Εθνικό & Καποδιστριακό Πανεπιστήμιο Αθηνών

Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Κ24: Προγραμματισμός Συστήματος
2010 – 2011

Διδάσκουσα: Μέμα Ρουσσοπούλου

Εργασία 2

Ημερομηνία παράδοσης: 3/5/2011

Σημείωση: Παρακαλείστε να παρουσιάσετε ένα σύντομο αρχείο README που να εξηγεί τη λογική του προγράμματος σας. Οδηγίες για την υποβολή της άσκησης υπάρχουν στη ιστοσελίδα του μαθήματος.

Υπεύθυνοι για την άσκηση αυτή (ερωτήσεις, βαθμολόγηση, κλπ) είναι: Athanasios Doukoudakis (grad1061) και Orestis Polychroniou (grad1142). Θερμές ευχαριστίες στον Ορέστη για την εκφώνηση και υλοποίηση της εργασίας και στον Θάνο για τις διευκρινίσεις στην εκφώνηση.

Ένα πολύ βασικό εργαλείο που χρησιμοποιείται τα τελευταία χρόνια για την διαχείριση μεγάλου όγκου δεδομένων είναι το μοντέλο MapReduce. Το μοντέλο αυτό εμφανίστηκε με τη σημερινή του μορφή το 2004, ανήκει στον τομέα των κατανεμημένων συστημάτων (distributed systems) και χρησιμοποιείται κατά κόρον από μεγάλες εταιρίες όπως η Google, η Yahoo κλπ.

Το μοντέλο αυτό είναι πρακτικά ένας αλγόριθμος που εκτελείται σε ένα δίκτυο υπολογιστών (cluster). Ο κύριος στόχος του μοντέλου είναι να επιτρέπει να επιτυγχάνουμε επεξεργασία τεράστιων όγκων δεδομένων χρησιμοποιώντας ένα μεγάλο αριθμό συμβατικών υπολογιστών χαμηλής αξίας. Το MapReduce εκτελείται σε 2 κύριες φάσεις όπως δείχνει και το όνομα του, το map και το reduce. Μια πιθανή μορφή είναι να χρησιμοποιούμε κάποιους υπολογιστές (έστω M) ώστε να εκτελούν την λειτουργία map και κάποιους άλλους υπολογιστές (έστω R) ώστε να εκτελούν τη λειτουργία reduce.

Ο αλγόριθμος πρακτικά στέλνει δεδομένα από τους mappers προς τους reducers. Κάθε mapper στέλνει πιθανώς σε όλους τους reducers και κάθε reducer λαμβάνει πιθανώς από όλους τους mappers. Πρακτικά το MapReduce είναι ένα μοντέλο υλοποίησης κατανεμημένων αλγορίθμων και μπορούμε να φτιάξουμε βιβλιοθήκες που να κάνουν μόνες τους τα πάντα αρκεί να δώσουμε μερικά σημεία κλειδιά:

- Αρχικά έχουμε ως είσοδο ένα σύνολο αρχείων και τα μετασχηματίζουμε σε ένα σύνολο εγγραφών. Για παράδειγμα μπορεί να διαβάζουμε μια ιστοσελίδα σε μορφή HTML και να δημιουργούμε μια εγγραφή για κάθε χρήσιμη λέξη που περιέχει.
- Αυτές οι εγγραφές έπειτα μέσω του “mapping” μετασχηματίζονται σε ζευγάρια κλειδί–δεδομένο. Ο ρόλος του κλειδιού εδώ είναι πολύ συγκεκριμένος και είναι η καρδιά του τρόπου λειτουργίας του μοντέλου. Πρακτικά όταν κάποια δεδομένα έχουν ίδιο κλειδί, θα καταλήξουν στον ίδιο reducer για να τα επεξεργαστεί και πιθανώς να τα συνδυάσει όπως θα δούμε.
- Για να αποφασιστεί κάθε κλειδί σε ποιον reducer θα σταλεί ένα ζευγάρι κλειδί – δεδομένο, χρησιμοποιείται η partition. Αυτή δέχεται ένα κλειδί και δίνει με βάση τον διαθέσιμο αριθμό των reducers και το κλειδί, τον προορισμό του ζευγαριού. Όπως είναι λογικό, ζευγάρια με ίδια κλειδιά θα

πάνε προς τον ίδιο προορισμό.

- Ο reducer που λαμβάνει ζευγάρια από πιθανώς όλους τους mappers, ταξινομεί τα ζευγάρια χρησιμοποιώντας τα κλειδιά τους και μια δοσμένη συνάρτηση compare.
- Στην επόμενη φάση εκτελείται η συνάρτηση reduce ανά διαφορετικό κλειδί για όλα τα ζευγάρια που έχουν το ίδιο κλειδί. Ως έξοδος παράγεται μια εγγραφή και πάλι. Για παράδειγμα αν είχαμε σαν κλειδί μια λέξη και σαν δεδομένα το πόσες φορές υπάρχει σε κάθε αρχείο, για να τα συνδυάσουμε θα επιστρέφαμε τη λέξη και το άθροισμα των εμφανίσεών της.
- Το τελευταίο σημείο κλειδί είναι το πώς θα γραφτούν οι εγγραφές που παρήχθησαν από το reduce σε αρχεία εξόδου.

Οι βασικές συναρτήσεις που περιγράψαμε παραπάνω ορίζονται αφαιρετικά ως εξής:

- input(file) → επιστρέφει ένα record από το αρχείο
- map(record) → παράγει ένα ζεύγος <key, value> από ένα record
- partition(key, total_reducers) → με βάση ένα κλειδί και τον αριθμό reducers, αποφασίζει τον #reducer που ανήκει το κλειδί
- compare(key_1, key_2) συνάρτηση σύγκρισης κλειδιών για ταξινόμηση με βάση τα κλειδιά
- reduce(key, values) → συνοψίζει ένα σύνολο δεδομένων με ίδιο κλειδί σε μια τελική εγγραφή
- output(records) → γράψιμο εξόδου με βάση το σύνολο εγγραφών που παρήγαγε ο reducer στο αρχείο file

Στα πλαίσια αυτής της εργασίας εμείς θέλουμε να υλοποιήσετε μια απλοποιημένη μορφή του MapReduce που μόνο θα προσομοιώνει το πραγματικό μοντέλο τοπικά σε έναν υπολογιστή. Πιο συγκεκριμένα θα πρέπει να φτιάξετε ένα πρόγραμμα που θα δημιουργεί, με την χρήση της fork, M παιδιά για να παίξουν τον ρόλο των mappers, και R παιδιά για να παίξουν τον ρόλο των reducers. Οι αριθμοί M και R θα δίνονται από την γραμμή εντολών ως ορίσματα στο πρόγραμμα.

Η επικοινωνία μεταξύ ενός worker και ενός reducer θα γίνεται με την χρήση σωλήνων (pipes). Όπως είπαμε και παραπάνω, κάθε mapper είναι ικανός να μιλήσει με οποιονδήποτε reducer, οπότε το σύνολο των pipes που θα χρειαστεί να κατασκευάσετε είναι MxR.

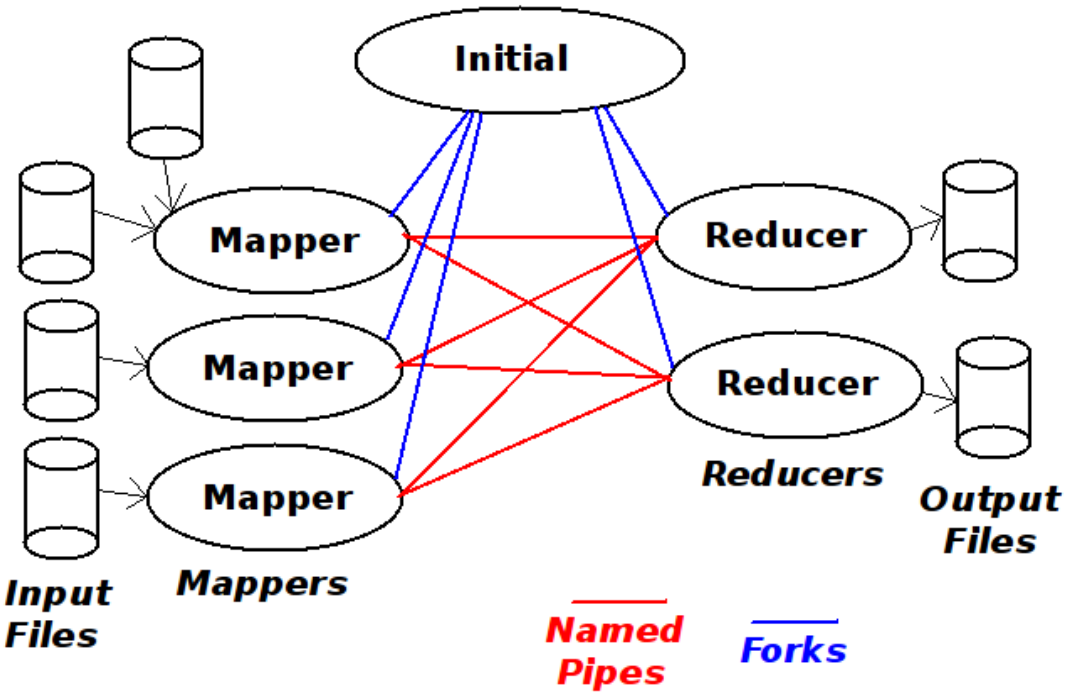
Σε κάθε mapper (map child process) αντιστοιχούν συγκεκριμένα αρχεία εισόδου. Τα αρχεία εισόδου και εξόδου έχουν συγκεκριμένα ονόματα και δεν ψάχνουμε αδιακρίτως στο φάκελο input_folder για δεδομένα. Πιο συγκεκριμένα στον φάκελο αυτόν μας ενδιαφέρουν τα αρχεία με ονόματα file_x όπου x είναι ένας αύξων ακέραιος. Αυτό μπορείτε να το θεωρήσετε δεδομένο και να το χρησιμοποιήσετε κατευθείαν στον κώδικα σας (με έλεγχο αν τυχόν δεν υπάρχουν τα αρχεία) ή να χρησιμοποιήσετε τις κλήσεις συστήματος για καταλόγους, και αφού βρείτε τα περιεχόμενα του φακέλου, να διαβάσετε τα αρχεία που μας ενδιαφέρουν. Ομοίως για τα αρχεία εξόδου, ο 1ος reducer θα γράφει στο file_1, ο 2ος reducer θα γράφει στο file_2 κλπ.

Κάθε αρχείο που διαβάζεται από έναν mapper, μετατρέπεται σε ένα σύνολο από εγγραφές, με την χρήση της συνάρτησης input. Έπειτα καλεί τη συνάρτηση map και μετατρέπει κάθε εγγραφή σε ζευγάρι <key, value>. Στην συνέχεια καλεί τη συνάρτηση partition πάνω στο κλειδί από το ζευγάρι. Η partition επιστρέφει τον αριθμό του reducer που αναλογεί στο συγκεκριμένο κλειδί. Έπειτα στέλνει το κάθε ζευγάρι στον κατάλληλο reducer μέσω των pipes.

Το reduce child process αφορά έναν συγκεκριμένο reducer. Μαζεύει από τους mappers ζευγάρια <key, value>, τα ταξινομεί με βάση τη συνάρτηση compare και εκτελεί στα δεδομένα με ίδιο κλειδί τη

συνάρτηση reduce. Τελικά γράφει την εγγραφή που δίνει ως έξοδο το reduce στο αρχείο εξόδου με βάση τις συναρτήσεις output. Κάθε reducer γράφει σε ένα αρχείο εξόδου.

Μπορείτε να δείτε τις συναρτήσεις αυτές στο mapreduce.h και μπορείτε να δείτε και μια υλοποίηση τους που υλοποιεί ταξινόμηση των λέξεων που περιέχουν τα input files και μια υλοποίηση που υλοποιεί την κατασκευή inverted index για keyword search των input files. Παρόμοια indexes χρησιμοποιούν όλες οι search engines όπως η Google. Η υλοποίηση αυτή είναι για να σας διευκολύνει στην υλοποίηση σας, ωστόσο η χρήση της δεν είναι δεσμευτική. Αν κάποιος μπερδεύεται μπορεί να υλοποιήσει δικές του συναρτήσεις που να υλοποιούν τις λειτουργείς που περιγράψαμε παραπάνω.



Οι 6 βασικές συναρτήσεις που θα υπάρχουν στη βιβλιοθήκη είναι οι:

1. MRIFILE * **input**_open(const char * filename);
   char * **input**_next(MRIFILE * file); /* Returns NULL when out of records */
   void **input**_close(MRIFILE * file);
2. pair **map**(const char * record);
3. int **partition**(const char * key, int reducers);
4. int **compare**(const char * key1, const char * key2); /* In most cases it is a plain strcmp */
5. char * **reduce**(const char * key, char ** values, int values_count);
6. MROFILE * **output**_open(const char * filename);

void **output**_next(MROFILE * file, const char *record);

void **output**_close(MROFILE * file);

Η διαχείριση σφαλμάτων που θα πρέπει να κάνετε είναι όταν συμβεί κάποιο σφάλμα, να ενημερώνονται όλες οι διεργασίες και να τερματίζουν, να διαγράφονται τα named pipes που χρησιμοποιήθηκαν για επικοινωνία κλπ. Το ίδιο συμβαίνει και αν ο χρήστης διακόψει το πρόγραμμα με Control+C για παράδειγμα. Αν συμβεί σφάλμα θα πρέπει τα παιδία να ενημερώνουν τον πατέρα και αυτός να λαμβάνει να τερματίσει ότι είναι ανοιχτό, ενώ στην συνέχεια να ενημερώσει τον χρήστη με τον κωδικό λάθους του συγκεκριμένου παιδιού.

Υποδείξεις - Διευκρινήσεις

Για την διευκόλυνση σας μπορείτε να χρησιμοποιήσετε named pipes για την επικοινωνία, τα οποία θα κατασκευάζονται από τη διεργασία-πατέρα πριν γίνουν τα forks. Τα pipes αυτά βολεύει να έχουν τη μορφή x.y όπου x ένα mapper ID και y ένα reducer ID, ώστε να είναι το pipe που θα χρησιμοποιήσει ο mapper x για να στείλει δεδομένα στον reducer y και αντίστοιχα το pipe που θα χρησιμοποιήσει ο reducer y για να λάβει δεδομένα από τον mapper x. Στα πλαίσια της άσκησης δεν είστε υποχρεωμένοι να υιοθετήσετε αυτή την πρακτική αλλά είναι αρκετά βολική. Για παράδειγμα μια καλή ονομασία θα ήταν .pipe.x.y, ώστε να είναι και κρυφά.

Ένα δεύτερο ζήτημα που η άσκηση δε σας περιορίζει είναι το πως θα στέλνετε την πληροφορία για το ζευγάρι <key, value> από mapper σε reducer. Ένα στοιχείο που σας βοηθάει αρκετά είναι το ότι οι συμβολοσειρές έχουν ειδικό χαρακτήρα τερματισμού, οπότε μπορείτε να υιοθετήσετε αυτήν την αναπαράσταση για την αποστολή δεδομένων. Αλλά και οι δύο αυτές προσεγγίσεις είναι αρκετά απλοϊκές και έχει ενδιαφέρον να δείτε αν μπορείτε να κάνετε κάποιου είδους buffering για να βελτιώσετε την απόδοση. Ένα ακόμη κομμάτι που θα πρέπει να προσέξετε εδώ, και καλείστε να αντιμετωπίσετε είναι στην επικοινωνία μέσω των pipes, όταν ένας mapper δεν έχει γράψει ολόκληρο το μήνυμα του στον σωλήνα και ο αντίστοιχος reducer διαβάζει μισό μήνυμα. Μέσα στις διαφάνειες υπάρχει η λύση του συγκεκριμένου προβλήματος.

Στην υλοποίηση σας πρέπει να χρησιμοποιήσετε system calls. Κατά συνέπεια αποκλείονται οι χρήσεις των fopen, fread, fwrite. Για να γράψετε σε αρχεία και pipes πρέπει να χρησιμοποιήσετε τα αντίστοιχα system calls όπως έχουμε δει στις διαφάνειες του μαθήματος.

Επίσης τα αρχεία εισόδου θα πρέπει να διαμοιράζονται μεταξύ των mappers, έτσι ώστε να μην επεξεργαστεί ο ένας mapper 5 αρχεία ενώ ο άλλος 1 αρχείο. Ένας απλός τρόπος είναι να υπολογίζεται την ποσότητα K mod N, όπου K ο αύξων αριθμός του αρχείου και N το σύνολο των mappers, και να αντιστοιχείτε έναν αριθμό σε έναν reducer.

Για να επιλέξει ο reducer από ποιον mapper θα διαβάσει υπάρχουν διάφορες μέθοδοι. Μια αποδοτική μέθοδος είναι το να διαβάζουν κυκλικά ή τυχαία. Αλλά θα πρέπει να σκεφτείτε μήπως αυτή η μέθοδος μπορεί να οδηγήσει σε αδιέξοδα. Μια μέθοδος που επιτρέπει να γνωρίζουμε αν μια είσοδος έχει δεδομένα για διάβασμα είναι η συνάρτηση poll, που μπορεί να χρησιμοποιηθεί και για να διαλέξουμε από ένα σύνολο εισόδων, ποιες θα διαβάσουμε σε αυτή τη φάση. Αφήνεται ανοιχτή η μέθοδος που θα χρησιμοποιήσετε, αλλά θα πρέπει να προσέξετε να μην εμφανιστούν deadlocks στο τελικό σας πρόγραμμα. Ένας καλός τρόπος να ελέγξετε για deadlocks είναι να χρησιμοποιήσετε σε κάποια tests μικρό αριθμό mappers και μεγάλο αριθμό reducers και αντίστροφα. Προσέξτε πολύ αυτό το σφάλμα.

Ο χρήστης αν θέλει να ορίσει τη λειτουργικότητα θα πρέπει να ορίσει μόνος του τις δομές MRIFILE

και MROFILE. Αυτές οι συναρτήσεις με τα ίδια πρωτότυπα υπάρχουν και στο mapreduce.h που σας δίνεται έτοιμο. Ανάλογα με το αντίστοιχο .c που υλοποιεί αυτές τις συναρτήσεις, εκτελούμε και διαφορετική ενέργεια, αλγόριθμο κλπ αν θέλουμε. Ο τύπος pair περιέχει απλά 2 C-strings με ονόματα πεδίων key, value και είναι ορισμένο στο mapreduce.h αρχείο.

Όπως προαναφέραμε στα πλαίσια της άσκησης σας δίνονται έτοιμα:

1. το mapreduce.h
2. το mapreduce_sort.o object file για linux του τμήματος, υλοποιεί ταξινόμηση και καταμέτρηση λέξεων
3. το mapreduce_index.o object file για linux του τμήματος, υλοποιεί κατασκευή ανεστραμμένου ευρετηρίου (inverted index) για keyword searching
4. τα sort.o και sort.h μια συνάρτηση ταξινόμησης στο κλειδί που διατηρεί αντιστοιχία με ένα πίνακα δεδομένων

Μπορείτε να βρείτε αρχεία εισόδου και υλοποιήσεις για τις 6 συναρτήσεις στο:

http://www.di.uoa.gr/~grad1142/map_reduce/

Οι υλοποιήσεις για distributed sorting και distributed inverted index creation δουλεύουν με απλά text files ως είσοδο, σας δίνονται μερικά σε υποφάκελο στον παραπάνω σύνδεσμο.

Ο γονέας πρέπει να αποδεσμεύει τους πόρους του συστήματος που χρησιμοποιεί (π.χ. named pipes) και να περιμένει την ολοκλήρωση των παιδιών του. Zombie processes δεν πρέπει να υπάρχουν.

Η εργασία θα γραφτεί σε γλώσσα C. Μπορείτε να γράψετε και C++, ωστόσο τα specifications της άσκησης έχουν γραφτεί για C (πχ η συνάρτηση ταξινόμησης). Σε κάθε περίπτωση θα πρέπει η άσκηση να συνοδεύεται με ένα Makefile που να αναλαμβάνει το «χτίσιμο» του προγράμματος σας. Θα θέλαμε εκτός των πλαισίων της άσκησης να μας πείτε ποια από τις 2 γλώσσες προτιμάτε.

Αν και το σημαντικότερο κομμάτι στην διόρθωση αποτελεί η ορθή υλοποίηση όσων περιγράφτηκαν στην εκφώνηση, θα ήταν κρίμα να μην μπορούμε να καταλάβουμε τι κάνετε σε κάποια σημεία του κώδικα σας και να λειτουργήσει αρνητικά στην βαθμολόγηση σας. Φροντίστε να έχετε ευανάγνωστο κώδικα και επεξηγηματικά σχόλια όπου χρειάζονται. Η άσκηση πρέπει να συνοδεύεται με ένα documentation που να περιγράφει περιληπτικά τι χρησιμοποιήθηκε στην άσκηση, τι υλοποιήθηκε και τι όχι και να εξηγεί τη λογική του προγράμματος σας. Οδηγίες για την υποβολή της άσκησης υπάρχουν στη ιστοσελίδα του μαθήματος.

Τέλος η απόδοση του προγράμματος σας μπορεί να λειτουργήσει θετικά ή αρνητικά στην βαθμολόγηση σας. Για να αποφύγετε κλασσικά λάθη που μπορεί να περιορίσουν την απόδοση του προγράμματος σας, προσπαθήστε να χρησιμοποιήσετε τις κλήσεις συστήματος για block δεδομένων και όχι για μικρά δεδομένα.