Dryad - Distributed data-parallel system

Το Dryad αποτελεί μια μηχανή εκτέλεσης εφαρμογών κατανεμημένου προγραμματισμού. Για την υλοποίηση των εργασιών του χρησιμοποιεί γράφους ροής δεδομένων με κόμβους, που συνιστούν απλοποιημένα προγράμματα, συνδεόμενων μέσω ακμών, που αποτελούν τα κανάλια επικοινωνίας.

Η παραλληλοποίηση στο Dryad επιτυγχάνεται με την ταυτόχρονη εκτέλεση των προγραμμάτων των κόμβων σε διαφορετικούς υπολογιστές ή σε πολλαπλές CPUs του ίδιου υπολογιστή. Κατά τον χρόνο εκτέλεσης, ο γράφος που δημιουργείται τροποποιείται δυναμικά ώστε να χρησιμοποιηθούν αποτελεσματικά όλοι οι διαθέσιμοι πόροι.

Στόχος του Dryad είναι η διευκόλυνση των προγραμματιστών στην ανάπτυξη αποτελεσματικών εφαρμογών παράλληλου και κατανεμημένου προγραμματισμού στα πλαίσια της αυξανόμενης ανάγκης για αντιμετώπιση των δεδομένων μεγάλου όγκου.

Σε αντίθεση με παρόμοια συστήματα, το Dryad επιτρέπει τον πλήρη έλεγχο του γράφου και την δυνατότητα χρήσης πολλαπλών εισόδων και εξόδων.

Μια εργασία(job) αποτελείται από έναν κατευθυνόμενο μη κυκλικό γράφο όπου οι κατακερματισμένες λειτουργίες δεν επηρεάζουν το σύνολο των υπολογισμών. Κατά την εκτέλεσή της, μια πεπερασμένη ακολουθία αντικειμένων μεταφέρεται μέσω των καναλιών στους κόμβους. Κάθε εργασία εκτελείται υπό την εποπτεία ενός Job Manager(JM). Αυτός περιέχει τον κώδικα της εφαρμογής για την σχεδίαση του γράφου αλλά και τον κώδικα για την δρομολόγηση της εργασίας προς τους διαθέσιμους πόρους. Δεν αποτελεί ενδιάμεσο μεσάζοντα στην ανταλλαγή δεδομένων και είναι υπεύθυνος μόνο για ελεγκτικές αποφάσεις. Η συστάδα των υπολογιστών περιέχει έναν Name Sever(NS) αρμοδιότητα του οποίου είναι να απαριθμεί όλους τους διαθέσιμους υπολογιστές και να γνωστοποιεί την θέση τους για την διευκόλυνση της χρονοδρομολόγησης. Ένας Daemon(D) λειτουργεί ως διακομιστής μεσολάβησης μέσω του JM και των κόμβων.

Για την κατανόηση του τρόπου λειτουργίας χρησιμοποιείται ένα παράδειγμα SQL ερωτήματος στο οποίο ζητούμενο είναι να βρεθούν όλα τα αντικείμενα που έχουν γειτονικά αντικείμενα σε χρόνο 30 arc δευτερολέπτων έτσι ώστε έστω ένα από τα γειτονικά αντικείμενα να έχει παρόμοιο χρώμα με το αρχικό. Για την υλοποίηση χρησιμοποιούνται 2 πίνακες, ο photoObjAll που περιέχει όλα τα αντικείμενα (354,254,163καταχωρίσεις) τα οποία κωδικοποιούνται μέσω του objID , περιεχομένου του χρώματος το οποίο διαχωρίζεται σε 5 ομάδες u, g, r, i και z. Ο δεύτερος πίνακας neighbors περιέχει τα γειτονικά αντικείμενα σε χρόνο 30 arc sec (2,803,165,372 καταχωρίσεις). Η παράμετρος “Mode” επιλέγει μόνο τα αρχικά αντικείμενα ενώ ο τελεστής “ < “ χρησιμοποιείται προς αποφυγή διπλοτύπων. Παραλείπονται αχρησιμοποίητες στήλες προς αποφυγή μεταφοράς βάσης δεδομένων τεραστίου όγκου.

Για τον υπολογισμό με το dryad συμπιέστηκαν οι δυο πίνακες σε δυο δυαδικά αρχεία “ugriz.bin” και “neighbors.bin,” ταξινομημένα με τον ίδιο τρόπο όπως και οι πίνακες. Τα αρχεία χωρίστηκαν σε n ισοδύναμα τμήματα Un και Nn με βάση το objID. Αυτά ενώθηκαν βάσει των παραμέτρων < και p.mode=1 δημιουργώντας την ένωση - κόμβο X. Οι κόμβοι D χωρίζουν τις εξόδους τους με βάση το neighborObjID και τις στέλνουν ως εισόδους στους κόμβους M όπου ενώνονται μη ντετερμινιστικά και στη συνέχεια ταξινομούνται στους κόμβους S. Οι έξοδοι σχηματίζουν την ένωση Y. Έπειτα κατακερματίζονται μέσω του πίνακα H του οποίου η απαρίθμηση οδηγεί στο τελικό αποτέλεσμα.

Ένας γράφος Dryad μπορεί να περιγραφεί με την βοήθεια ενός ειδικά διαμορφωμένου λεξιλογίου όπου βασικό αντικείμενο είναι ένας άκυκλος γράφος G = <Vg, Eg, Ig,Og>. Το G περιέχει μια αλληλουχία κόμβων Vg, ένα σύνολο κατευθυνόμενων ακμών Eg και δύο σύνολα Ig και Og που προσδιορίζουν τους κόμβους εισόδου(Ig) και εξόδου(Og). Ένας γράφος παράγεται από την δημιουργία ενός κόμβου ως εξής G =<(v), Ø, {v}, {v}>. Έπειτα κλωνοποιείται σε έναν νέο γράφο με k αντίγραφα του εαυτού του με το σύμβολο ^ ως C = G^k. Για την προσθήκη ακμών χρησιμοποιείται μια λειτουργία C = A ◦ B. Θα έχουμε δύο διαφορετικούς τύπους ανάλογα με τον αριθμό των εισερχόμενων κόμβων και ακμών του A και του B. Εάν A >= B θα δημιουργηθεί ένας γράφος με μια ακμή εισερχόμενη ή εξερχόμενη ενώ εάν A >> B θα δημιουργηθεί ένας διμερής γράφος όπου οι ακμές είναι πολλαπλές.
Μπορούμε να συνενώσουμε δύο γράφους με την λειτουργία C = A || B.

Η υλοποίηση των καναλιών ανταλλαγής δεδομένων γίνεται με τη χρήση ενός αρχείου (temporary file). Ο παραγωγός γράφει στον δίσκο και ο καταναλωτής διαβάζει το αρχείο. Το πρωτόκολλο μεταφοράς μπορεί να προσδιοριστεί από τον χρήστη κατά την δημιουργία του συνόλου ακμών. Οι διαθέσιμες επιλογές είναι : File (προεπιλεγμένη), TCP pipe και Shared-memory FIFO. Αρχεία εισόδου μεγάλου όγκου κατανέμονται στους υπολογιστές του δικτύου και στη συνέχεια δημιουργείται ένας γράφος G = <V_P , □, □, V_P > όπου το Vp αποτελεί τους κόμβους που δείχνουν στα διάφορα κομμάτια της εισόδου. Παρόμοια κατά την έξοδο, τα κομμάτια παράγουν ένα κατανεμημένο αρχείο.

Για την διευκόλυνση του JM οι κόμβοι του γράφου διαχωρίζονται σε στάδια τα οποία μπορεί να συνδέονται είτε με τον τελεστή >= είτε με τον τελεστή >>.

Ένα πρόγραμμα κόμβου βασίζεται σε αντικείμενα και κλάσεις της γλώσσας προγραμματισμού C++, παρόλα αυτά οι διεπαφές του είναι αρκετά σαφείς ώστε να μπορούν οι προγραμματιστές να δημιουργήσουν κόμβους και σε άλλες γλώσσες. Εκτελούνται μέσω μιας βιβλιοθήκης εκτέλεσης η οποία δέχεται από τον JM πληροφορίες για το κόμβο που πρόκειται να εκτελεστεί και τις συνδεδεμένες σε αυτόν ακμές. Οι readers και writers των καναλιών περιέχονται σε μια μέθοδο Main. Ο κόμβος μπορεί να ενημερώσει τον JM για τυχόν errors και για την πρόοδο των καναλιών επικοινωνίας. Επιπρόσθετα, χρησιμοποιείται ένα κοινόχρηστο thread pool για όλα τα κανάλια για εφαρμογές read,write,serialization και deserialization.

Για την εκτέλεση της εργασίας, ο JM περιέχει έναν δρομολογητή που καταγράφει την κατάσταση και το ιστορικό κάθε κόμβου, καθένας από τους οποίους μπορεί να εκτελεστεί πολλαπλές φορές. Κάθε περίπτωση ταυτοποιείται από έναν αριθμό έκδοσης και το αντίστοιχο ιστορικό εκτέλεσής της. Η εργασία εκτελείται υπό την υπόθεση πως είναι η μόνη που τρέχει την δεδομένη στιγμή στο δίκτυο και θεωρείται ολοκληρωμένη στην περίπτωση που όλοι οι κόμβοι της έχουν ολοκληρωθεί επιτυχώς. Εάν όμως κάποιος κόμβος υπερβεί ένα όριο επανεκτέλεσης τότε η εργασία θεωρείται αποτυχημένη.

Η πολιτική σφάλματος εικάζει πως όλα τα προγράμματα-κόμβοι είναι ντετερμινιστικά. Ανάλογα με την περίπτωση αποτυχίας της εκτέλεσης ενός κόμβου, ο JM ενημερώνεται απευθείας ή μέσω του D. Οι κόμβοι που έχουν ήδη αποτύχει σύμφωνα με το ιστορικό τους θέτονται απευθείας προς επανεκτέλεση. Οι συνθέσεις κόμβων αποτυγχάνουν ομαδικά στην περίπτωση που αποτύχει έστω και ένας. Σε αυτό εξυπηρετούν τα διάφορα στάδια κόμβων, καθώς ελέγχεται ευκολότερα σε ποιο συγκεκριμένα απέτυχε η εργασία.

Όσον αφορά την βελτίωση του χρόνου εκτέλεσης, είναι σημαντικό να δρομολογούνται οι κόμβοι στον ίδιο υπολογιστή ή στο ίδιο τμήμα που υπάρχουν τα δεδομένα εισόδου

τους. Επιπλέον, μπορούν να συμβάλλουν δέντρα συνάθροισης με την προσθήκη εσωτερικών κόμβων που θα διαβάζουν δεδομένα από υποσύνολα της εισόδου που βρίσκονται πιο κοντά στο συγκεκριμένο σημείο του δικτύου. Γενικότερα η δυναμική βελτιστοποίηση μπορεί να αποδειχθεί αποτελεσματικότερη από μια πρώιμη στατική ομαδοποίηση των δεδομένων.
Στη συνέχεια επιδιώκεται μια δοκιμαστική αξιολόγηση του Dryad μέσα από δύο διαφορετικά πειράματα, με το πρώτο να συγκρίνει την απόδοση του με αυτή ενός SQL server, και το δεύτερο να εκτελεί μια απλή map reduce εργασία data-mining.
Χρησιμοποιήθηκαν δυο εκδοχές του γράφου in-memory" και "two-pass". Τα αποτελέσματα του πρώτου πειράματος έδειξαν πως το two-pass λειτουργεί σε όλα τα μεγέθη συστάδων με γραμμική ταχύτητα ενώ το in-memory από ένα σημείο και πάνω με ταχύτητα σχεδόν διπλάσια του πρώτου. Τελικά, το πρόγραμμα του Dryad τρέχει αρκετά γρηγορότερα από την συγκρινόμενη εκτέλεση του SQLServer. Στο δεύτερο πείραμα ο πρώτος γράφος που δημιουργείται είναι ανεπαρκής για μεγάλο όγκο δεδομένων αλλά αφού βελτιωθεί κάνει τον υπολογισμό σε ένα σύνολο 33,375,616,713 Bytes σε χρόνο 11 λεπτών και 30 δευτερολέπτων.

Μετέπειτα, αναλύεται η δυνατότητα ανάπτυξης κώδικα ανωτέρου επιπέδου με βάση το Dryad, με παραδείγματα την γλώσσα Nebula που επιτρέπει στον χρήστη την προσδιορισμό ενός υπολογισμού σε στάδια, την εφαρμογή SQL Server Integration Services (SSIS) που μετατρέπει τους γράφους της σε Dryad γράφους αλλά και την προσαρμογή των ερωτημάτων SQL ώστε να μεταγλωττίζονται απευθείας σε γράφους Dryad, γεγονός που αποτελεί μελλοντικό στόχο.

Καταλήγοντας στα συμπεράσματα, βλέπουμε πως το Dryad αποτελεί μια μηχανή γενικής χρήσης με εξαιρετική απόδοση ακόμα και σε σύγκριση με εμπορικά συστήματα ανάλυσης βάσεων δεδομένων. Εξίσου αποτελεσματικά συμπεριφέρεται και στην επεξεργασία δεδομένων terabytes σε μεγαλύτερου όγκου συστάδες σε ελάχιστα λεπτά. Ο στόχος για διευκόλυνση των προγραμματιστών στην δημιουργία μεγάλης κλίμακας κατανεμημένων εφαρμογών είναι επιτυχής καθώς τους απαλλάσσει από την αυστηρή αναπαράσταση του κώδικά τους σε βήματα. Παρέχει τέλος την ελευθερία στον προσδιορισμό της μεταφοράς δεδομένων που μπορεί να προσφέρει σημαντικά στην απόδοση με ελάχιστο κόστος.