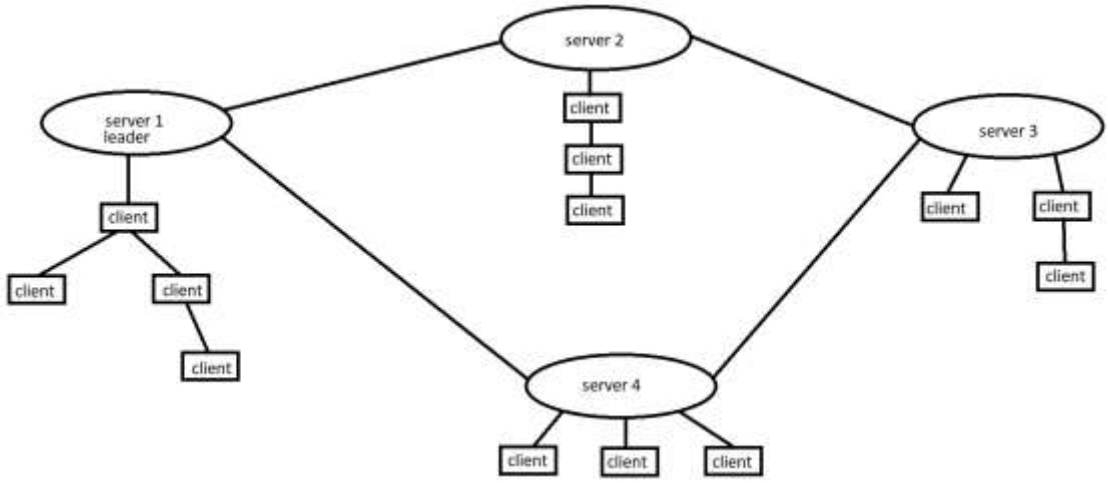
Επίσης, μια συγκεκριμένη διεργασία εξυπηρετητής είναι υπεύθυνη για την αγορά νέων προμηθειών, τις οποίες (προμήθειες) κατανέμει σε όλες τις αποθήκες.

Με βάση τα παραπάνω, το σύστημα θα πρέπει να διαμορφωθεί ως εξής:

- Στο σύστημα θα υπάρχουν διεργασίες πελάτες που ζητούν την εκτέλεση αιτημάτων και διεργασίες εξυπηρετητές που τα εξυπηρετούν.
- Η διεργασία με αναγνωριστικό (MPI rank) 0 θα παίζει το ρόλο του συντονιστή (coordinator), του οποίου μοναδικός σκοπός είναι να διαβάζει ένα testfile που θα περιέχει την περιγραφή των διαφόρων γεγονότων που πρέπει να προσομοιώνουν οι υπόλοιπες διεργασίες και να αποστέλλει αντίστοιχα μηνύματα στις κατάλληλες διεργασίες για να πραγματοποιηθεί αυτό. Άρα, η διεργασία συντονιστής δεν αποτελεί ούτε διεργασία πελάτη, ούτε διεργασία εξυπηρετητή. Η δομή του testfile, καθώς και τα γεγονότα περιγράφονται αναλυτικά στη συνέχεια.
- Υπάρχει ένας πεπερασμένος αριθμός διεργασιών εξυπηρετητών και είναι συνδεδεμένες μεταξύ τους σχηματίζοντας ένα λογικό δακτύλιο. Οι διεργασίες που θα λειτουργούν ως εξυπηρετητές θα έχουν πλήθος **NUM_SERVERS**, όπου το **NUM_SERVERS** θα δίνεται ως παράμετρος στη γραμμή εντολών κατά την εκκίνηση (δείτε Ενότητα 2.1). Στο σύστημα υπάρχει και ο ίδιος αριθμός αποθηκών καθώς αντιστοιχεί ένας εξυπηρετητής σε κάθε αποθήκη.

Η διεργασία συντονιστής θα διατηρεί πληροφορία σχετικά με τα αναγνωριστικά των υπόλοιπων εξυπηρετητών σε έναν πίνακα μεγέθους ίσου με το πλήθος, **NUM_SERVERS**, των διεργασιών εξυπηρετητών που υπάρχουν στο σύστημα.

- Κάθε διεργασία εξυπηρετητής θα διατηρεί σε μια τοπική μεταβλητή το απόθεμα του προϊόντος που αποθηκεύει.
- Μια εκ των διεργασιών εξυπηρετητών θα εκλέγεται αρχηγός (με τρόπο που περιγράφεται παρακάτω) και θα αναλαμβάνει καθήκοντα προμηθευτή. Η πληροφορία αυτή θα του γνωστοποιείται κατά την εκτέλεση του αλγορίθμου εκλογής αρχηγού.
- Οι υπόλοιπες διεργασίες στο σύστημα είναι διεργασίες πελάτες. Κάθε διεργασία πελάτη έχει συσχετισθεί με μια διεργασία εξυπηρετητή. Οι διεργασίες πελάτες ενός εξυπηρετητή σχηματίζουν ένα δένδρο, στη ρίζα του οποίου βρίσκεται ο εξυπηρετητής. Η τοπολογία του δικτύου απεικονίζεται στο Σχήμα 1. Κάθε διεργασία πελάτης πρέπει να διατηρεί μια τοπική μεταβλητή που αποθηκεύει τη συνολική ποσότητα του προϊόντος που έχει αγοραστεί από αυτόν τον πελάτη.
- Ο συντονιστής μπορεί να ζητήσει από μια διεργασία πελάτη να εκτελέσει ένα αίτημα πριν τερματίσει η εκτέλεση του προηγούμενου αιτήματος. Επομένως, κάθε διεργασία πελάτη μπορεί να έχει παραπάνω από ένα αιτήματα ενεργά, τα οποία έχει αποστείλει προς εξυπηρέτηση στον εξυπηρετητή αρχηγό. Για το λόγο αυτό, κάθε διεργασία πελάτη θα πρέπει να διατηρεί έναν τοπικό μετρητή, **ο οποίος ονομάζεται active_requests,** με αρχική τιμή 0 τον οποίο θα αυξομειώνει ως εξής: κάθε φορά που στέλνει μήνυμα προς τον αρχηγό εξυπηρετητή, ο μετρητής θα αυξάνεται κατά 1, ενώ κάθε φορά που λαμβάνει μήνυμα από τον αρχηγό, ο μετρητής θα μειώνεται κατά 1. Αυτό χρησιμεύει ώστε να γνωρίζει η διεργασία πελάτη αν εκκρεμεί κάποια απάντηση από τον εξυπηρετητή αρχηγό.



Σχήμα 1: Παράδειγμα δικτύου

2.1 Εκκίνηση

Κατά την εκκίνηση του συστήματος, εκτελείται μια ρουτίνα, που ονομάζεται MPI_init(), η οποία αρχικοποιεί όλες τις διεργασίες που θα εκτελεστούν στο σύστημα (μία σε κάθε κόμβο του συστήματος).

Η διεργασία συντονιστής ξεκινάει να διαβάζει το testfile, το οποίο αρχικά περιέχει γεγονότα που καθορίζουν ποιες διεργασίες έχουν ρόλο εξυπηρετητή και ταυτόχρονα περιγράφουν τη συνδεσμολογία των διεργασιών προκειμένου να δημιουργηθεί ο δακτύλιος που τις συνδέει. Συγκεκριμένα, τα γεγονότα αυτά έχουν την εξής μορφή:

SERVER <server_rank> <left_neighbour_rank> <right_neighbour_rank>

Γεγονός καθορισμού διεργασίας εξυπηρετητή. Κατά το γεγονός αυτό, ο συντονιστής στέλνει ένα μήνυμα τύπου **<SERVER>** στη διεργασία με αναγνωριστικό **<server_rank>,** ώστε να την ενημερώσει ότι είναι μια διεργασία εξυπηρετητής, καθώς και για το ποιες είναι οι γειτονικές της διεργασίες στο λογικό δακτύλιο. Κάθε διεργασία που λαμβάνει ένα τέτοιο μήνυμα, αποθηκεύει την πληροφορία ότι η διεργασία με αναγνωριστικό <**right_neighbour_rank>** αποτελεί το δεξιό της γείτονα, ενώ η διεργασία με αναγνωριστικό <**left_neighbour_rank>** αποτελεί τον αριστερό της γείτονα στο λογικό δακτύλιο. Κάθε διεργασία που λαμβάνει ένα τέτοιο μήνυμα από το συντονιστή, απλά καταγράφει ότι είναι διεργασία εξυπηρετητής, καθώς και τα αναγνωριστικά του αριστερού και του δεξιού της γειτονικού κόμβου. Ο σχηματιζόμενος δακτύλιος είναι μονής κατεύθυνσης.

Για καθένα από τα παραπάνω μηνύματα τύπου **<SERVER>**, η διεργασία συντονιστής θα πρέπει να περιμένει ένα μήνυμα τύπου **<ACK>** από τη διεργασία στην οποία έστειλε το μήνυμα, πριν προχωρήσει στην επεξεργασία του επόμενου γεγονότος του testfile. Με αυτόν τον τρόπο εξασφαλίζεται ότι όλες οι διεργασίες εξυπηρετητές γνωρίζουν το ρόλο τους και τη θέση τους στο λογικό δακτύλιο κι έτσι μπορεί να ξεκινήσει ομαλά η προσομοίωση του συστήματος.

Μετά το πέρας των γεγονότων <**SERVER>**, το testfile περιέχει ένα γεγονός τύπου **<START_LEADER_ELECTION>**. Όταν ο συντονιστής διαβάσει ένα τέτοιο γεγονός αποστέλλει ένα μήνυμα τύπου **<START_LEADER_ELECTION>** σε κάθε διεργασία εξυπηρετητή που υπάρχει στο σύστημα και έπειτα περιμένει να λάβει ένα μήνυμα τύπου **<LEADER_ELECTION_DONE>** από τη διεργασία που θα εκλεγεί αρχηγός (την οποία δεν γνωρίζει) (βλέπε 2.2).

Όταν ληφθεί αυτό το μήνυμα, η διεργασία συντονιστής αποστέλλει ένα μήνυμα τύπου <**CLIENT>** που περιέχει το **αναγνωριστικό του αρχηγού** σε όλες τις διεργασίες πελάτες**,** ενημερώνοντάς τες για τον ρόλο τους ως πελάτες, καθώς και για το αναγνωριστικό του εκλεγμένου εξυπηρετητή αρχηγού. Τέλος, ο συντονιστής πρέπει να λάβει μηνύματα τύπου **<ACK>** από κάθε διεργασία πελάτη πριν συνεχίσει με την ανάγνωση του testfile.

Επίσης, αρχικά, κάθε αποθήκη (δηλαδή κάθε εξυπηρετητής) έχει αποθηκευμένες 300 μονάδες προϊόντος ενώ οι πελάτες δεν έχει υποβάλει καμία παραγγελία.

2.2 Εκλογή Αρχηγού (Leader Election)

Προκειμένου να καθοριστεί μια διεργασία εξυπηρετητή που θα χειρίζεται αιτήματα προμηθειών που παραλαμβάνονται από το συντονιστή και αφορούν όλες τις αποθήκες της εταιρείας, θα πρέπει να εκλεγεί ένας εξυπηρετητής αρχηγός.

Ο αλγόριθμος εκλογής αρχηγού που πρέπει να υλοποιήσετε είναι ο εξής:

Ο αλγόριθμος που λειτουργεί με γειτονιές και έχει χρονική πολυπλοκότητα O(nlogn) (όπως περιγράφεται στις διαφάνειες). Συγκεκριμένα, για κάθε ακέραιο k, η k-γειτονιά μιας διεργασίας p_i στο δακτύλιο είναι το σύνολο από εκείνες τις διεργασίες που βρίσκονται σε απόσταση το πολύ k από την p_i στο δακτύλιο (είτε προς τα αριστερά, είτε προς τα δεξιά).

Όταν μια διεργασία εξυπηρετητής, s, λάβει μήνυμα τύπου **<START_LEADER_ELECTION>** από το συντονιστή, ξεκινά τον αλγόριθμο εκλογής αρχηγού (αποστέλλοντας ένα μήνυμα τύπου **<PROBE>** προς κάθε έναν από τους γειτονικούς της κόμβους, δηλαδή στους κόμβους της 1-γειτονιάς του). Αν το μήνυμα **<START_LEADER_ELECTION>** ληφθεί αφού η διεργασία έχει ήδη αποστείλει μήνυμα τύπου **<PROBE>** στους γειτονικούς της κόμβους, τότε η s το αγνοεί.

Κάθε διεργασία που έχει λάβει μήνυμα τύπου **<START_LEADER_ELECTION>** από τη διεργασία συντονιστή θεωρείται ενεργή.

Πιο συγκεκριμένα, ο αλγόριθμος λειτουργεί σε φάσεις, όπου σε κάθε φάση k, $k \geq 0$, γίνονται τα εξής:

1. Κάθε ενεργή διεργασία προσπαθεί να νικήσει στη φάση k. Μια διεργασία είναι νικητής της φάσης αν έχει το μεγαλύτερο ID στην 2^k γειτονιά της.
2. Μόνο οι διεργασίες που νίκησαν στην k φάση μπορούν να συνεχίσουν στη k+1 φάση. Οι υπόλοιπες δεν διεκδικούν πλέον να γίνουν αρχηγοί και απλά προωθούν μηνύματα που λαμβάνουν από άλλες διεργασίες. Επομένως, μόνο οι ενεργές διεργασίες που έχουν νικήσει στην φάση k-1 συμμετέχουν στη φάση k. Έστω pi μια τέτοια διεργασία.
3. Η pi στέλνει μηνύματα τύπου <probe> με το ID της στην 2^k γειτονιά της (ένα προς τα αριστερά και ένα προς τα δεξιά).
4. Ένα μήνυμα τύπου <probe> δεν προωθείται από διεργασίες με μεγαλύτερα ID από αυτό που περιέχει το μήνυμα <probe> . (Αν αυτό συμβεί, το μήνυμα παύει να κυκλοφορεί στο δακτύλιο.)
5. Αν το μήνυμα φτάσει στην τελευταία διεργασία της 2^k γειτονιάς της p_i, τότε αυτή η τελευταία διεργασία στέλνει πίσω ένα μήνυμα <reply> στην διεργασία p_i.
6. Αν η διεργασία p_i λάβει μηνύματα τύπου reply κι από τις δύο κατευθύνσεις, τότε είναι ο νικητής της k φάσης και συνεχίζει στην k+1 φάση.
7. Μια διεργασία που λαμβάνει το δικό της μήνυμα τύπου <probe> είναι ο αρχηγός και τερματίζει τον αλγόριθμο στέλνοντας είναι μήνυμα τερματισμού γύρο από τον δακτύλιο.

Η διεργασία εξυπηρετητής που εκλέγεται αρχηγός, θα πρέπει να ενημερώσει όλες τις άλλες διεργασίες εξυπηρετητές ότι εκλέχτηκε αρχηγός και στη συνέχεια να αποστείλει στη διεργασία συντονιστή ένα μήνυμα τύπου **<LEADER_ELECTION_DONE>** για να την ενημερώσει για τον τερματισμό της φάσης υπολογισμού του αρχηγού.

Συνδεσμολογία πελατών που σχετίζονται με κάποιον εξυπηρετητή

Αφότου η συνδεσμολογία των εξυπηρετητών δημιουργηθεί και επιλεγεί εξυπηρετητής αρχηγός, ο συντονιστής προχωρά στην εκτέλεση γεγονότων για τη δημιουργία της συνδεσμολογίας των πελατών. Με κάθε εξυπηρετητή συσχετίζει έναν αριθμό από διεργασίες πελάτες.

[Undergraduates] Οι διεργασίες πελάτες που συσχετίζονται με κάθε εξυπηρετητή συνδέονται μεταξύ τους και με τον εξυπηρετητή με ένα δένδρο. Το δέντρο καθορίζεται από το testfile κατά την εκκίνηση του συστήματος.

Η διεργασία συντονιστής (rank=0) διαβάζοντας το testfile, ανακαλύπτει ποιους πελάτες θα συσχετίσει με κάθε εξυπηρετητή (μέσω ενός γεγονότος τύπου CONNECT). Σε κάθε πελάτη αντιστοιχεί ένας γονικός κόμβος ώστε να σχηματιστεί ένα δένδρο με ρίζα τον εξυπηρετητή. Το γεγονός τύπου CONNECT περιγράφεται πιο αναλυτικά στη συνέχεια:

- CONNECT <client_rank> <parent_rank>: Κατά το γεγονός αυτό, ο συντονιστής αρχικά στέλνει ένα μήνυμα τύπου CONNECT στη διεργασία με αναγνωριστικό <client_rank> ώστε να την ενημερώσει ότι ο γονικός της κόμβος στο (προς δημιουργία) δένδρο είναι η διεργασία με αναγνωριστικό <parent_rank>. Η διεργασία με αναγνωριστικό <client_rank> στέλνει στη διεργασία με αναγνωριστικό <parent_rank> ένα μήνυμα ώστε να την ενημερώσει ότι είναι θυγατρικός της κόμβος. Η διεργασία με αναγνωριστικό <parent_rank> την καταγράφει ως έναν από τους θυγατρικούς της κόμβους και της στέλνει πίσω ένα μήνυμα επιβεβαίωσης ACK. Όταν η θυγατρική διεργασία λάβει το ACK

στέλνει με τη σειρά της ένα μήνυμα ACK στον συντονιστή για να τον ενημερώσει για την περάτωση της εκτέλεσης του γεγονότος αυτού.

[Graduates] Οι διεργασίες πελάτες που συσχετίζονται με κάθε εξυπηρετητή συνδέονται μεταξύ τους και με τον εξυπηρετητή με ένα γράφο. Ο γράφος καθορίζεται από το testfile κατά την εκκίνηση του συστήματος.

Η διεργασία συντονιστής (rank=0) διαβάζοντας το testfile, ανακαλύπτει ποιους πελάτες θα συσχετίσει με κάθε εξυπηρετητή (μέσω ενός γεγονότος τύπου CONNECT). Σε κάθε πελάτη αντιστοιχεί ένας γονικός κόμβος ώστε να σχηματιστεί ένα δένδρο με ρίζα τον εξυπηρετητή. Το γεγονός τύπου CONNECT περιγράφεται πιο αναλυτικά στη συνέχεια:

CONNECT <client1 > <client2>: Κατά το γεγονός αυτό, ο συντονιστής αρχικά στέλνει ένα μήνυμα τύπου CONNECT στη διεργασία με αναγνωριστικό <client1 > ώστε να την ενημερώσει ότι ένας από τους γειτονικούς της κόμβους στο γράφο είναι η διεργασία με αναγνωριστικό <client2>. Η διεργασία με αναγνωριστικό <client1 > στέλνει στη διεργασία με αναγνωριστικό < client2> ένα μήνυμα ώστε να την ενημερώσει ότι είναι γειτονικοί κόμβοι. Η διεργασία με αναγνωριστικό < client2> την καταγράφει ως έναν από τους γειτονικούς της κόμβους και της στέλνει πίσω ένα μήνυμα επιβεβαίωσης ACK. Όταν η πρώτη διεργασία λάβει το ACK στέλνει με τη σειρά της ένα μήνυμα ACK στον συντονιστή για να τον ενημερώσει για την περάτωση της εκτέλεσης του γεγονότος αυτού.

Όταν ο συντονιστής τελειώσει την επεξεργασία όλων των CONNECT γεγονότων στο testfile, στέλνει ένα μήνυμα τύπου SPANNING-TREE στον εξυπηρετητή αρχηγό. Ο αρχηγός στέλνει αυτό το μήνυμα σε όλους τους εξυπηρετητές και κάθε εξυπηρετητής που το παραλαμβάνει ξεκινά την εκτέλεση ενός FS-Tree αλγορίθμου για τη δημιουργία ενός σκελετικού δένδρου του γράφου πελατών του. Την ίδια διαδικασία ακολουθεί και ο αρχηγός. Σκοπός αυτού του γεγονότος είναι ο κάθε εξυπηρετητής να δημιουργήσει ένα δένδρο από τους πελάτες του για να μπορεί να εκτελεί αλγορίθμους broadcast και convergecast χρησιμοποιώντας το. Ο εξυπηρετητής θα πρέπει να αποτελεί τη ρίζα του δένδρου αυτού.

Όταν η δημιουργία του δένδρου πελατών ενός εξυπηρετητή S ολοκληρωθεί, ο S εκτελεί αλγόριθμο για να υπολογίσει πόσοι κόμβοι υπάρχουν στο δένδρο πελατών του. Στη συνέχεια, στέλνει το νούμερο αυτό στον εξυπηρετητή αρχηγό και τυπώνει το μήνυμα:

SPANNING-TREE <server id> <nodes>,

όπου <server id> είναι το αναγνωριστικό του S και <nodes> είναι ο αριθμός των πελατών στο σκελετικό δένδρο πελατών του.

Αφού λάβει απαντήσεις από όλους τους εξυπηρετητές, ο εξυπηρετητής αρχηγός αθροίζει όλους τους αριθμούς που έλαβε, στέλνει <ACK> στο συντονιστή και τυπώνει

REPORT <server id> <nodes>,

όπου <server id> είναι το αναγνωριστικό του εξυπηρετητή αρχηγού και <nodes> είναι ο συνολικός αριθμός πελατών που έχουν προσμετρηθεί στο σύστημα.

2.2 Λειτουργία του Συστήματος και Γεγονότα

Μετά την ομαλή εκκίνηση του συστήματος και τη δημιουργία της τοπολογίας του, η διεργασία συντονιστής συνεχίζει την ανάγνωση του testfile κι αποστέλλει τα αντίστοιχα μηνύματα στους κατάλληλους κόμβους. Τα γεγονότα που θα διαβάζει, καθώς και οι ενέργειες που θα πρέπει να γίνονται, περιγράφονται αναλυτικά στη συνέχεια.

- ORDER <client_rank> <NUM> : Ο συντονιστής στέλνει περιγραφή αιτήματος παραγγελίας στον πελάτη με αναγνωριστικό <client_rank> ποσότητας <NUM>. Αυτός

την προωθεί στον εξυπηρετητή μέσω του σκελετικού δένδρου στο οποίο ανήκει (ακολουθώντας το μονοπάτι από τον εαυτό του προς την ρίζα όπου βρίσκεται ο εξυπηρετητής που συσχετίζεται με τον εν λόγω πελάτη). Όταν ο εξυπηρετητής παραλάβει την παραγγελία, μειώνει την ποσότητα του προϊόντος που αποθηκεύεται στην αποθήκη κατά <NUM> και στέλνει πίσω ένα ACK στον πελάτη. Ο πελάτης αυξάνει κατά <NUM> την ποσότητα του προϊόντος που έχει αγοράσει και στέλνει ένα <ACK> στο συντονιστή. Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος, η διεργασία πελάτη θα πρέπει να στείλει ένα μήνυμα τύπου <ACK> στον συντονιστή και να τυπώσει:

CLIENT <client_rank> SOLD <NUM>

- SUPPLY <server_rank> <NUM>: Ο συντονιστής στέλνει στον εξυπηρετητή S με αναγνωριστικό <server_rank> ένα μήνυμα τύπου SUPPLY. Αυτός με τη σειρά του ζητάει από τους υπόλοιπους εξυπηρετητές να του στείλουν συνολική ποσότητα προϊόντος <NUM>. Η επικοινωνία γίνεται αριστερόστροφα και πραγματοποιείται ως εξής:
    - Ο εξυπηρετητής με αναγνωριστικό <server_rank> προωθεί στον αριστερό γείτονά του S' την αίτηση.
    - Αν η ποσότητα Q' του προϊόντος που αποθηκεύεται στον S' είναι μεγαλύτερη από 150+<NUM> (δηλαδή Q' > 150 + <NUM>), τότε ο S' στέλνει απευθείας στον εξυπηρετητή S ένα μήνυμα με την ποσότητα <NUM> και θέτει την μεταβλητή Q'=Q'-<NUM>. Ο εξυπηρετητής S ενημερώνει τη δική του μεταβλητή Q ως εξής: Q=Q+<NUM> και στέλνει ένα μήνυμα τύπου SUPPLY_ACK στο συντονιστή.
    - Αν 150 < Q' < 150 +<NUM>, τότε ο S' στέλνει απευθείας στον εξυπηρετητή S ένα μήνυμα με την ποσότητα q = Q'-150 και θέτει την μεταβλητή Q'=150. Επίσης προωθεί προς τον αριστερό του γείτονα την αίτηση του S ανανεώνοντας το πεδίο <NUM> σε <NUM>-q δηλαδή ζητάει να στείλει αυτός την υπολειπόμενη ποσότητα. Ο εξυπηρετητής πηγή εφόσον λάβει ένα μήνυμα SUPPLY από δεξιά με <server_rank> ίσο με δικό του (δηλαδή αν το αίτημα του S κάνει το γύρο του δακτυλίου χωρίς να έχει βρεθεί η απαιτούμενη ποσότητα), ο S συνεισφέρει το δικό του ποσοστό (έτσι ώστε να του απομένουν προμήθειες 150 όπως και στους υπόλοιπους) και στη συνέχεια στέλνει ένα μήνυμα τύπου SUPPLY_ACK στον συντονιστή (ανεξαρτήτου αν έχουν συμπληρωθεί όλες οι <NUM> μονάδες που ζητήθηκαν ή όχι).
    - Αν ο S' έχει ποσότητα μικρότερη από 150 (δηλαδή Q' < 150), τότε ο S' προωθεί την αίτηση στον αριστερό του γείτονα ο οποίος πραγματοποιεί αντίστοιχες ενέργειες.

  Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος, η διεργασία εξυπηρετητή S θα πρέπει θα πρέπει να στείλει ένα μήνυμα τύπου ACK στο συντονιστή να τυπώσει:

  SERVER <server_rank> RECEIVED <received_amount>

- EXTERNAL_SUPPLY <NUM> : Ο συντονιστής στέλνει στον εξυπηρετητή αρχηγό ένα μήνυμα τύπου EXTERNAL _SUPPLY. Ο αρχηγός πρέπει να βρει ποιοί εξυπηρετητές έχουν προμήθεια μικρότερη από 150 (συμπεριλαμβανομένου του ίδιου). Για τον εαυτό του ελέγχει απευθείας την μεταβλητή Q και αν είναι μικρότερη του 150 προσθέτει όσο χρειάζεται από την προμήθεια ώστε το Q να γίνει ίσο με 150 . Για τους υπόλοιπους στέλνει ένα μήνυμα πάλι αριστερόστροφα έως ότου φτάσει πίσω στον ίδιο. Όσοι εξυπηρετητές λάβουν αυτό το μήνυμα και έχουν Q μικρότερο του 150 στέλνουν απευθείας στον αρχηγό ένα μήνυμα SUPPLY_REQUEST ζητώντας την ποσότητα 150-Q . Ο αρχηγός όταν λάβει ένα μήνυμα SUPPLY_REQUEST στέλνει απευθείας στον αποστολέα το ποσό που του ζήτησε και περιμένει πίσω ένα μήνυμα επιβεβαίωσης. Αν η ποσότητα

δεν φτάνει του στέλνει ότι περίσσεψε από την προμήθεια. Αν περισσεύει προμήθεια, ο αρχηγός την προσθέτει στη δική του αποθήκη. Όταν ο αρχηγός έχει λάβει επιβεβαίωση από όλους και έχει μοιράσει όλη την προμήθεια στέλνει ένα μήνυμα επιβεβαίωσης στον συντονιστή.

Πριν ξεκινήσει την εκτέλεση ενός γεγονότος EXTERNAL_ SUPPLY, ο συντονιστής θα πρέπει να έχει λάβει επιβεβαίωση τερματισμού από όλα τα γεγονότα που έχει εκκινήσει στο παρελθόν. Αυτό θα υλοποιηθεί με την βοήθεια ενός counter που θα αυξάνεται κατά 1 όταν ο συντονιστής θα ξεκινάει ένα γεγονός και θα μειώνεται κατά 1 όταν θα λάβει ένα μήνυμα επιβεβαίωσης. Μόνο όταν ο μετρητής αυτός έχει την τιμή 0, ο συντονιστής εκκινεί την εκτέλεση του γεγονότος EXTERNAL_SUPPLY.

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος, η διεργασία εξυπηρετητή θα πρέπει να τυπώσει:

LEADER SUPPLY <received_amount> OK

- PRINT : Ο συντονιστής στέλνει στον εξυπηρετητή αρχηγό ένα μήνυμα τύπου PRINT εφόσον έχει λάβει ACK από τα προηγούμενα γεγονότα. Ο αρχηγός προωθεί αριστερά το μήνυμα. Όταν μια διεργασία λάβει το μήνυμα, πρέπει να τυπώσει:

  SERVER <server_rank> HAS QUANTITY <Q>

  και να προωθήσει το μήνυμα στα αριστερά της.

  Όταν ο εξυπηρετητής αρχηγός παραλάβει από δεξιά ένα μήνυμα PRINT στέλνει ένα ACK στον συντονιστή ώστε να συνεχίσει με το επόμενο γεγονός.

- REPORT: Ο συντονιστής στέλνει στον εξυπηρετητή αρχηγό ένα μήνυμα τύπου REPORT. Ο αρχηγός στέλνει αυτό το μήνυμα σε όλους τους εξυπηρετητές και κάθε εξυπηρετητής που το παραλαμβάνει το προωθεί στα φύλλα του σκελετικού δέντρου των πελατών που συσχετίζονται μ' αυτόν τον εξυπηρετητή. Την ίδια διαδικασία ακολουθεί και ο αρχηγός. Σκοπός αυτού του γεγονότος είναι ο κάθε εξυπηρετητής να αποκτήσει πληροφορία από τους πελάτες για τη συνολική ποσότητα προϊόντος που έχει καταναλωθεί από αυτούς. Για να το πετύχει αυτό, χρειάζεται να εκτελεστεί ένα broadcast και ένα convergecast στο δένδρο του οποίου ο εξυπηρετητής αποτελεί ρίζα. Κάθε εξυπηρετητής, αφού υπολογίσει τη ζητούμενη ποσότητα την προωθεί στον εξυπηρετητή αρχηγό και τυπώνει το μήνυμα:

  REPORT <server id> <quantity>,

  όπου <server id> είναι το αναγνωριστικό του S και <quantity> είναι η ποσότητα του προϊόντος που υπολογίστηκε.

  Αφού λάβει απαντήσεις από όλους τους εξυπηρετητές, ο εξυπηρετητής αρχηγός αθροίζει όλες τις ποσότητες που έλαβε, στέλνει <ACK> στο συντονιστή και τυπώνει

  REPORT <server id> <quantity>,

  όπου <server id> είναι το αναγνωριστικό του εξυπηρετητή αρχηγού και <quantity> είναι η συνολική ποσότητα προϊόντος που έχει καταναλωθεί από όλους τους πελάτες στο δίκτυο.

2.3 Τερματισμός Λειτουργίας

Όταν η διεργασία συντονιστή τελειώσει την ανάγνωση του testfile, θα πρέπει να περιμένει όλες τις υπόλοιπες διεργασίες να ολοκληρώσουν τις ανταλλαγές μηνυμάτων που εκκρεμούν. Αυτό θα γίνει με τη βοήθεια του counter που αναφέρθηκε παραπάνω. Έτσι, έχοντας τελειώσει την ανάγνωση του testfile και γνωρίζοντας πότε όλα τα ενεργά γεγονότα έχουν ολοκληρωθεί, μπορεί να τερματίσει τη λειτουργία του συστήματος με ασφάλεια.

Συγκεκριμένα, πριν τερματίσει ο ίδιος, θα αποστέλλει ένα μήνυμα τύπου <TERMINATE> σε όλες τις υπόλοιπες διεργασίες έτσι ώστε να τις ενημερώσει για τον τερματισμό της λειτουργίας του συστήματος.

3. Message Passing Interface

Για την υλοποίηση της εργασίας θα πρέπει να χρησιμοποιήσετε τη βιβλιοθήκη Message Passing Interface (MPI). Η βιβλιοθήκη αυτή παρέχει τη δυνατότητα ανταλλαγής μηνυμάτων μεταξύ διεργασιών (που μπορεί να εκτελούνται στο ίδιο ή σε διαφορετικά μηχανήματα) μέσω ενός καθιερωμένου Application Programming Interface (API), ανεξαρτήτως γλώσσας και υλοποίησης. Το API αυτό παρέχει τη δυνατότητα ανταλλαγής μηνυμάτων μεταξύ των διεργασιών όχι μόνο με τη χρήση μηνυμάτων σημείο-προς-σημείο (point-to-point messaging), αλλά και με συλλογικό τρόπο, π.χ. υποστηρίζει επικοινωνία μεταξύ ομάδων διεργασιών (multicast), καθώς και καθολική επικοινωνία (broadcasts). Για την προσομοίωση που ζητείται, κάθε κόμβος του συστήματος θα προσομοιώνεται από μία διεργασία MPI. Είναι αξιοσημείωτο ότι περισσότερες από μία διεργασίες MPI μπορεί να τρέχουν στο ίδιο μηχάνημα (παρότι προσομοιώνουν διαφορετικούς κόμβους του συστήματος). Το σύστημα MPI είναι εγκατεστημένο στα μηχανήματα του Τμήματος. Οι δύο βασικές εντολές που απαιτούνται για τη λειτουργία του είναι οι mpicc και mpirun. Η εντολή mpicc χρησιμοποιείται για την μεταγλώττιση προγραμμάτων που χρησιμοποιούν το API του MPI. Με την εντολή mpirun μπορείτε να τρέξετε το εκτελέσιμο ταυτόχρονα σε περισσότερα του ενός μηχανήματα. Ένα παράδειγμα χρήσης παρατίθεται στη συνέχεια:

$ mpirun -np <count> --hostfile <file containing hostnames> <executable> <NUM_SERVERS>

όπου η επιλογή **<np>** καθορίζει το συνολικό αριθμό των MPI διεργασιών που θα εκτελούν το εκτελέσιμο αρχείο που δίνεται σαν τελευταίο όρισμα στην παραπάνω γραμμή εντολών.

Στην επιλογή **<hostfile>** δίνεται το όνομα του αρχείου στο οποίο περιέχονται τα hostnames των μηχανημάτων όπου θα εκτελεστούν διεργασίες MPI. Κάθε διεργασία που εκκινείται με τη χρήση της εντολής mpirun έχει ένα ξεχωριστό αναγνωριστικό στο σύστημα MPI, το οποίο ονομάζεται βαθμός (rank) MPI. Επίσης, μπορεί να μάθει πόσες άλλες διεργασίες MPI τρέχουν στο σύστημα και να τους στείλει μηνύματα, χρησιμοποιώντας το rank τους σαν αναγνωριστικό. Είναι αξιοσημείωτο πως μία διεργασία μπορεί να στείλει μηνύματα μόνο σε διεργασίες που έχουν ξεκινήσει από την ίδια εντολή mpirun και είναι κατά συνέπεια μέλη του ίδιου "κόσμου" MPI. Για παραπάνω πληροφορίες μπορείτε να διαβάσετε το υλικό που βρίσκεται στους παρακάτω συνδέσμους:

https://computing.llnl.gov/tutorials/mpi/

http://mpitutorial.com/tutorials/

4. Παράδοση Εργασίας

Για την παράδοση της εργασίας θα πρέπει να χρησιμοποιήσετε το πρόγραμμα turnin, που υπάρχει εγκατεστημένο στα μηχανήματα του τμήματος. Συγκεκριμένα, η εντολή παράδοσης είναι:

turnin project2@hy486

Για να επιβεβαιώσετε ότι η υποβολή της εργασίας ήταν επιτυχής, μπορείτε να χρησιμοποιήσετε την εντολή:

verify-turnin project2@hy486

Προσοχή, τα παραδοτέα σας θα πρέπει να περιέχουν ό,τι χρειάζεται και να είναι σωστά δομημένα ώστε να κάνουν compile και να εκτελούνται στα μηχανήματα της σχολής, όπου και θα γίνει η εξέταση της εργασίας. Η προθεσμία παράδοσης είναι την: 22 Δεκεμβρίου 2020.