



ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

**Ελαστικές μέθοδοι συμπίεσης για ερωτήματα αναλυτικής
επεξεργασίας**

Ιωάννης Ε. Φούφουλας

Επιβλέπων : Γιάννης Ιωαννίδης, Καθηγητής

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2014

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλγόριθμοι συμπίεσης σχεσιακών δεδομένων

Ιωάννης Ε. Φούφουλας

A.M.: M1106

ΕΠΙΒΛΕΠΩΝ: Γιάννης Ιωαννίδης, Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: Αλέξης Δελής, Καθηγητής
Γιάννης Ιωαννίδης, Καθηγητής

Οκτώβριος 2014

ΠΕΡΙΛΗΨΗ

Αντικείμενο της εργασίας αυτής είναι η υλοποίηση αλγορίθμων συμπίεσης σχεσιακών δεδομένων. Τα δεδομένα που μας απασχολούν χρησιμοποιούνται σε κατανεμημένα συστήματα στόχος των οποίων είναι η αναλυτική επεξεργασία τους (OLAP). Σε αυτά τα συστήματα σημαντική διαδικασία είναι το κόστος μεταφοράς των δεδομένων πάνω στο δίκτυο καθώς και η διάταξή τους στο δίσκο, που επηρεάζει τον χρόνο εκτέλεσης των σαρώσεων που εκτελούνται πολύ συχνά. Στα πλαίσια της εργασίας εξετάζουμε τις διαφορετικές διατάξεις των σχεσιακών δεδομένων στο δίσκο οι οποίες οδηγούν σε βελτιστοποίηση των ερωτημάτων, καθώς και τη συμπίεση τους η οποία μπορεί να μειώσει το κόστος μεταφοράς των δεδομένων. Προτείνουμε παραμετροποιήσιμους αλγόριθμους που προσφέρουν συμφέρουσες λύσεις ανάμεσα στο βαθμό συμπίεσης και στο χρόνο εκτέλεσής τους, ώστε να επιλέγονται κατά περίπτωση οι καλύτερες παράμετροι ανάλογα με την ταχύτητα του δικτύου και την κατάστασή του την δεδομένη στιγμή.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Συστήματα αναλυτικής επεξεργασίας δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Συμπίεση, διάταξη δεδομένων, αναλυτική επεξεργασία, μεταφορά δεδομένων, σειριοποίηση

ABSTRACT

The goal of this thesis is the implementation of compression algorithms for distributed systems that are used for online analytical processing (OLAP) . We process relational data and we pay attention to 2 critical issues :

- Data Layout. The data layout is very important since it can affect query execution times. OLAP systems use mainly simple scans so we need appropriate layouts for such queries.
- Data movement. In these systems large amounts of data are transferred over the network so a high compression will reduce significantly the data movement cost.

During this thesis we examine different data layouts and compression algorithms that are used by current systems and we propose some new algorithms and approaches. We produce parameterized algorithms which offer tradeoffs between compression level and execution times, so that the system on top may select options according to the network speed and the load at every time.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Online Analytical Processing Systems

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Compression, Data Layout, Online analytical processing (OLAP), Data transfer, Serialization

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ	9
1. ΕΙΣΑΓΩΓΗ.....	10
2. ΔΙΑΤΑΞΗ ΔΕΔΟΜΕΝΩΝ ΣΤΟ ΔΙΣΚΟ ΚΑΙ ΑΛΓΟΡΙΘΜΟΙ ΣΥΜΠΙΕΣΗΣ ΣΧΕΣΙΑΚΩΝ ΔΕΔΟΜΕΝΩΝ	13
2.1 Διάταξη δεδομένων στο δίσκο.....	13
2.2 Αλγόριθμοι συμπίεσης σχεσιακών δεδομένων.....	16
3. ΑΛΓΟΡΙΘΜΟΙ ΣΥΜΠΙΕΣΗΣ ΔΕΔΟΜΕΝΩΝ.....	17
3.1 Κώδικες Lempel-Ziv.....	17
3.2 Κωδικοποίηση Huffman.....	18
3.3 Αριθμητική κωδικοποίηση.....	19
3.4 Burrows-Wheeler Transform (BWT)	21
4. ΑΛΓΟΡΙΘΜΟΙ ΣΥΜΠΙΕΣΗΣ ΣΧΕΣΙΑΚΩΝ ΔΕΔΟΜΕΝΩΝ	23
4.1 Αλγόριθμοι σειριοποίησης	23
4.2 Αλγόριθμοι συμπίεσης	28
4.2.1 Sorted Dictionary per Column (SdicC).....	28
4.2.1 Structured Partition Attribute Compression (SPAC)	31
4.2.1 Correlation SPAC (cSPAC).....	33
5. ΛΕΠΤΟΜΕΡΕΙΕΣ ΥΛΟΠΟΙΗΣΗΣ	38
6. ΠΕΙΡΑΜΑΤΑ	40
7. ΠΡΟΤΑΣΕΙΣ ΓΙΑ ΤΟ ΜΕΛΛΟΝ	52

8. ΣΥΜΠΕΡΑΣΜΑΤΑ	54
ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ	55
ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ	56
ΑΝΑΦΟΡΕΣ	57

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

Σχήμα 1: Κόστος μεταφοράς δεδομένων	11
Σχήμα 2: Διάταξη δεδομένων ανά σειρά	13
Σχήμα 3: Διάταξη δεδομένων ανά στήλη	14
Σχήμα 4: Σελίδα PAX.....	15
Σχήμα 5: Παραλλαγές οικογένειας Lempel-Ziv	18
Σχήμα 6: Παράδειγμα εφαρμογής BWT	22
Σχήμα 7: Διάγραμμα σύγκρισης αλγορίθμων σειριοποίησης (μέγεθος αρχείου).....	26
Σχήμα 8: Διάγραμμα σύγκρισης αλγορίθμων σειριοποίησης (ταχύτητα)	27
Σχήμα 9: Σελίδα SdicC	29
Σχήμα 10: Σελίδα διαφορών SPAC	31
Σχήμα 11: Διάγραμμα σύγκρισης αλγορίθμων γενικής συμπίεσης σε πεδία συμβολοσειρών	42
Σχήμα 12: Διάγραμμα σύγκρισης αλγορίθμων γενικής συμπίεσης σε πεδία αριθμητικών τιμών.....	44
Σχήμα 13: Διάγραμμα σύγκρισης αλγορίθμων συμπίεσης σχεσιακών δεδομένων (TPC-H)	47
Σχήμα 14: Διάγραμμα σύγκρισης αλγορίθμων συμπίεσης σχεσιακών δεδομένων (BMRB).....	48
Σχήμα 15: Διάγραμμα σύγκρισης συσχετίσεων πεδίων σε αρχείο SPAC 2 σελίδων	49
Σχήμα 16: Διάγραμμα σύγκρισης συσχετίσεων πεδίων σε πίνακες πολλών σελίδων ...	50

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

Πίνακας 1: Σύγκριση αλγορίθμων σειριοποίησης (μέγεθος αρχείου)	25
Πίνακας 2: Σύγκριση αλγορίθμων σειριοποίησης (ταχύτητα).....	26
Πίνακας 3: Παράδειγμα στήλης.....	28
Πίνακας 4: Πίνακας συσχετιζόμενων πεδίων	33
Πίνακας 5: Πλήθος διαμερίσεων	36
Πίνακας 6: Παράμετροι πειραμάτων	40
Πίνακας 7: Σύγκριση αλγορίθμων γενικής συμπίεσης σε πεδία συμβολοσειρών.....	41
Πίνακας 8: Σύγκριση αλγορίθμων γενικής συμπίεσης σε πεδία αριθμητικών τιμών	43
Πίνακας 9: Σύγκριση αλγορίθμων σειριοποίησης (1)	45
Πίνακας 10: Σύγκριση αλγορίθμων σειριοποίησης (2)	45

ΠΡΟΛΟΓΟΣ

Η εργασία αυτή πραγματοποιήθηκε στο τμήμα Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Αθηνών υπό την επίβλεψη του καθηγητή Γιάννη Ιωαννίδη. Όλες οι υλοποιήσεις έγιναν στα πλαίσια της ανάπτυξης του συστήματος EXAREME που αναπτύσσεται από προπτυχιακούς, μεταπτυχιακούς φοιτητές και υποψήφιους διδάκτορες του τμήματος. Πολύτιμη συνεισφορά στην εξέλιξη της διπλωματικής αυτής είχε ο υποψήφιος διδάκτωρ του τμήματος Herald Killari , και ο ερευνητής του τμήματος Λευτέρης Σταματογιαννάκης.

1. ΕΙΣΑΓΩΓΗ

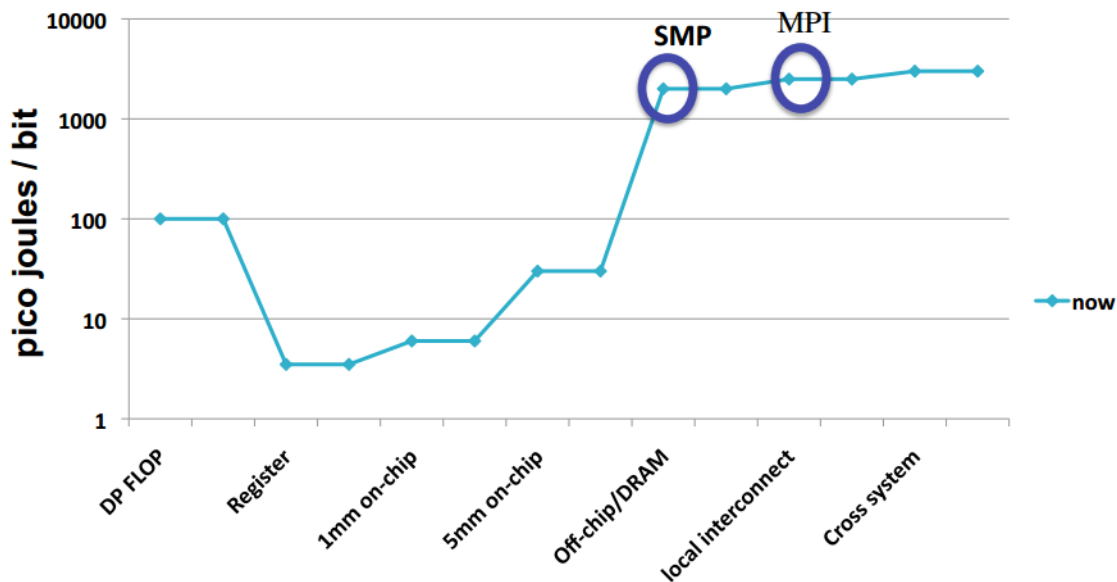
Η χρησιμοποίηση δεδομένων στη λήψη σωστών, έγκυρων, και έγκαιρων αποφάσεων έχει αναχθεί σε «εκ των ουκ άνευ» παράγοντα επιτυχίας για τις περισσότερες σύγχρονες επιχειρήσεις και οργανισμούς. Ταυτόχρονα, τα τελευταία χρόνια, με την ανάπτυξη νέων τεχνολογιών και εφαρμογών – όπως η εξάπλωση των κοινωνικών δικτύων, η εκτεταμένη χρήση smart phones, η εγκατάσταση αισθητήρων κ.α. – ο όγκος και η μορφή των δεδομένων έχει αλλάξει δραματικά, ενώ οι δυνατότητες ανάλυσης και επεξεργασίας αυτών είναι εντυπωσιακές. Αυτό επέφερε μεγάλες αλλαγές στο πως αντιλαμβάνονται τη λήψη αποφάσεων οι εταιρίες και οργανισμοί. Πλέον, τις περισσότερες φορές, αντί να αναπτύσσουν μοντέλα, ξεκινούν από την ανάλυση των δεδομένων που κατέχουν, τα συνδυάζουν με αλλά που συλλέγουν από διάφορες πηγές και τα μοντέλα προκύπτουν αυτόματα με τη μορφή προτύπων. Όπως διατυπώνει περιεκτικά το Wired Magazine [Αύγουστος 2008]: «Η αναζήτηση της γνώσης άρχιζε με ‘μεγάλες’ θεωρίες. Πλέον ξεκινά με τεράστιους όγκους δεδομένων.» Οι όροι Big Data, Business Analytics και Data Science βρίσκονται πλέον στο επίκεντρο των δραστηριοτήτων των εταιριών και οργανισμών, ανεξαρτήτως μεγέθους.

Ένας απλός ορισμός που θα μπορούσε να χρησιμοποιηθεί για την περιγραφή των Μεγάλων Δεδομένων (Big Data) είναι ο εξής: «Managing, analyzing and mining data, no matter how large it is, what format it has and how fast is produced.» Όπως μπορεί να παρατηρήσει κανείς εύκολα, δεν αφορά το μέγεθος, τον τύπο ή το είδος των δεδομένων, αλλά ένα σύνολο διαδικασιών που απαιτούν γνώσεις πληροφορικής, στατιστικής, μηχανικής εκμάθησης και διοίκησης επιχειρήσεων, που απλά εμπλέκουν και χρησιμοποιούν δεδομένα. Πολλές φορές αναφέρεται στη σχετική βιβλιογραφία ότι τα Μεγάλα Δεδομένα χαρακτηρίζονται από τα 5Vs: Volume, Variety, Velocity, Veracity and Value. Ο μεγάλος όγκος (Volume) δεδομένων χαρακτηρίζει τις μοντέρνες εφαρμογές, ενώ τα δεδομένα έρχονται σε διάφορες μορφές, όπως σχεσιακά δεδομένα, εικόνα, ήχος, video. Τέλος, η παραγωγή των δεδομένων μπορεί να γίνεται σε πολύ μεγάλους ρυθμούς (π.χ. tweets, financial streams, αισθητήρες).

Η ανάγκη διαχείρισης των δεδομένων σε εφαρμογές Big Data οδήγησε στην ανάπτυξη μίας νέας γενιάς συστημάτων, μοντέλων και προγραμματιστικών εργαλείων – που ακόμα βρίσκεται σε εμβρυακό στάδιο- όπως: Map Reduce, Hadoop, NoSQL, Cloud κ.α., τεχνολογίες που επιτρέπουν την παράλληλη επεξεργασία δεδομένων σε μεγάλη κλίμακα.

Καθώς τα συστήματα αυτά χρειάζονται πολλά μηχανήματα για να λειτουργήσουν, υπάρχει μεγάλη πιθανότητα σφάλματος, η οποία θα πρέπει να περιοριστεί. Οπότε είναι σημαντικό τα συστήματα αυτά να είναι ανεκτικά σε λάθη (fault-tolerant). Ταυτόχρονα, η αξιοποίηση αυτών των δεδομένων για την παραγωγή analytics απαιτούν ικανότητες και γνώσεις σε ένα ευρύ πεδίο αντικειμένων, όπως στατιστική, τεχνητή νοημοσύνη και τεχνικές μηχανικής εκμάθησης, επιχειρησιακή έρευνα, διοίκηση επιχειρήσεων, κ.α.

Στα συστήματα κατακεντρωμένης επεξεργασίας (distributed systems) αυτού του τύπου σημαντική διαδικασία είναι η μεταφορά δεδομένων (data transfer). Στην επόμενη εικόνα φαίνεται το κόστος της μεταφοράς δεδομένων όπως παρουσιάζεται από τον Horst Simon [1].



Σχήμα 1. Κόστος μεταφοράς δεδομένων

Αυτό που φαίνεται στο διάγραμμα είναι ότι είναι σημαντικό το κόστος μεταφοράς δεδομένων πάνω στο τοπικό δίκτυο. Για να γίνει η μεταφορά αυτή με αποτελεσματικότερο και γρήγορο τρόπο είναι απαραίτητη η ανάπτυξη ενός ευέλικτου αλγορίθμου συμπίεσης των δεδομένων.

Ιδανικά, ο αλγόριθμος θα πρέπει να αποφασίζει ανάμεσα στον βαθμό συμπίεσης και στην ταχύτητα ανάλογα με την κατάσταση και την διαθέσιμη ταχύτητα του δικτύου την δεδομένη

στιγμή. Επιπλέον, επειδή τα δεδομένα αυτά θα πρέπει μετά την μεταφορά τους να υποστούν επεξεργασία θα πρέπει η διάταξή τους (layout) να βοηθάει να εκτελούνται οι επερωτήσεις αποδοτικά και ιδανικά με αποσυμπίεση μόνο των απαραίτητων τμημάτων. Η εκτέλεση αλγορίθμων συμπίεσης εξωτερικά της βάσης δεν δίνουν τα επιθυμητά αποτελέσματα καθώς δεν εκμεταλλεύονται πλήρως τα χαρακτηριστικά των πινάκων και των τιμών τους και απαιτείται πλήρης αποσυμπίεση της βάσης πριν την εκτέλεση επερωτήσεων.

Σε αυτή την εργασία θα παρουσιάσουμε έναν νέο ευέλικτο αλγόριθμο συμπίεσης σχεσιακών δεδομένων για συστήματα που προσφέρουν αναλυτική επεξεργασία δεδομένων (analytical processing - OLAP). Οι στόχοι του αλγορίθμου αυτού περιγράφονται παρακάτω:

- Είσοδος και έξοδος ροών δεδομένων (streamed input/output).
- Δυνατότητα πολλών επιλογών συμπίεσης ανάμεσα στον βαθμό συμπίεσης και στην ταχύτητα ώστε να μπορεί να επιλέγεται η πιο συμφέρουσα με βάση την κατάσταση του δικτύου κάθε στιγμή.
- Δυνατότητα απευθείας επεξεργασίας των συμπιεσμένων δεδομένων χωρίς αποσυμπίεση ολόκληρου του πίνακα.
- Διάταξη αποθηκευμένων δεδομένων που οδηγεί σε βελτιστοποίηση των σαρώσεων που είναι πολύ βασικές για αναλυτική επεξεργασία δεδομένων.

Στην συνέχεια της εργασίας, στο κεφάλαιο 2 θα κάνουμε μια ανασκόπηση υπαρχόντων προσεγγίσεων για τους τρόπους διάταξης των σχεσιακών δεδομένων, ενώ θα αναφερθούμε στους αλγορίθμους συμπίεσης σχεσιακών δεδομένων που χρησιμοποιούνται από άλλα συστήματα. Στην επόμενη ενότητα θα αναφερθούμε γενικά σε αλγορίθμους συμπίεσης (gzip / bzip κλπ). Στο κεφάλαιο 4 θα περιγραφούν οι αλγόριθμοι συμπίεσης που έχουμε υλοποιήσει στα πλαίσια της παρούσας εργασίας, ενώ στο κεφάλαιο 5 αναλύονται οι λεπτομέρειες υλοποίησης. Τέλος στο 6ο κεφάλαιο παρουσιάζουμε πειραματικά αποτελέσματα των αλγορίθμων και ακολουθούν τα οι ιδέες για το μέλλον.

2. ΔΙΑΤΑΞΗ ΔΕΔΟΜΕΝΩΝ ΣΤΟ ΔΙΣΚΟ ΚΑΙ ΑΛΓΟΡΙΘΜΟΙ ΣΥΜΠΙΕΣΗΣ ΣΧΕΣΙΑΚΩΝ ΔΕΔΟΜΕΝΩΝ

Στο κεφάλαιο αυτό θα αναφέρουμε τους τρόπους διάταξης σχεσιακών δεδομένων στο δίσκο που χρησιμοποιούνται από διάφορα συστήματα και αντίστοιχους αλγόριθμους συμπίεσης.

2.1 Διάταξη δεδομένων στο δίσκο

Ανα σειρά (Row store)

Είναι ο πιο ευρέως χρησιμοποιούμενος τρόπος διάταξης δεδομένων στο δίσκο. Χρησιμοποιείται από τα περισσότερα συστήματα διαχείρισης βάσεων δεδομένων (πχ. PostgreSQL [2], SQLite [3], MySQL[4] κ.α.). Σύμφωνα με αυτό τα δεδομένα είναι αποθηκευμένα ανά σειρά όπως φαίνεται στο παρακάτω σχήμα :

Record #	Name	Address	City	State
0003623	ABC	125 N Way	Cityville	PA
0003626	Newburg	1300 Forest Dr.	Troy	VT
0003647	Flotsam	5 Industrial Pkwy	Springfield	MT
0003705	Jolly	529 S 5th St.	Anywhere	NY

Σχήμα 2. Διάταξη δεδομένων ανά σειρά

Αυτός ο τρόπος αποθήκευσης είναι αποδοτικός για ενημερώσεις (transactions) όχι όμως όταν μια επερώτηση αφορά σάρωση μιας κολώνας καθώς τα δεδομένα μιας μεμονωμένης στήλης δεν βρίσκονται συνεχόμενα στον δίσκο.

Ανά στήλη (Column store)

Η μέθοδος αυτή αφορά την αποθήκευση δεδομένων ανά πεδίο, και χρησιμοποιείται από συστήματα όπως η MonetDB [5]. Είναι μέθοδος αποθήκευσης αποτελεσματική σε σαρώσεις λίγων πεδίων, όχι όμως σε σαρώσεις πολλών πεδίων σε μια επερώτηση και κυρίως σε ενημερώσεις καθώς τα δεδομένα σε μια σειρά βρίσκονται σε διαφορετικά σημεία στο δίσκο με αποτέλεσμα να απαιτούνται πολλαπλά χτυπήματα στο δίσκο. Για αυτό το λόγο τα συστήματα αυτά δεν υποστηρίζουν ενημερώσεις. Ένα παράδειγμα column store φαίνεται στο επόμενο σχήμα:

Record #	Name	Address	City	State
0003623	ABC	125 N Way	Cityville	PA
0003626	Newburg	1300 Forest Dr	Troy	VT
0003647	Flotsam	Industrial Pkwy	Springfield	MT
0003705	Jolly	529 S 5th St.	Anywhere	NY

Σχήμα 3. Διάταξη δεδομένων ανά στήλη

Καθολικό λεξικό (Global Dictionary)

Η διάταξη αυτή χρησιμοποιείται από την IBM DB2 [6] και αφορά την αποθήκευση όλων των μοναδικών τιμών του πίνακα σε ένα λεξικό και την χρησιμοποίηση ακεραίων σαν δείκτες σε αυτό το λεξικό. Η μέθοδος αυτή οδηγεί σε υψηλούς βαθμούς συμπίεσης καθώς περιλαμβάνει την αφαίρεση όλων των διπλοτύπων στον πίνακα. Όμως δεν είναι αποδοτική σε επερωτήματα (ενημερώσεις, σαρώσεις) [7] καθώς περιλαμβάνει ένα μεγάλο λεξικό με όλα τα δεδομένα του πίνακα το οποίο θα πρέπει να ανακτάται κάθε φορά εξολοκλήρου.

Τοπικό λεξικό (Local Dictionary)

Η διάταξη αυτή είναι όμοια με την παραπάνω με την διαφορά ότι χρησιμοποιείται ένα λεξικό ανά σελίδα στο δίσκο και όχι ένα καθολικό για όλο το table. Η διαφορά αυτή έχει σαν αποτέλεσμα μικρότερους βαθμούς συμπίεσης από τα καθολικά λεξικά, αλλά παράλληλα πιο αποδοτικές σαρώσεις και ενημερώσεις [7]. Αυτή η διάταξη δεδομένων χρησιμοποιείται από την Oracle [8].

PAX [9]

Το PAX[9] είναι ένας υβριδικός τρόπος διάταξης των σχεσιακών δεδομένων που αποσκοπεί στο να αξιοποιήσει τα πλεονεκτήματα τόσο του column store όσο και του row store. Σύμφωνα με το PAX σε κάθε σελίδα του δίσκου οι εγγραφές αποθηκεύονται ανά πεδίο. Με αυτό επιτυγχάνονται γρήγορες σαρώσεις σε στήλες καθώς οι τιμές κάθε στήλης βρίσκονται μαζί σε κάθε σελίδα, αλλά ταυτόχρονα δεν αυξάνεται ο χρόνος των updates καθώς δεν χρειάζεται να μεταφερθούν πολλαπλές σελίδες από τον δίσκο στη μνήμη δεδομένου ότι όλες οι στήλες είναι μαζί σε μια σελίδα. Μια PAX σελίδα φαίνεται στο παρακάτω σχήμα:

PAX PAGE					
PAGE HEADER			0962	7658	
3859	5523				
Jane	John	Jim	Susan		
30	52	45	20		

Σχήμα 4. Σελίδα PAX

2.2 Αλγόριθμοι συμπίεσης σχεσιακών δεδομένων

Οι αλγόριθμοι συμπίεσης που χρησιμοποιούνται βασίζονται πάνω στις διατάξεις (layouts) που παρουσιάστηκαν στην προηγούμενη υποενότητα. Στη συνέχεια παρουσιάζονται αλγόριθμοι που χρησιμοποιούνται από υπάρχοντα συστήματα Map Reduce.

RCFile [10]

Το RCFile έχει υιοθετηθεί από το Apache Hive [11] και χρησιμοποιείται ευρέως από μια πληθώρα διαδικτυακών πλατφόρμων όπως Facebook, Taobao και Netflix. Το RCFile χρησιμοποιεί το PAX για την διάταξη δεδομένων ενώ χρησιμοποιεί τον Gzip [12] αλγόριθμο συμπίεσης για την συμπίεση των τιμών που αποθηκεύονται στις σελίδες του δίσκου ανά κολώνα. Χαρακτηριστικό του RCFile είναι ότι για την σάρωση μιας στήλης δεν απαιτείται η αποσυμπίεση όλης της σελίδας παρά μόνο του τμήματος που περιλαμβάνει την στήλη αυτή.

Parquet [13]

Το Parquet έχει υιοθετηθεί σαν αλγόριθμος συμπίεσης από το Impala [14] ενώ χρησιμοποιείται από το Twitter. Η διάταξη των δεδομένων στο δίσκο είναι ανά στήλη (column store). Η συμπίεση γίνεται σε κάθε στήλη ξεχωριστά, και οι αλγόριθμοι που χρησιμοποιεί για την συμπίεση είναι ο Snappy [15] και ο Gzip. Και σε αυτή την περίπτωση είναι εφικτή η μερική αποσυμπίεση μόνο των απαραίτητων τμημάτων του αρχείου.

ORCFile [16]

Το ORCFile (optimized RCFile) αποτελεί μια βελτίωση του RCFile. Εκμεταλλεύεται τους τύπους δεδομένων κάθε πεδίου. Η διαφοροποίηση που εισάγει είναι ότι αν μια στήλη περιέχει αλφαριθμητικά χρησιμοποιεί ταξινομημένα τοπικά λεξικά για να αφαιρέσει τα διπλότυπα, επιτυγχάνοντας υψηλούς βαθμούς συμπίεσης με χρήση αλγορίθμων Snappy / Gzip. Απαραίτητη προϋπόθεση για την μέθοδο είναι το σύστημα στο οποίο εφαρμόζεται να υποστηρίζει μόνο έναν τύπο δεδομένων ανά πεδίο. Κάτι τέτοιο δεν είναι εφικτό σε συστήματα διαχείρισης βάσεων δεδομένων όπως η SQLite που επιτρέπει πολλαπλούς τύπους.

3. ΑΛΓΟΡΙΘΜΟΙ ΣΥΜΠΙΕΣΗΣ ΔΕΔΟΜΕΝΩΝ

Η επιλογή του αλγορίθμου που θα χρησιμοποιηθεί για την δυαδική συμπίεση των δεδομένων είναι ένα πολύ σημαντικό βήμα. Υπάρχουν αλγόριθμοι που υπόσχονται υψηλό βαθμό συμπίεσης με μεγάλη πολυπλοκότητα και πιο γρήγοροι αλγόριθμοι με μικρότερα ποσοστά συμπίεσης.

Οι βασικές κατηγορίες αλγορίθμων συμπίεσης περιγράφονται στις παρακάτω υποενότητες:

3.1 Κώδικες Lempel-Ziv

Οι αλγόριθμοι λεξικού αντιπροσωπεύονται από αλγορίθμους της οικογένειας Ziv-Lempel. Ο Jacob Ziv και ο Abraham Lempel με δύο δημοσιεύσεις τους το 1977 και το 1978 (και αντίστοιχους αλγορίθμους LZ77 [17] και LZ78 [18]) έθεσαν τις βάσεις για το σύνολο αυτών των αλγορίθμων. Η βασική τους ιδέα ήταν να αντικαταστήσουν μοτίβα (ακολουθίες από bytes) που έχουν εμφανιστεί ήδη στο κείμενο με ένα δείκτη που δείχνει στην προηγούμενη θέση στο κείμενο όπου είχαν εμφανιστεί. Αποδεικνύεται, μάλιστα, ότι, στο όριο, οι αλγόριθμοι αυτής της οικογένειας τείνουν ασυμπτωτικά στην εντροπία των δεδομένων.

Οι δύο κύριοι διαφοροποιοί παράγοντες μεταξύ των αλγορίθμων της οικογένειας είναι το όριο μέχρι πόσο πίσω μπορεί να φτάσει ο εκάστοτε δείκτης και πόσες υποακολουθίες αλφαριθμητικών χαρακτήρων εντός αυτού του ορίου μπορούν να αποτελέσουν το πεδίο εφαρμογής του δείκτη. Το όριο του δείκτη μπορεί να μην υπόκειται σε περιορισμούς (αυξανόμενο παράθυρο (growing window)) ή να υπόκειται σε περιορισμό σταθερού μεγέθους παραθύρου, δηλαδή παραθύρου συγκεκριμένου αριθμού χαρακτήρων. Αντίστοιχα, οι υποακολουθίες μπορεί να είναι απεριόριστου ή πεπερασμένου μεγέθους. Ανάλογα με τον συνδυασμό των επιλογών επιτυγχάνεται συμβιβασμός μεταξύ ταχύτητας, απαιτήσεων μνήμης και αποδοτικότητας συμπίεσης.

Στον παρακάτω πίνακα [19] απεικονίζονται οι κυριότερες παραλλαγές της οικογένειας Lempel-Ziv

TABLE 8-2 SUMMARY OF PRINCIPAL LZ VARIATIONS

~ LZ77	Ziv and Lempel (1977)	Pointers and characters alternate Pointers indicate a substring in the previous N characters
LZR	Rodeh et al. (1981)	Pointers and characters alternate Pointers indicate a substring anywhere in the previous characters
~ LZSS	Bell (1986)	Pointers and characters are distinguished by a flag bit Pointers indicate a substring in the previous N characters
LZB	Bell (1987)	Same as LZSS, except a different coding is used for pointers
LZH	Brent (1987)	Same as LZSS, except Huffman coding is used for pointers on a second pass
~ LZ78	Ziv and Lempel (1978)	Pointers and characters alternate Pointers indicate a previously parsed substring
~ LZW	Welch (1984)	The output contains pointers only Pointers indicate a previously parsed substring Pointers are of fixed size
LZC	Thomas et al. (1985)	The output contains pointers only Pointers indicate a previously parsed substring
LZT	Tischer (1987)	Same as LZC but with phrases in a LRU list
LZMW [†]	Miller and Wegman (1984)	Same as LZT but phrases are built by concatenating the previous two phrases
LZI	Jakobsson (1985)	The output contains pointers only Pointers indicate a substring anywhere in the previous characters
LZFG	Fiala and Greene (1989)	Pointers select a node in a trie Strings in the trie are from a sliding window

Σχήμα 5. Παραλλαγές οικογένειας Lempel-Ziv

3.2 Κωδικοποίηση Huffman

Η κωδικοποίηση Huffman [20] επιτελείται με ένα αλγόριθμο, μεταβλητού μήκους κώδικα, που χρησιμοποιείται για συμπίεση δεδομένων. Η βασική αρχή του αλγορίθμου έγκειται στην ελαχιστοποίηση του μέσου μήκους του κώδικα $E[L]$. Όμως γνωρίζουμε από την ανισότητα Kraft (Kraft Inequality) [21] ότι, για δυαδικούς κώδικες, τα μήκη των κωδικών λέξεων, l_i , πρέπει να ικανοποιούν τη σχέση $\sum 2^{-l_i} \leq 1$. Μια λογική λύση που προήλθε από τον Shannon ήταν $l_i = \log(1/p_i)$ με p_i την πιθανότητα εμφάνισης των συμβόλων για i ακέραιο (οι πιθανότητες απαρτίζονται από αρνητικές δυνάμεις του δύο). Στη γενική περίπτωση όπου i δεν είναι ακέραιος ο Shannon πήρε τον ακόλουθο τύπο $l_i = \lceil \log(1/p_i) \rceil$. Αρχικά, οι πιθανότητες εμφάνισης των συμβόλων διατάσσονται με φθίνουσα σειρά. Στη συνέχεια, υπολογίζουμε την αθροιστική πιθανότητα κάθε συμβόλου και βρίσκουμε το μήκος κάθε κωδικής λέξης με βάση τη σχέση $l_i = \lceil \log(1/p_i) \rceil$. Τέλος, η κωδική λέξη που αντιστοιχεί στο κάθε σύμβολο της πηγής είναι τα πρώτα l_i κλασματικά bits του

δυναμικού αναπτύγματος της αντίστοιχης αθροιστικής συχνότητας. Μία άλλη οπτική απεικόνιση της ιδέας του Shannon είναι η ανάπτυξη ενός δυαδικού δέντρου από την κορυφή ως τη ρίζα με τη χρήση των μηκών που βρήκαμε. Όμως, έχουμε υποθέσει ότι το μήκος l πρέπει να είναι φυσικός αριθμός για να είναι βέλτιστη η απεικόνισή μας. Ο Huffman λοιπόν έφτιαξε ένα αντίστοιχο δέντρο ξεκινώντας από τη ρίζα με τα πιο απίθανα συμβάντα και πηγαίνοντας προς τα φύλλα, προς τα πιο πιθανά, υλοποιώντας έτσι ένα βέλτιστο κώδικα. Αποδεικνύεται πως το μέσο πληροφοριακό μήκος των δύο κωδικών (Huffman και Shannon) βρίσκεται σε απόσταση έως ένα bit από την εντροπία. Επίσης, αποδεικνύεται ότι, ενώ για συγκεκριμένο σύμβολο πηγής το μήκος της κωδικής λέξης Huffman ενδέχεται να είναι μεγαλύτερο από την κωδική λέξη του κώδικα Shannon για το ίδιο σύμβολο, ο κώδικας Huffman είναι βέλτιστος, δηλαδή το μέσο μήκος συμπίεσης που επιτυγχάνει δεν μπορεί να υπερβεί το μέσο μήκος του κώδικα Shannon. Μάλιστα, ο πλεονασμός της πληροφορίας (η ποσότητα της πληροφορίας που είναι περιττή για την κατανόησή της) κατά την κωδικοποίηση Huffman δεν υπερβαίνει την τιμή $p + \log[2(\log e)/e] = p + 0.86$, όπου p είναι η πιθανότητα του πιο πιθανού συμβόλου. Δηλαδή, όταν υπάρχουν αρκετά σύμβολα με παρεμφερή συχνότητα εμφάνισης έχουμε πολύ μικρότερο πλεονασμό της πληροφορίας σε σχέση με την κωδικοποίηση Shannon όπου ο πλεονασμός πληροφορίας παίρνει τιμές έως και 1 bit. Αλγόριθμοι που χρησιμοποιούν κωδικοποίηση Huffman είναι οι Gzip/Bzip2[22] και άλλοι.

3.3 Αριθμητική κωδικοποίηση [23]

Η αριθμητική κωδικοποίηση βασίζεται στον αλγόριθμο Shannon-Fano-Elias [24] προκειμένου να αποφευχθεί η διάταξη των πιθανοτήτων εμφάνισης των συμβόλων. Είναι μια εναλλακτική μέθοδος κωδικοποίησης της εντροπίας. Βασίζεται στη λογική της κωδικοποίησης μιας σειράς από σύμβολα και όχι μεμονωμένων συμβόλων

- Για παράδειγμα σε μια ακολουθία συμβόλων $a_1 a_2 \dots a_n$ τα a_i δεν κωδικοποιούνται ξεχωριστά όπως στην περίπτωση της κωδικοποίησης Huffman αλλά ως μια ομάδα συμβόλων

- Όσο περισσότερα σύμβολα ομαδοποιήσουμε τόσο αποτελεσματικότερη κωδικοποίηση έχουμε αλλά τόσο πολυπλοκότερη γίνεται η υλοποίηση του κωδικοποιητή
- Τέλος, στην αριθμητική κωδικοποίηση κατασκευάζεται μόνο ο κώδικας που αντιστοιχεί στην ακολουθία που κωδικοποιείται. Δηλαδή σε αντίθεση με άλλους αλγορίθμους, δεν κατασκευάζονται εκ των προτέρων όλες οι κωδικές λέξεις. Αυτό έχει ως αποτέλεσμα να επιτυγχάνεται σημαντική μείωση της πολυπλοκότητας.

Η αριθμητική κωδικοποίηση επιτυγχάνει συμπίεση κοντά στην εντροπία επειδή κωδικοποιεί από κοινού ομάδες πολλών συμβόλων. Μάλιστα, στη γενική περίπτωση κωδικοποιούνται από κοινού όλα τα σύμβολα του αρχείου που συμπιέζεται. Δηλαδή, αντιμετωπίζουμε όλη την έξοδο της πηγής ως μία κωδική λέξη, την οποία κωδικοποιούμε σταδιακά. Η αριθμητική κωδικοποίηση αντιστοιχίζει όλη την έξοδο της πηγής σε έναν πραγματικό αριθμό στο διάστημα $[0,1]$. Η δυαδική απεικόνιση αυτού του αριθμού αποτελεί τον κώδικα της εξόδου της πηγής. Δεν απαιτείται να γνωρίζουμε όλη την έξοδο της πηγής όταν αρχίζει η κωδικοποίηση. Ο πραγματικός αριθμός υπολογίζεται σταδιακά χρησιμοποιώντας διαδοχικά σύμβολα της πηγής για να προσδιοριστεί το υποδιάστημα του $[0,1]$ στο οποίο ανήκει ο αριθμός. Κατά την ανάγνωση νέων συμβόλων της πηγής, το μέγεθος του υποδιαστήματος μικραίνει συνεχώς. Επομένως, τα απαιτούμενα bits για την απεικόνισή του διαρκώς αυξάνονται. Τα πιο πιθανά σύμβολα μειώνουν το μέγεθος του υποδιαστήματος λιγότερο από τα λιγότερο πιθανά σύμβολα και, επομένως, απαιτούν λιγότερα bits για την απεικόνισή τους. Τελικά, καταλήγουμε σε ένα υποδιάστημα το οποίο μπορούμε να περιγράψουμε με οποιονδήποτε πραγματικό αριθμό βρίσκεται μέσα σε αυτό. Η αριθμητική κωδικοποίηση χρησιμοποιείται από εργαλεία συμπίεσης όπως Bzip[22] / LZMA[25]

Σύγκριση αριθμητικής κωδικοποίησης με κωδικοποίηση Huffman

Η κωδικοποίηση Huffman είναι βέλτιστη, εφόσον οι πιθανότητες που χρησιμοποιούμε είναι όλες αρνητικές δυνάμεις του δύο. Βέβαια, αυτό συμβαίνει πρακτικά σπάνια, αλλά ακόμα και σε αυτήν την περίπτωση, ο βαθμός συμπίεσης της αριθμητικής κωδικοποίησης βρίσκεται αρκετά κοντά στην εντροπία. Επίπλέον η κωδικοποίηση Huffman, για μεγάλα αλφάβητα (με άρα μικρή μέγιστη πιθανότητα), οδηγεί σε πολύ καλό βαθμό συμπίεσης, καλύτερο από αυτό της αριθμητικής. Όμως, για μικρά αλφάβητα, η κωδικοποίηση Huffman είναι αρκετά χειρότερη από την αριθμητική, όπως για παράδειγμα στα μοντέλα G3 και G4 για

χρήση τηλεομοιοτύπου (με αλφάβητο λευκό και μαύρο). Τέλος, αποδεικνύεται ότι επειδή στην κωδικοποίηση Huffman έχουμε κάνει κάποιες ιδανικές υποθέσεις, ως επί το πλείστον, η αριθμητική κωδικοποίηση προσφέρει ελαφρά καλύτερο ρυθμό συμπίεσης κατά μέσο όρο και σε ζητήματα προσαρμοστικότητας βρίσκεται σε μη συγκρίσιμα υψηλότερο επίπεδο από την κωδικοποίηση Huffman.

3.4 Burrows-Wheeler Transform (BWT) [26]

Ο BWT στην ουσία δεν αποτελεί οικογένεια αλγορίθμων συμπίεσης αλλά είναι ένας μετασχηματισμός που σαν σκοπό έχει να ταξινομήσει τα δεδομένα με τρόπο τέτοιο ώστε να μπορούν να συμπιεστούν καλύτερα με κωδικοποίηση huffman. Τα βήματα του αλγορίθμου είναι τα ακόλουθα:

1. Πάρε όλες τις δυνατές περιστροφές του προς συμπίεση όρου σαν ένα πίνακα τιμών όπου κάθε σειρά περιλαμβάνει μια περιστροφή και κάθε στήλη ένα byte της περιστροφής
2. Ταξινόμησε τον πίνακα αυτό αλφαριθμητικά
3. Βγάλε σαν έξοδο την τελευταία στήλη.

Στο παρακάτω σχήμα φαίνεται ένα παράδειγμα λειτουργίας του αλγορίθμου.

Transformation			
Input	All Rotations	Sorted List of Rotations	Output Last Column
^BANANA@	^BANANA@ @^BANANA A@^BANAN NA@^BANA ANA@^BAN NANA@^BA ANANA@^B BANANA@^	ANANA@^B ANA@^BAN A@^BANAN BANANA@^ NANA@^BA NA@^BANA ^BANANA@ @^BANANA	BNN^AA@A
(1)	(2)	(3)	(4)

Σχήμα 6. Παράδειγμα εφαρμογής BWT.

Ο Burrows-Wheeler Transform χρησιμοποιείται στο εργαλείο συμπίεσης Bzip2.

Στα σχεσιακά δεδομένα που χρησιμοποιήσαμε για τα πειράματα φαίνεται να λειτουργούν καλύτερα οι αλγόριθμοι της οικογένειας LZ και οι αλγόριθμοι που βασίζονται στην κωδικοποίηση huffman. Η αριθμητική κωδικοποίηση έχει υψηλή πολυπλοκότητα χωρίς να δίνει σημαντικά καλύτερα ποσοστά συμπίεσης, λόγω των μεγάλων αλφαβήτων στα σχεσιακά δεδομένα, ενώ ο Burrows-Wheeler Transform φαίνεται να δίνει σημαντικό όφελος στην συμπίεση των δεδομένων. Τα σχετικά πειράματα βρίσκονται στο 6ο κεφάλαιο.

4. ΑΛΓΟΡΙΘΜΟΙ ΣΥΜΠΙΕΣΗΣ ΣΧΕΣΙΑΚΩΝ ΔΕΔΟΜΕΝΩΝ

Στο κεφάλαιο αυτό θα περιγραφούν οι αλγόριθμοι που αναπτύχθηκαν στα πλαίσια αυτής της διπλωματικής. Όπως έχει αναφερθεί ήδη στόχος είναι η ανάπτυξη δυναμικού αλγορίθμου που θα αποφασίζει ανάμεσα στην υψηλή συμπίεση και την ταχύτητα ενώ θα αποθηκεύει τα δεδομένα σε μια διάταξη που να είναι φιλική σε σαρώσεις. Το πρώτο ζήτημα που προκύπτει είναι οι αλγόριθμοι σειριοποίησης που θα χρησιμοποιηθούν. Στη συνέχεια του κεφαλαίου θα αναλύσουμε τους διαθέσιμους αλγόριθμους σειριοποίησης, και στη συνέχεια θα περιγράψουμε τους αλγόριθμους συμπίεσης που έχουν υλοποιηθεί.

4.1 Αλγόριθμοι σειριοποίησης

Με τον όρο σειριοποίηση (serialization) εννοούμε την διαδικασία μετάφρασης των δομών δεδομένων και των αντικειμένων μιας γλώσσας σε μια μορφή που να μπορεί να αποθηκευτεί στο δίσκο και να ανακτηθεί από αυτόν. Η γλώσσα που χρησιμοποιούμε είναι η Python [27] οπότε για την σειριοποίηση υπάρχουν οι επιλογές που αναλύονται στη συνέχεια.

Python Struct

Μετατρέπει τιμές της python σε δομές της C ώστε να αποθηκευτούν στο δίσκο. Χρησιμοποιείται για την διαχείριση δυαδικών δεδομένων που αποθηκεύονται σε αρχεία ή μεταφέρονται στο δίκτυο.

Δεν μπορεί να χρησιμοποιηθεί για απευθείας σειριοποίηση μιας λίστας ή ενός λεξικού της python καθώς χρησιμοποιείται για κάθε τιμή ξεχωριστά γεγονός που αυξάνει τον χρόνο εκτέλεσης ενώ έχει και υψηλή δυσκολία υλοποίησης.

Python pickle / Cpickle

Το pickle είναι ο πιο διαδεδομένος τρόπος για σειριοποίηση δεδομένων σε python, ενώ το cPickle είναι η υλοποίηση του ίδιου module σε C όπου λειτουργεί έως και 1000 φορές γρηγορότερα. Με το pickle μπορούν να σειριοποιηθούν δυαδικά όλες οι δομές της python (λίστες, λεξικά κλπ) .

Το pickle επιπλέον κρατάει τα αντικείμενα τα οποία έχει σειριοποιήσει ώστε να μην τα ξανασειριοποιήσει στην συνέχεια εφόσον τα ξανασυναντήσει. Με αυτό τον τρόπο αποθηκεύεται κάθε αντικείμενο μια φορά.

Υπάρχει δυνατότητα αποεπιλογής αυτού του χαρακτηριστικού πράγμα που οδηγεί μεν σε μεγαλύτερα αρχεία αλλά ταυτόχρονα σε ακόμα γρηγορότερη σειριοποίηση έως 10 φορές. Καθώς στους αλγόριθμους που θα παρουσιάσουμε στη συνέχεια θα κάνουμε αφαίρεση διπλοτύπων και συμπίεση πάνω στα σειριοποιημένα δεδομένα το pickle θα χρησιμοποιείται χωρίς αυτό το χαρακτηριστικό ώστε να δοθεί έμφαση στην ταχύτητα.

Python marshal

Το marshal λειτουργεί παρόμοια με το pickle με την διαφορά ότι δεν κρατάει τα αντικείμενα που έχει ήδη σειριοποιήσει, αλλά σειριοποιεί τα πάντα. Το marshal είναι λίγο γρηγορότερο από το cPickle καθώς έχει ορισμένες διαφορές στη σειριοποίηση με στόχο την μεγαλύτερη ταχύτητα. Το μειονέκτημα του marshal είναι ότι θα πρέπει να είναι ίδια η έκδοση της python που χρησιμοποιείται για την σειριοποίηση και για το διάβασμα. Η βασική χρήση του marshal είναι για το γράψιμο των .pyc (byte code files) αρχείων της python.

MessagePack

Το MessagePack είναι ένα αποτελεσματικό format για δυαδική σειριοποίηση δεδομένων. Δίνει την δυνατότητα ανταλλαγής δεδομένων σε πολλαπλές γλώσσες όπως και το json. Είναι όμως γρηγορότερο και οδηγεί σε μικρότερα αρχεία. Μπορεί να χρησιμοποιηθεί για να σειριοποιήσει python δομές.

Python Array

Χρησιμοποιείται αποκλειστικά για την σειριοποίηση πινάκων αριθμητικών τιμών. Μπορεί να σειριοποιήσει ακεραίους του 1, 2 και 4 bytes.

Python json/bson

Το python json είναι μια βιβλιοθήκη για json σειριοποίηση των δομών της python ενώ το bson (binary json) είναι παρόμοιο με την διαφορά ότι γράφει τα δεδομένα σε δυαδικά αρχεία. Το bson κάνει γρηγορότερη σειριοποίηση ενώ παράγει μικρότερα αρχεία από το json.

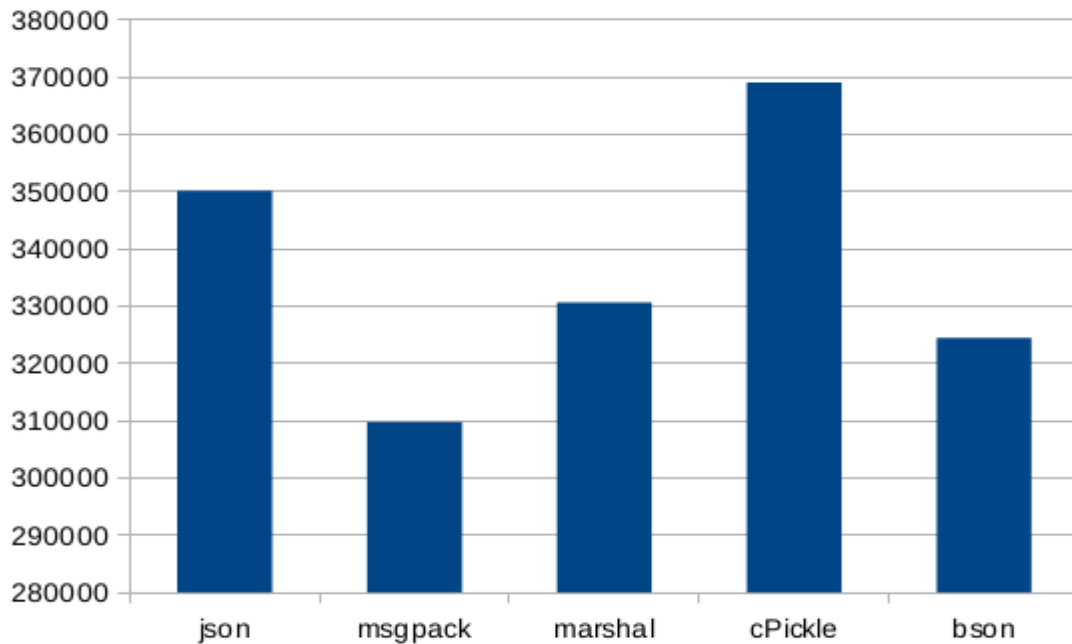
Σύγκριση αλγορίθμων σειριοποίησης

Οι παραπάνω αλγόριθμοι συγκρίθηκαν ως προς το μέγεθος του παραγώμενου αρχείου και ως προς την ταχύτητα γραψίματος και διαβάσματος. Τα πειράματα περιγράφονται στο [28] και έτρεξαν πάνω σε 4096 json αρχεία διαφόρων μεγεθών.

Στον παρακάτω πίνακα και στην γραφική φαίνονται τα μεγέθη των αρχείων που προέκυψαν από τα διάφορα serializations:

Πίνακας 1. Σύγκριση αλγορίθμων σειριοποίησης (μέγεθοςαρχείου)

Json	msgpack	marshal	cPickle	bson
350168 (342M)	309772 (303M)	330660 (323M)	369060 (361M)	324452 (317M)



Σχήμα 7. Διάγραμμα σύγκρισης αλγορίθμων σειριοποίησης (μέγεθος αρχείου)

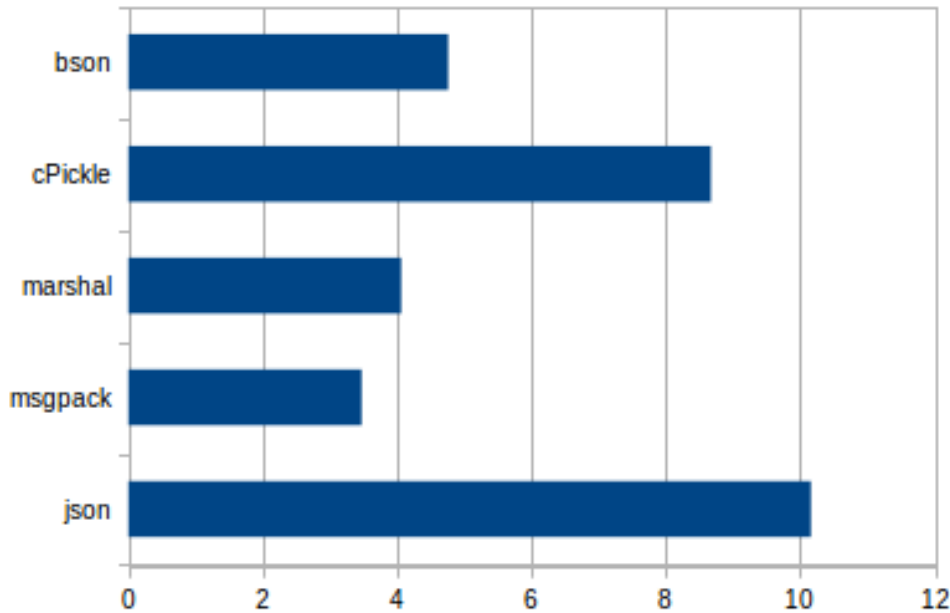
Στον επόμενο πίνακα παρουσιάζονται οι χρόνοι γραψίματος και διαβάσματος:

Πίνακας 2. Σύγκριση αλγορίθμων σειριοποίησης (ταχύτητα)

Library	decoding	encoding	total
Json	6.83s	3.33s	10.16s
Msgpack	0.72s	2.75s	3.47s
Marshal	2.76s	1.30s	4.06s
cPickle	3.86s	4.81s	8.67s

Bson	2.58s	2.18s	4.76s
------	-------	-------	-------

Ακολουθεί γραφική παράσταση για τον συνολικό χρόνο κωδικοποίησης και αποκωδικοποίησης.



Σχήμα 8. Διάγραμμα σύγκρισης αλγορίθμων σειριοποίησης (ταχύτητα)

Όπως φαίνεται από τους πίνακες το msgpack έχει το μικρότερο μέγεθος αρχείου και την πιο γρήγορη αποκωδικοποίηση, ενώ την πιο γρήγορη κωδικοποίηση την βρίσκουμε στο marshal. Συνολικά καλύτερο χρόνο έχει το msgpack.

Ενώ το msgpack φαίνεται ως η πιο συμφέρουσα επιλογή αυτό αναιρείται από το γεγονός ότι κατά την κωδικοποίηση κωδικοποιούμε όλα τα δεδομένα οπότε με το marshal είμαστε πάνω από 2 φορές γρηγορότεροι ενώ στην αποκωδικοποίηση διαβάζουμε μόνο τα τμήματα των σελίδων του δίσκου που ζητάει μια επερώτηση. Στη συνέχεια περιγράφονται οι αλγόριθμοι συμπίεσης που υλοποιήσαμε ενώ για την σειριοποίηση χρησιμοποιείται κυρίως το marshal.

Συγκρίσεις των 2 format στα δικά μας δεδομένα θα δούμε στην ενότητα των πειραμάτων.

4.2 Αλγόριθμοι συμπίεσης

4.2.1 Sorted Dictionary Per Column (SdicC)

Ο αλγόριθμος που παρουσιάζεται εδώ βασίζεται στην PAX διάταξη των δεδομένων στο δίσκο. Συγκεκριμένα κάθε σελίδα που γράφεται στο δίσκο περιέχει συγκεκριμένο πλήθος εγγραφών με την αποθήκευση να γίνεται ανά στήλη. Στο SdicC οι στήλες στις οποίες ο αριθμός των μεμονωμένων τιμών ξεπερνάει το 50% του πλήθους εγγραφών συμπιέζονται με ένα εργαλείο συμπίεσης (gzip/bzip2). Για τις στήλες που έχουν πολλά διπλότυπα (οι μεμονωμένες τιμές είναι λιγότερες του 50% του συνόλου) , χρησιμοποιείται ένα λεξικό στο οποίο κρατούνται οι τιμές αυτές και ταξινομούνται αλφαριθμητικά ώστε να επιτυγχάνεται μεγαλύτερος βαθμός συμπίεσης. Παράλληλα γράφεται και ένας πίνακας με ακεραίους, του 1 ή 2 bytes , ο οποίος περιέχει τους δείκτες στο λεξικό. Η παραπάνω επεξεργασία γίνεται για κάθε σελίδα η οποία περιέχει 65536 εγγραφές έτσι ώστε να αρκούν δείκτες των 2 bytes. Για παράδειγμα θεωρούμε την στήλη :

Πίνακας 3. Παράδειγμα στήλης

A
B
A
Γ
A
B
Γ

Το ταξινομημένο λεξικό που προκύπτει από την κολώνα είναι το {A,B,Γ} και ο πίνακας με τους δείκτες ο [0,1,0,2,0,1,2] . Στο συγκεκριμένο παράδειγμα το μήκος του λεξικού είναι μικρότερο από 256 οπότε ο πίνακας περιέχει ακραίους 1 byte. Στην περίπτωση που το μήκος του λεξικού είναι μεγαλύτερο τότε περιέχονται ακέραιοι των 2 bytes.

Στο αρχείο SdicC υπάρχουν 2 τύποι σελίδων. Η σελίδα επικεφαλίδας και οι σελίδες δεδομένων. Η σελίδα επικεφαλίδας περιέχει στην αρχή ένα byte που υποδεικνύει ότι πρόκειται για επικεφαλίδα και το σχήμα του πίνακα, ενώ οι σελίδες δεδομένων περιέχουν ένα αναγνωριστικό byte στην αρχή, μια επικεφαλίδα (πίνακας ακεραίων) που υποδεικνύει το σημείο στη σελίδα που βρίσκεται κάθε κολώνα και τις κολώνες.

Η διάταξη μιας σελίδας δεδομένων του αρχείου εμφανίζεται παρακάτω:

1	Block Header	
Sorted Dict With Values		Index
Full Column Data		
Full Column Data		
Sorted Dict With Values		Index

Σχήμα 9. Σελίδα SdicC

Στην παραπάνω απεικόνιση παρουσιάζεται μια σελίδα δεδομένων ενός αρχείου SdicC.

Στην αρχή είναι αναμμένο το αναγνωριστικό byte της σελίδας ενώ στην συνέχεια ακολουθεί η κεφαλίδα με πληροφορίες για την θέση κάθε στήλης. Το συγκεκριμένο παράδειγμα αφορά πίνακα με τέσσερις κολώνες οι οποίες ακολουθούν. Η πρώτη και η τέταρτη στήλη πληρούν τις προϋποθέσεις να γραφτούν με ταξινομημένο λεξικό και πίνακα δεικτών.

Σειριοποίηση στο SdicC

Στο SdicC χρησιμοποιούνται διαφορετικά εργαλεία σειριοποίησης για τα δεδομένα και για τους πίνακες δεικτών. Έτσι η σειριοποίηση των δεδομένων και των λεξικών γίνεται με το marshal ενώ η σειριοποίηση της κεφαλίδας και των δεικτών με Python Arrays που είναι υλοποιημένα για ακεραίους.

Συμπίεση στο SdicC

Οι αλγόριθμοι συμπίεσης που χρησιμοποιεί το SdicC είναι ο gzip και ο bzip2. Η επιλογή του αλγορίθμου είναι παραμετροποιημένη όπως και του επιπέδου συμπίεσης. Έτσι ο χρήστης του αλγορίθμου μπορεί να επιλέξει το gzip με χαμηλό επίπεδο συμπίεσης για να δώσει μεγαλύτερη έμφαση στην ταχύτητα ή bzip2 με υψηλό επίπεδο συμπίεσης για έμφαση στην συμπίεση. Επιπλέον μπορεί να χρησιμοποιηθεί διαφορετικός αλγόριθμος συμπίεσης για τα δεδομένα και διαφορετικός για τους δείκτες.

Αποσυμπίεση και σάρωση στο SdicC

Στο SdicC κάθε στήλη σε κάθε σελίδα είναι αυτόνομα συμπιεσμένη οπότε για να γίνει σάρωση σε ένα πεδίο ή αναζήτηση μιας τιμής δεν χρειάζεται διάβασμα από το δίσκο και αποσυμπίεση άλλων σελίδων, ούτε τμημάτων της σελίδας που αφορούν άλλα πεδία. Στη συνέχεια φαίνεται πως λειτουργούν οι απλές σαρώσεις και οι σαρώσεις με φίλτρο στο SdicC.

- Απλή Σάρωση κολώνας. Στην αρχή διαβάζεται η σελίδα από το δίσκο και αποσυμπίεζεται η κεφαλίδα για να αποκτηθεί η θέση της κολώνας στη σελίδα. Στη συνέχεια μόνο η συγκεκριμένη κολώνα (είτε τα δεδομένα, είτε το λεξικό και οι δείκτες) αποσυμπίεζεται και διαβάζονται οι τιμές της.
- Σάρωση με φίλτρο. Η σάρωση με φίλτρο γίνεται με τον ίδιο τρόπο επιλέγοντας τις τιμές που ικανοποιούν τα φίλτρα. Στην περίπτωση που μια κολώνα έχει αποθηκευτεί με λεξικό και δείκτες η αναζήτηση μπορεί να γίνει πιο αποδοτικά με λογαριθμική πολυπλοκότητα σε κάθε σελίδα καθώς το λεξικό είναι ταξινομημένο χωρίς διπλότυπα.

4.2.2 Structured Partition Attribute Compression (SPAC)

Ο αλγόριθμος SPAC αποτελεί μια βελτίωση του SdicC που στοχεύει σε μεγαλύτερους βαθμούς συμπίεσης. Η βελτίωση εφαρμόζεται στα πεδία τα οποία αποθηκεύονται με ταξινομημένα λεξικά. Η αρχική ιδέα είναι ότι εφόσον στις πρώτες 65536 εγγραφές μια κολώνα έχει λιγότερες από 50% μεμονωμένες τιμές είναι πολύ πιθανόν στις επόμενες σελίδες οι τιμές αυτής της κολώνας να μην διαφέρουν σημαντικά. Πράγματι πειραματικά είδαμε ότι στην πρώτη μόνο σελίδα του αρχείου περιέχονται περίπου το 80% των τιμών όλου του πίνακα. Η διαπίστωση αυτή αφορά τον πίνακα lineitem του TPC-H και τη βάση BMRB (fact table) που αφορά βιολογικά δεδομένα. Η ιδέα λοιπόν είναι σε κάθε επόμενη σελίδα του αρχείου να μην γράφεται στον δίσκο όλο το λεξικό αλλά μόνο οι διαφορές του με το προηγούμενο. Έτσι κατά τη διάρκεια του γραψίματος υπάρχει ένα τρέχον λεξικό ανά κολώνα το οποίο συγκρίνεται κάθε φορά με τις τιμές της τρέχουσας σελίδας γράφονται οι διαφορές στο δίσκο, ενώ παράλληλα το τρέχον λεξικό ενημερώνεται με τις διαφορές αυτές.

Στο SPAC η πρώτη σελίδα δεδομένων είναι ίδια με αυτή του SdicC η κάθε επόμενη όμως είναι όπως στο παρακάτω σχήμα:

1	Block Header	
Diff Values	Diff Index	Index
Full Column Data		
Full Column Data		
Diff Values	Diff Index	Index

Σχήμα 10. Σελίδα διαφορών SPAC.

Στην παραπάνω απεικόνιση με τον όρο Diff Values εννοούμε ένα ταξινομημένο πίνακα τιμών οι οποίες βρίσκονται στα δεδομένα της στήλης, αλλά δεν βρίσκονται στο τρέχον λεξικό. Με τον όρο Diff Index αναφερόμαστε σε έναν πίνακα ακεραίων ο οποίος περιέχει τις θέσεις του τρέχοντος λεξικού στις οποίες οι τιμές αντικαθίστανται. Στην ουσία είναι ένα λεξικό διαφορών όπου γράφουμε ξεχωριστά τα κλειδιά και τις τιμές για να επιτύχουμε καλύτερη συμπίεση σε αυτά. Σημαντική λεπτομέρεια της υλοποίησης είναι ότι μια νέα τιμή μπορεί είτε απλά να προστίθεται στο λεξικό (το οποίο έχει μέγιστο μήκος 65536) είτε να αντικαθιστά κάποια παλιά. Η αντικατάσταση συμβαίνει όταν η προσθήκη θα οδηγούσε σε λεξικό με μήκος μεγαλύτερο του 65536 αλλά και όταν η προσθήκη θα σήμαινε χρησιμοποίηση πίνακα δεικτών των 2 bytes αντί του ενός. Πχ αν το τρέχον λεξικό έχει 127 τιμές θα προτιμηθεί να μην προστεθεί άλλη μια αλλά να αντικατασταθεί κάποια.

Το SPAC με αυτό τον τρόπο γράφει λιγότερα δεδομένα στο δίσκο οδηγώντας έτσι σε υψηλά ποσοστά συμπίεσης. Βασικός στόχος του SPAC είναι ότι σε ένα μεγάλο πίνακα που πιάνει πολλές σελίδες στο δίσκο τα δεδομένα γράφονται στις πρώτες σελίδες ενώ όλες οι υπόλοιπες αποτελούνται κυρίως από δείκτες καθώς τα λεξικά διαφορών είτε έχουν πολύ λίγες διαφορές είτε είναι άδεια (στην περίπτωση που το πλήθος των μεμονωμένων τιμών είναι < 65536)

Για τους αλγορίθμους σειριοποίησης και συμπίεσης ισχύει ότι και στο SdicC.

Βασικό μειονέκτημά της μεθόδου είναι ότι οι σελίδες που περιέχουν διαφορές δεν μπορούν να διαβαστούν χωρίς τις προηγούμενες. Γιαυτό σε περίπτωση ενός μεγάλου πίνακα είναι εφικτό ανά διαστήματα να γράφεται μια σελίδα με τα πλήρη λεξικά. Η τεχνική αυτή προσομοιώνει τα I και P frames στην συμπίεση video (MPEG-4 [29]).

Στο SPAC είναι επίσης εφικτές οι σαρώσεις αποσυμπιέζοντας μόνο τα τμήματα του αρχείου που περιέχουν τις κολώνες που σαρώνονται.

4.2.3 Correlation SPAC (cSPAC)

Όπως αναφέρθηκε παραπάνω το SPAC οδηγεί σε ένα αρχείο όπου οι πρώτες σελίδες περιέχουν δεδομένα και οι επόμενες κυρίως δείκτες. Το Correlation SPAC (cSPAC) αποτελεί μια επέκταση του SPAC που αποσκοπεί στην μείωση των δεικτών. Η γενική ιδέα είναι ότι μπορούμε να εκμεταλλευτούμε το γεγονός ότι κάποιες κολώνες σε έναν πίνακα μπορεί να συσχετίζονται με κάποιο τρόπο. Στην συνέχεια ορίζουμε τι εννοούμε με τον όρο συσχετιζόμενες κολώνες:

Συσχετιζόμενες κολώνες είναι 2 ή περισσότερες κολώνες όπου οι τιμές τους αλληλοεξαρτώνται σε κάποιο βαθμό. Δηλαδή η τιμή μιας εγγραφής σε μια κολώνα δίνει κάποια πληροφορία για την τιμή της εγγραφής σε μια άλλη κολώνα. Πλήρως συσχετιζόμενες κολώνες είναι οι κολώνες για τις οποίες υπάρχει 1-1 αντιστοίχιση των τιμών.

Αν υποθέσουμε τη βάση δεδομένων του υπουργείου Οικονομικών, παράδειγμα 2 πλήρως συσχετιζόμενων στηλών είναι ο αριθμός δελτίου ταυτότητας των πολιτών με το ΑΦΜ τους. Μερικώς συσχετιζόμενες κολώνες είναι το ονοματεπώνυμο των πολιτών με τον ΑΔΤ και το ΑΦΜ. Στην περίπτωση αυτή δεν υπάρχει απολύτως 1-1 αντιστοίχιση καθώς υπάρχουν και συνωνυμίες όμως μπορούμε να πούμε ότι υπάρχει συσχέτιση.

Σύμφωνα με το cSPAC αν έχουμε 2 ή περισσότερες πλήρως συσχετιζόμενες κολώνες μπορούμε να τις ενώσουμε σε μια χωρίς να αυξηθεί το μέγεθος των λεξικών δεδομένων και να αφαιρέσουμε τους πλεονάζοντες πίνακες δεικτών. Αν έχουμε κολώνες μερικώς συσχετιζόμενες μπορούμε να τις ενώνουμε αν η άυξηση του μεγέθους των λεξικών είναι μικρότερη από το μέγεθος του πίνακα δεικτών.

Ένα παράδειγμα ακολουθεί παρακάτω:

Έστω ότι έχουμε τον ακόλουθο πίνακα:

Πίνακας 4. Πίνακας συσχετιζόμενων πεδίων

ID	VAL1	VAL2
1	A	E

2	B	G
3	A	E
4	A	E
5	B	G
6	C	G
7	D	H
8	D	H
9	C	F
10	A	E

Στον παραπάνω πίνακα τα πεδία VAL1 και VAL2 είναι μερικώς συσχετιζόμενες. Το SPAC θα πραγματοποιήσει τα παρακάτω:

Καταρχήν, η κολώνα ID θα συμπιεστεί χωρίς λεξικό καθώς οι μεμονωμένες τιμές είναι πάνω από το 50% των συνολικών τιμών του πίνακα.

Οι τιμές της κολώνας VAL1 θα μπουν σε ταξινομημένο λεξικό ως εξής:

{A,B,C,D}

και ο πίνακας δεικτών (1 byte) θα είναι ο [0,1,0,0,1,2,3,3,2,0]

Ομοίως οι τιμές της κολώνας VAL2 θα μπουν σε ταξινομημένο λεξικό ως εξής:

{E,F,G,H}

και ο πίνακας δεικτών (1 byte) θα είναι ο [0,2,0,0,2,2,3,3,1,0]

Συνολικά θα έχουμε γράψει τις 8 τιμές και 20 bytes για τους πίνακες δεικτών στην πρώτη σελίδα.

Το cSPAC θα ενώσει τις κολώνες VAL1 και VAL2 σε ένα λεξικό :

{(A,E) , (B,G) , (C,G) ,(C,F), (D,H) }

και πλέον θα έχουμε ένα πίνακα δεικτών (1 byte) ο οποίος θα είναι ο [0,1,0,0,1,2,4,4,3,0]

Εδώ έχουμε γράψει στο λεξικό τις 8 τιμές και 2 επιπλέον επαναλήψεις τιμών (καθώς οι κολώνες δεν είναι πλήρως συσχετιζόμενες) έχοντας μειώσει στο μισό το τα bytes των πινάκων δεικτών.

Αν υποθέσουμε ότι έχουμε ένα πίνακα με πολλές σελίδες όπου κάποια στιγμή το τρέχον λεξικό θα περιέχει όλες τις τιμές της κολώνας με ελάχιστες ή χωρίς καθόλου αντικαταστάσεις θα γράφουμε λιγότερους δείκτες στον δίσκο επιτυγχάνοντας ένα πολύ μικρότερο αρχείο.

Στο παράδειγμα αυτό υπάρχει όφελος από την πρώτη σελίδα αν το άθροισμα του μεγέθους των C,G είναι μικρότερο από 10 bytes. Είναι προφανές ότι αν οι κολώνες ήταν πλήρως συσχετιζόμενες δεν θα γράφαμε περισσότερα δεδομένα στο δίσκο οπότε θα είχαμε μόνο το όφελος από την αφαίρεση του ενός πίνακα δεικτών.

Εύρεση συσχετιζόμενων πεδίων

Πολύ σημαντικός για το cSPAC είναι ο αλγόριθμος εύρεσης των συσχετιζόμενων στηλών. Εφόσον οι κολώνες μπορούν να συσχετίζονται ανα 2, 3 ή περισσότερες η εύρεση όλων των πιθανών διαμερίσεων στηλών είναι στην ουσία το πρόβλημα διαμέρισης συνόλου (set partitioning problem) [30].

Δοθέντος ενός συνόλου U , n στοιχείων, μιας συλλογής υποσυνόλων του U , $S = \{S_1, S_2, \dots, S_k\}$ και μιας συνάρτησης κόστους $c : S \rightarrow \mathbb{Q}^+$, να βρεθεί μια ελάχιστου κόστους συλλογή $S \subseteq S$ στοιχείων ξένων μεταξύ τους, που καλύπτει όλα τα στοιχεία του U (δηλαδή $U = \bigcup_{S_i \in S} S_i$ και κάθε στοιχείο του U να ανήκει σε ένα ακριβώς σύνολο της S).

Το πρόβλημα είναι NP πλήρες και στη συνέχεια ακολουθεί ένας πίνακας που δείχνει πόσες είναι οι πιθανές διαμερίσεις ανάλογα με το πλήθος των πεδίων.

Πίνακας 5. Πλήθος διαμερίσεων

Πεδία	Διαμερίσεις
1	1
2	2
3	5
4	15
5	52
6	203
7	877
8	4140
9	21147
10	115975
15	1382958545
20	51724158235372

Η εύρεση ενός άπληστου αλγόριθμου που να βρίσκει συσχετιζόμενες κολώνες με χαμηλή πολυπλοκότητα αφήνεται ως μελλοντική δουλειά, ενώ στην παρούσα φάση το cSPAC

ψάχνει στην πρώτη σελίδα δεδομένων, με εξαντλητική αναζήτηση πιθανές συσχετίσεις στηλών ανά 2.

Η επιλογή 2 συσχετιζόμενων στηλών από το cSPAC γίνεται ως ακολούθως:

Θεωρούμε A και B τα σύνολα των τιμών των 2 στηλών και C το σύνολο των αντιστοιχίσεων μεταξύ τους. Με τον όρο αντιστοιχήσεις εννοούμε ότι αντιστοιχίζουμε μια τιμή από το A σε μια τιμή του B όταν στον πίνακα βρίσκονται οι τιμές αυτές στις 2 κολώνες στην ίδια σειρά.

Στο cSPAC για να μπορούν να θεωρηθούν δύο κολώνες συσχετιζόμενες αναγκαίες αλλά όχι ικανές συνθήκες είναι να ισχύουν:

- Αν $|A| < 128$ και $|B| < 128$ τότε $|C| < 128$
- Αν $127 < |B| < 65536$ ή $127 < |B| < 65536$ τότε $|C| < 65536$

Η πρώτη σχέση είναι απαραίτητη καθώς αν $|C| > 128$ τότε αφαιρώντας τον έναν από τους δύο πίνακες δεικτών του 1 byte, θα πρέπει να τους αντικαταστήσουμε με πίνακα των 2 bytes, και δεν θα έχουμε κάποιο όφελος. Παράλληλα αν η δεύτερη σχέση δεν ισχύει δεν θα φτάνουν στην άλλη περίπτωση ούτε πίνακες των 2 bytes.

Μια επιπλέον ευριστική συνάρτηση που χρησιμοποιούμε είναι να ισχύει:

$$|C| < |A| + |B|$$

καθώς σε αντίθετη περίπτωση θα πρέπει να επαναλάβουμε πολλές τιμές στο νέο λεξικό που θα δημιουργηθεί γράφοντας τουλάχιστον δύο φορές τις τιμές του A ή του B . Για όσα ζεύγη στηλών ισχύει η παραπάνω σχέση επιλέγουμε τα ζεύγη με ελάχιστο μέγεθος του C .

Η ιδανική περίπτωση είναι όταν $|C| = |A| = |B|$

όπου η αντιστοίχιση είναι 1-1 και άρα δεν χρειάζεται να υπάρχουν επαναλήψεις δεδομένων στα λεξικά.

5. ΛΕΠΤΟΜΕΡΕΙΕΣ ΥΛΟΠΟΙΗΣΗΣ

Για την υλοποίηση και τεκμηρίωση των παραπάνω αλγορίθμων έχει χρησιμοποιηθεί το EXAREME [31]. Το EXAREME είναι ένα σύστημα που χρησιμοποιείται για την επεξεργασία μεγάλων δεδομένων στο νέφος. Το σύστημα προσφέρει μια δηλωτική γλώσσα επερωτήσεων βασισμένη στην SQL η οποία είναι εμπλουτισμένη με συναρτήσεις ορισμένες από τον χρήστη. Το EXAREME εκμεταλλεύεται την ελαστικότητα στο νέφος ώστε να προσφέρει tradeoffs ανάμεσα στους χρόνους εκτέλεσης και στο κόστος χρήσης των διαθέσιμων πόρων.

Το EXAREME δεν θα μπορούσε να λειτουργήσει χωρίς το madis[32] το οποίο παρέχει δυνατότητες επεξεργασίας δεδομένων.

Πιο αναλυτικά το madis είναι ένα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων χτισμένο στην κορυφή της SQLite. Το σύστημα είναι υλοποιημένο σε Python και παρέχει δυνατότητες προσθήκης επεκτάσεων στην SQLite με την μορφή τελεστών ορισμένων από τον χρήστη.

Κύριος στόχος του madis είναι να παρέχει μια εκτεταμένη λίστα από εργαλεία για την επεξεργασία και ανάλυση δεδομένων ενώ μπορεί να διαχειριστεί δεκάδες εκατομμύρια πλειάδες σε ένα pc.

Οι τελεστές που μπορούν να υλοποιηθούν στο madis χωρίζονται σε τρεις κατηγορίες:

Row

Οι row τελεστές παίρνουν σαν είσοδο μια ή περισσότερες κολώνες και παράγουν μια τιμή για κάθε τιμή που παίρνουν στην είσοδο (πχ. συνάρτηση UPPER()). Είναι αντίστοιχοι των Map τελεστών στα συστήματα Map/Reduce.

Aggregate

Οι τελεστές αυτοί είναι αντίστοιχοι των Reduce τελεστών στα Map/Reduce συστήματα και παρέχουν aggregate συναρτήσεις όπως η AVG().

Virtual table

Χρησιμοποιούνται για την δημιουργία εικονικών πινάκων στη βάση οι οποίοι να μπορούν να χρησιμοποιηθούν με όμοιο τρόπο με τους υλοποιημένους πίνακες στο δίσκο.

Οι οριζόμενες από το χρήστη συναρτήσεις εκτελούνται πάνω στα δεδομένα της βάσης με αποτέλεσμα να περιορίζεται το κόστος μετακίνησης των δεδομένων ανάμεσα στα 2 επίπεδα εκτέλεσης (λειτουργικό και σχεσιακό)

Οι αλγόριθμοι συμπίεσης που περιγράφηκαν στο προηγούμενο κεφάλαιο υλοποιήθηκαν σαν virtual table τελεστές στο madis και στόχος τους είναι να ελαχιστοποιήσουν τον χρόνο και το κόστος μετακινήσεων δεδομένων στο δίκτυο που χρησιμοποιεί το EXAREME.

6. ΠΕΙΡΑΜΑΤΑ

Παράμετροι πειραμάτων

Για τα πειράματα που παρουσιάζουμε σε αυτό το κεφάλαιο χρησιμοποιήθηκαν οι ακόλουθοι σχεσιακοί πίνακες:

1. lineitem (TPC-H [33])
2. fact table (BMRB , πραγματικά βιολογικά δεδομένα)

Τα στοιχεία των πινάκων φαίνονται παρακάτω :

Πίνακας 6. Παράμετροι πειραμάτων

Table	row number	column number	size on disc (SQLite file)
Lineitem	10000000	17	1.3GB
fact table	6497673	36	554MB

Στη συνέχεια το πρώτο πείραμα αφορά την σύγκριση των γενικών αλγορίθμων συμπίεσης στα σχεσιακά δεδομένα. Το επόμενο πείραμα συγκρίνουμε στα δεδομένα μας τους αλγορίθμους σειριοποίησης marshal και msgpack. Στο τρίτο πείραμα εκτελούμε τους πλήρης αλγορίθμους συμπίεσης πάνω στα δεδομένα και παρουσιάζουμε τα αποτελέσματα. Στο τέταρτο πείραμα συγκρίνουμε τις πιθανές συσχετίσεις 8 πεδίων για ένα τμήμα του TPC-H (μεγέθους 2 σελίδων) , ενώ στο τελευταίο πείραμα συγκρίνουμε τις πιθανές συσχετίσεις 8 πεδίων σε όλο τον πίνακα.

Πείραμα 1. Σύγκριση γενικών αλγορίθμων συμπίεσης

Το πρώτο πείραμα αφορά την σύγκριση γενικών αλγορίθμων συμπίεσης. Για τις ανάγκες του πειράματος χρησιμοποιήσαμε τον πίνακα lineitem του TPC-H . Τα εργαλεία συμπίεσης που χρησιμοποιήσαμε ανήκουν σε διαφορετικές κατηγορίες αλγορίθμων. Συγκεκριμένα χρησιμοποιήθηκε ο Bzip2 που υλοποιεί BWT, ο Gzip που χρησιμοποιεί κώδικες huffman, και ο lzma που χρησιμοποιεί αριθμητική κωδικοποίηση, λεξικά και αλυσίδες markov.

Τρέξαμε τους αλγορίθμους συμπίεσης σε κολώνες που περιέχουν συμβολοσειρές, και σε κολώνες με αριθμητικές τιμές.

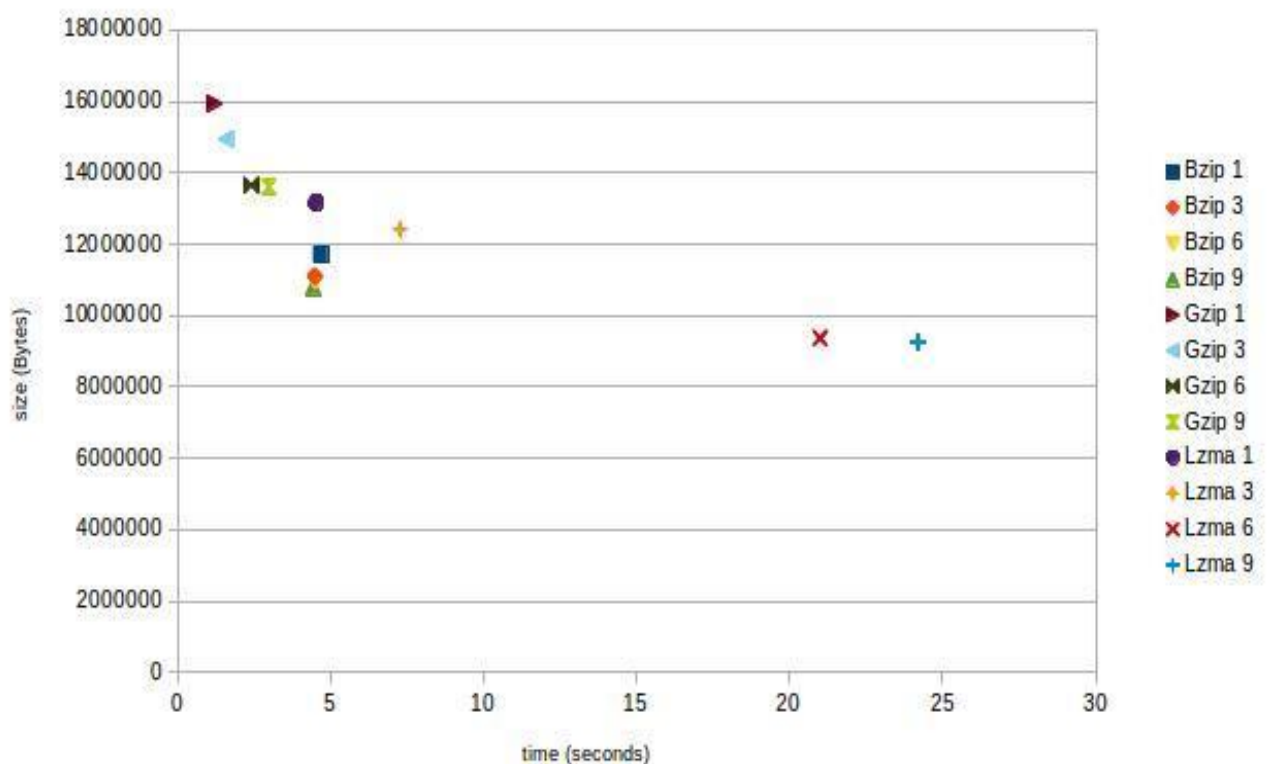
Στον παρακάτω πίνακα φαίνεται η σύγκριση των αλγορίθμων συμπιέζοντας την κολώνα `l_comment` (συμβολοσειρές)

Πίνακας 7. Σύγκριση αλγορίθμων γενικής συμπίεσης σε πεδία συμβολοσειρών

Compression algorithm	Compression Level	Compressed Size (Bytes)	Compression Time (seconds)	Decompress Time (seconds)	Total time
Bzip2	1	11699470	3.349	1.331	4.680
Bzip2	3	11099740	3.057	1.427	4.484
Bzip2	6	10981259	3.169	1.316	4.485
Bzip2	9	10786766	3.128	1.305	4.433
Gzip	1	15937256	0.764	0.436	1.200
Gzip	3	14942968	0.881	0.699	1.580
Gzip	6	13652219	1.857	0.578	2.435
Gzip	9	13593733	2.557	0.419	2.976
Lzma	1	13159272	3.573	0.951	4.524
Lzma	3	12413661	6.400	0.867	7.267

Lzma	6	9363282	20.23	0.761	20.991
Lzma	9	9269256	23.441	0.752	24.193

Παρακάτω βλέπουμε ένα διάγραμμα που απεικονίζει τα δεδομένα του παραπάνω πίνακα:



Σχήμα 11. Διάγραμμα σύγκρισης αλγορίθμων γενικής συμπίεσης σε πεδία συμβολοσειρών

Όπως φαίνεται στο διάγραμμα καλύτερη απόδοση χρόνου και βαθμού συμπίεσης επιτυγχάνουν οι αλγόριθμοι gzip με επίπεδο συμπίεσης 1,3 και 6 ο bzip με επίπεδο συμπίεσης 9 και ο lzma με επίπεδο 6 και 9 όπου επιτυγχάνεται η μέγιστη συμπίεση, με μικρή όμως διαφορά από τον bzip με επίπεδο 9, αλλά συγχρόνως με μεγάλη χρονική καθυστέρηση.

Γενικά φαίνεται ότι στις κολώνες με συμβολοσειρές ο gzip αλγόριθμος είναι καλύτερος όταν θέλουμε να συμπίεσουμε γρήγορα μια κολώνα ενώ ο bzip με επίπεδο πετυχαίνει μεγαλύτερα ποσοστά συμπίεσης όταν αυτό που μας ενδιαφέρει είναι κυρίως η συμπίεση.

Στη συνέχεια επαναλάβαμε τα ίδια πειράματα σε πεδίο με αριθμητικές τιμές (ακεραίους και δεκαδικούς)

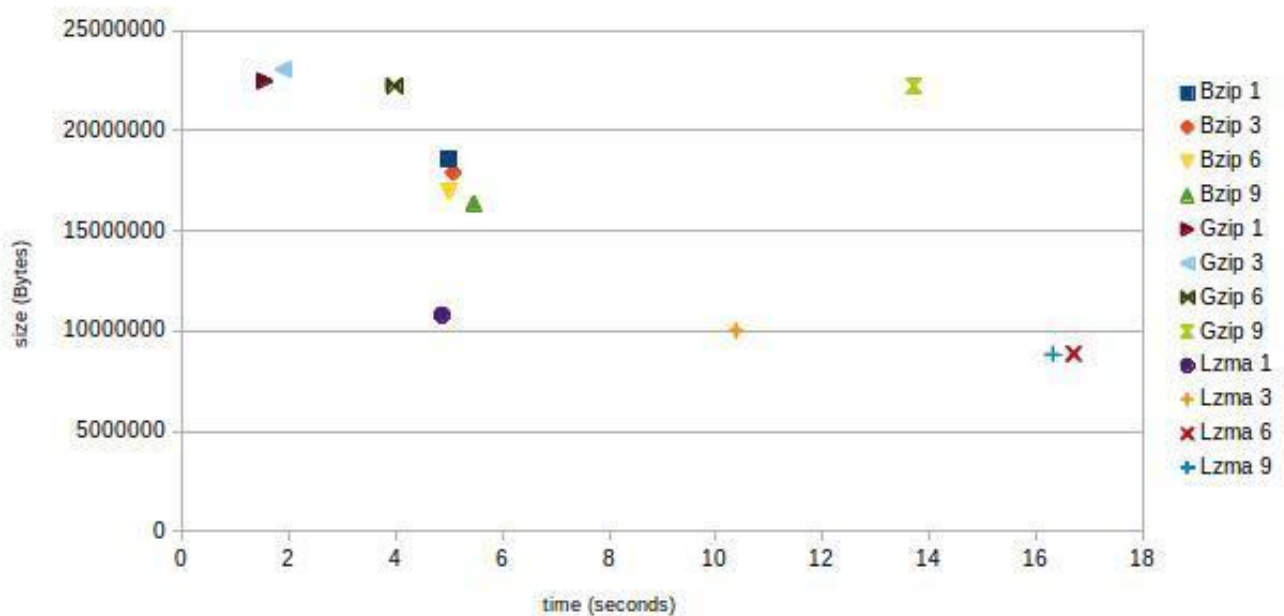
Στον παρακάτω πίνακα φαίνεται η σύγκριση των αλγορίθμων συμπιέζοντας την κολώνα l_extendedprice.

Πίνακας 8. Σύγκριση αλγορίθμων γενικής συμπίεσης σε πεδία αριθμητικών τιμών

Compression algorithm	Compression Level	Compressed Size (Bytes)	Compression Time (seconds)	Decompress Time (seconds)	Total time
Bzip2	1	18574016	3.487	1.505	4.992
Bzip2	3	17905995	3.450	1.632	5.082
Bzip2	6	16936937	3.479	1.520	4.999
Bzip2	9	16369981	3.976	1.489	5.465
Gzip	1	22483594	1.025	0.526	1.551
Gzip	3	23058066	1.325	0.568	1.893
Gzip	6	22229243	3.544	0.422	3.966
Gzip	9	22224386	12.935	0.786	13.721
Lzma	1	10786844	3.937	0.931	4.868

Lzma	3	10006865	9.505	0.884	10.389
Lzma	6	8861583	15.761	0.962	16.723
Lzma	9	8820000	15.423	0.909	16.332

Ακολουθεί το διάγραμμα που προκύπτει από τον πίνακα :



Σχήμα 12. Διάγραμμα σύγκρισης αλγορίθμων γενικής συμπίεσης σε πεδία αριθμητικών τιμών

Όπως φαίνεται για αριθμητικές τιμές τα αποτελέσματα διαφέρουν καθώς φαίνεται ότι ο Bzip2 εδώ δεν είναι αποδοτικός ενώ τη θέση του έχει πάρει ο Lzma με επίπεδο συμπίεσης 1. Συνοπτικά φαίνονται πιο αποδοτικοί ο gzip με επίπεδο συμπίεσης 1 ο Lzma με επίπεδο συμπίεσης 1 και ο Lzma με μεγαλύτερα επίπεδα συμπίεσης ο οποίος όμως απαιτεί πολύ περισσότερο χρόνο χωρίς αντίστοιχα σημαντικά μεγαλύτερη συμπίεση.

Συνολικά φαίνεται ότι στις κολώνες με συμβολοσειρές μπορούμε να επιλέγουμε ανάμεσα στους gzip, bzip δίνοντας έμφαση ανάμεσα στην ταχύτητα και τον βαθμό συμπίεσης, ενώ στις κολώνες με αριθμητικές τιμές είναι προτιμότερο η επιλογή να γίνεται ανάμεσα σε lzma και gzip.

Πείραμα 2. Σύγκριση αλγορίθμων σειριοποίησης

Το επόμενο πείραμα έγινε για να συγκριθούν στα σχεσιακά δεδομένα οι αλγόριθμοι σειριοποίησης marshal και msgpack που αναφέρθηκαν σε προηγούμενη ενότητα. Τα πειράματα που παρουσιάστηκαν επιβεβαιώνεται και στα σχεσιακά δεδομένα. Χρησιμοποιήθηκε η βάση του TPC-H και η σύγκριση έτρεξε για τον αλγόριθμο SdicC με και χωρίς συμπίεση με gzip. Το μέγεθος της βάσης που χρησιμοποιήθηκε είναι 133MB και αποτελείται από 17 κολώνες και 1000000 σειρές.

Τα αποτελέσματα φαίνονται στους παρακάτω πίνακες:

Πίνακας 9. Σύγκριση αλγορίθμων σειριοποίησης (1)

Serialization	Size without gzip (MB)	Decoding Time (seconds)	Encoding Time (seconds)
Marshal	69.38	5.600	9.650
Msgpack	63.77	5.200	10.150

Πίνακας 10. Σύγκριση αλγορίθμων σειριοποίησης (2)

Serialization	Size with gzip (MB)	Decoding Time (seconds)	Encoding Time (seconds)
Marshal	29.77	5.800	10.900

Msgpack	28.68	5.400	11.100
---------	-------	-------	--------

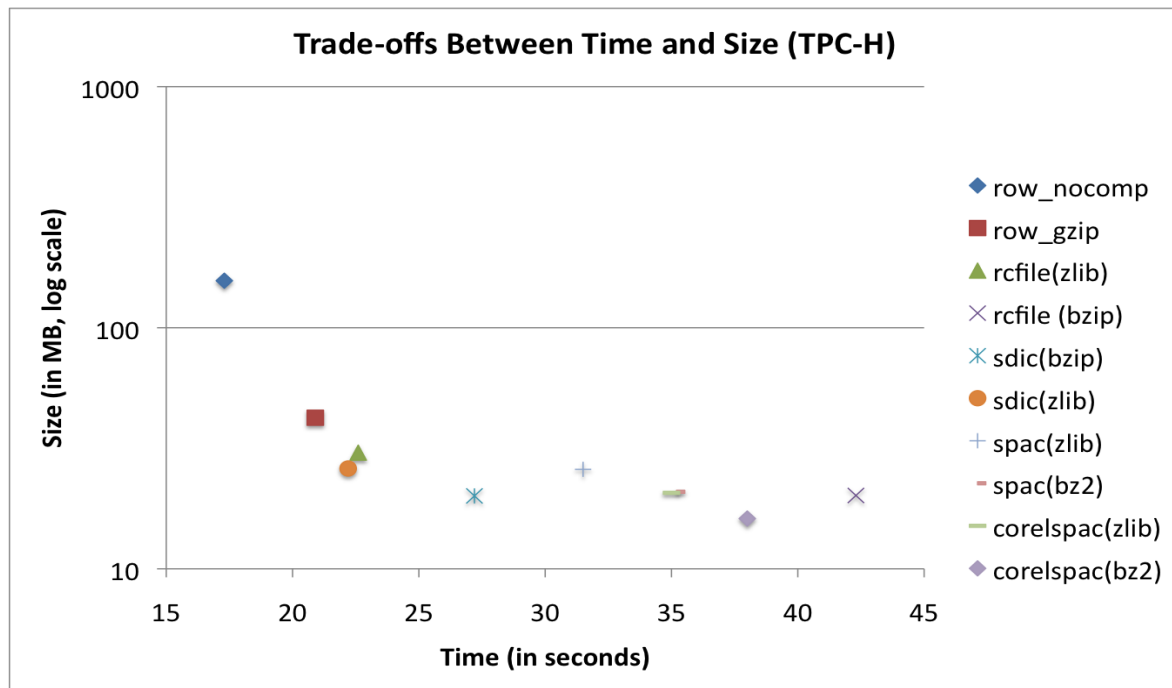
Οι χρόνοι αποκωδικοποίησης που εμφανίζονται παραπάνω περιλαμβάνουν και τον χρόνο εισαγωγής των πλειάδων εκ νέου σε πίνακα βάσης.

Το συμπέρασμα είναι ότι αν δεν χρησιμοποιείται συμπίεση τα αποτελέσματα συμφωνούν με τα πειράματα που παρουσιάστηκαν σε προηγούμενη ενότητα με το msgpack να απαιτεί μεγαλύτερο χρόνο για την κωδικοποίηση και λιγότερο χρόνο για την αποκωδικοποίηση.

Αυτό που φαίνεται από το πείραμα είναι ότι στη συμπίεση με gzip ο χρόνος συμπίεσης είναι μικρότερος όταν χρησιμοποιούμε msgpack καθώς το msgpack παράγει μικρότερο όγκο δεδομένων με αποτέλεσμα η απόσταση σε χρόνο των ανάμεσα στις 2 επιλογές να μειώνεται.

Πείραμα 3. Αλγόριθμοι συμπίεσης σχεσιακών δεδομένων

Στο επόμενο πείραμα γίνεται μια σύγκριση των αλγορίθμων συμπίεσης σχεσιακών δεδομένων για την βάση δεδομένων TPC-H και για το fact table του BMRB (βιολογικά δεδομένα)



Σχήμα 13. Διάγραμμα σύγκρισης αλγορίθμων συμπίεσης σχεσιακών δεδομένων (TPC-H)

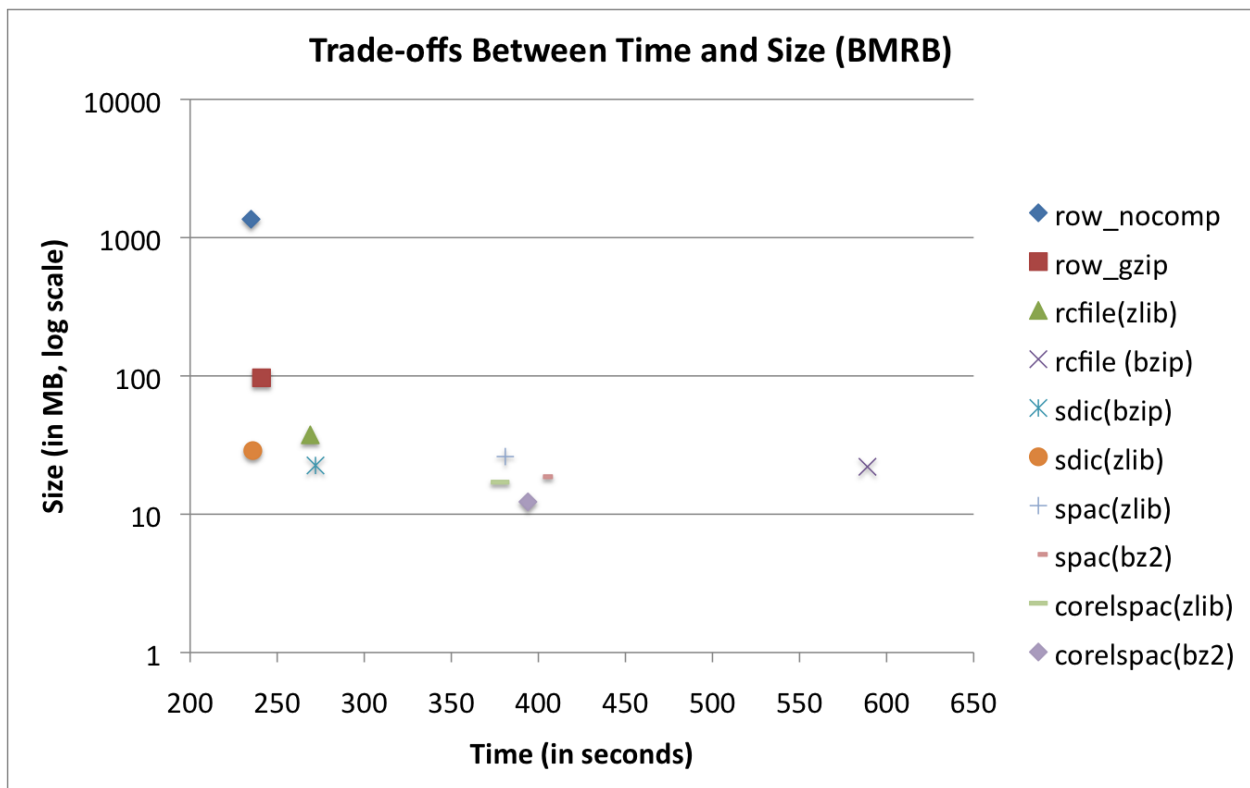
Οι αλγόριθμοι γενικής συμπίεσης που έχουν χρησιμοποιηθεί στα παραπάνω πειράματα είναι ο zlib και ο bz2 με επίπεδο συμπίεσης 5.

Από τα πειράματα για το TPC-H φαίνεται ότι ο SdicC με zlib βγάζει εκτός pareto τον rcfile αλγόριθμο για το TPC-H. Άλλοι αλγόριθμοι που επιτυγχάνουν μεγαλύτερη συμπίεση και βρίσκονται στο pareto frontier είναι οι SdicC με bz2 και ο cSPAC με bz2.

Ο βαθμός συμπίεσης στους αλγόριθμους που έχουμε υλοποιήσει είναι από 7x έως 10x.

Στο πείραμα για το TPC-H φαίνεται επίσης ότι σε δίκτυα γρήγορα (όπου δεν μας απασχολεί ιδιαίτερα ο βαθμός συμπίεσης αλλά η ταχύτητα) συμφέρει συμπίεση στις σειρές ή και καθόλου συμπίεση.

Η γραφική για το δεύτερο πείραμα που έγινε στην βάση BMRB ακολουθεί παρακάτω:



Σχήμα 14. Διάγραμμα σύγκρισης αλγορίθμων συμπίεσης σχεσιακών δεδομένων (BMRB)

Η εικόνα διαφέρει στο ότι εδώ η επιλογή του ασυμπίεστου πίνακα δεν είναι στο pareto frontier ενώ βρίσκονται σε αυτό οι αλγόριθμοι SdicC και cSPAC με zlib και bzip2.

Η διαφορές στα αποτελέσματα των πειραμάτων οφείλονται στη φύση των δεδομένων.

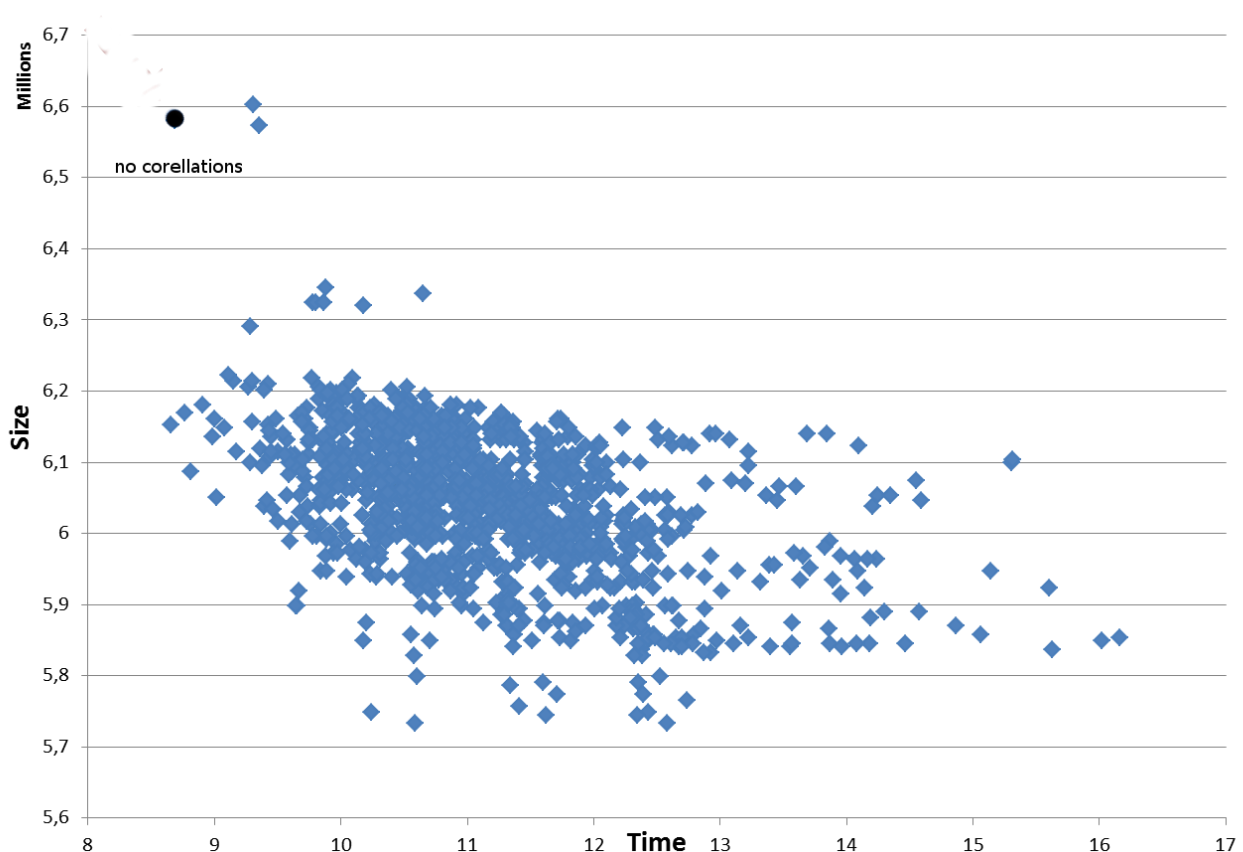
Στο TPC-H υπάρχουν κολώνες με μεγάλες τυχαίες συμβολοσειρές οι οποίες δεν επαναλαμβάνονται και έτσι είναι πιο ακριβές στη συμπίεση χωρίς να μπορούν να συμπεστούν σε μεγάλο βαθμό. Στο BMRB δεν ισχύει αυτό καθώς οι περισσότερες κολώνες είναι με τιμές που επαναλαμβάνονται αρκετές φορές ενώ δεν υπάρχει κάποια κολώνα που να περιέχει συμβολοσειρές. Οι τιμές του πίνακα αυτού είναι είτε απλοί χαρακτήρες είτε αριθμητικές τιμές. Αυτά τα δεδομένα ευνοούν την υψηλή συμπίεση η οποία ξεπερνάει και το 50x.

Και στις 2 όμως περιπτώσεις ότι οι αλγόριθμοι που έχουν υλοποιηθεί βρίσκονται στο pareto frontier και δίνουν διαφορετικές λύσεις για έμφαση στο βαθμό συμπίεσης ή την ταχύτητα.

Πείραμα 4. Συσχετισμοί πεδίων (μικροί πίνακες - 2 σελίδες στο δίσκο)

Το πείραμα αυτό εφαρμόστηκε σε ένα τμήμα του πίνακα lineitem του TPC-H και με αυτό εξετάστηκαν όλοι οι πιθανοί συνδυασμοί από κολώνες (cSPAC). Επιλέχθηκαν 8 πεδία από τον πίνακα lineitem από τα οποία και δημιουργήθηκε ένας νέος πίνακας. Σε αυτόν τον πίνακα εκτελέστηκε ο αλγόριθμος cSPAC για όλες τις πιθανές διαμερίσεις στηλών (4140 περιπτώσεις). Το μέγεθος του αρχείου SQLite που περιείχε τον πίνακα ήταν 70.67MB έτσι ώστε να χρειάζονται 2 σελίδες του cSPAC αρχείου για την αποθήκευση.

Τα αποτελέσματα παρουσιάζονται παρακάτω:



Σχήμα 15. Διάγραμμα σύγκρισης συσχετίσεων πεδίων σε αρχείο SPAC 2 σελίδων

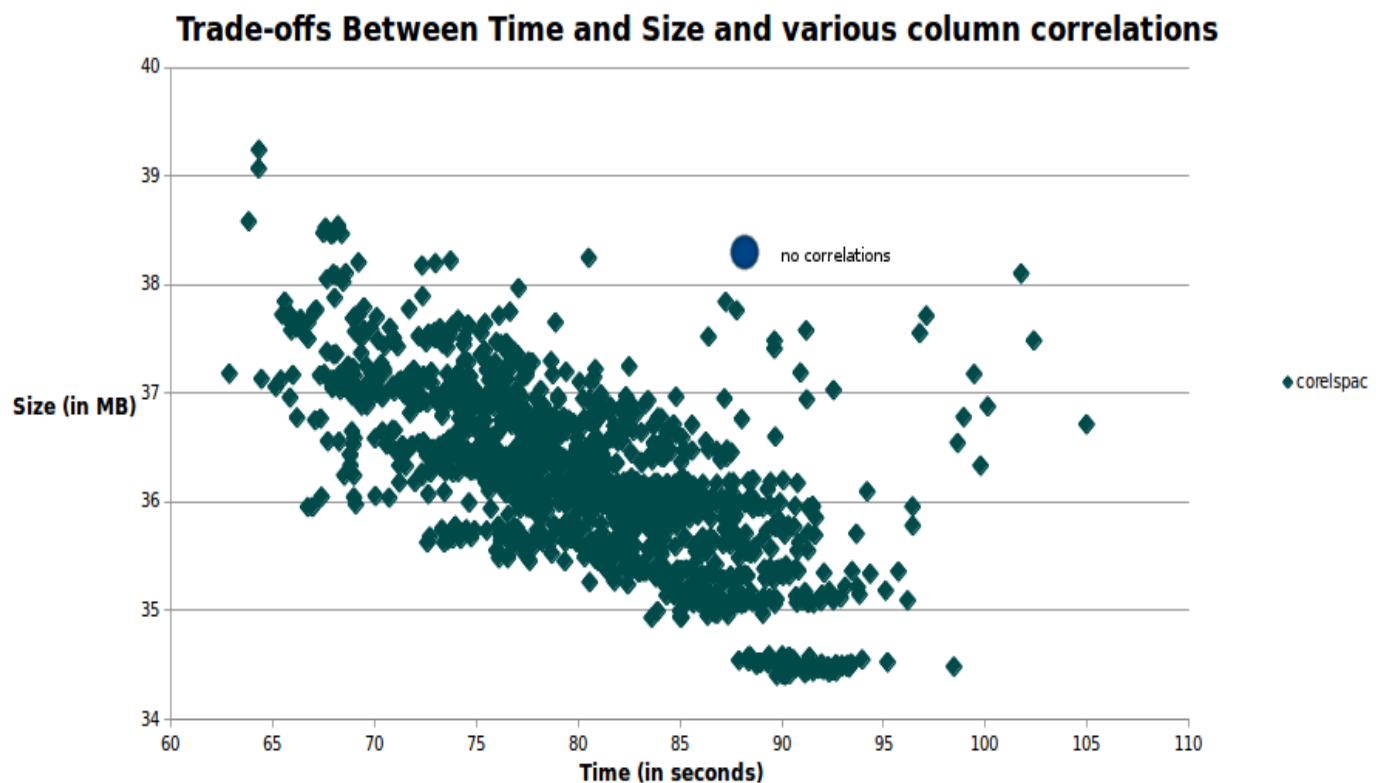
Από την παραπάνω γραφική βλέπουμε ότι σε μικρά αρχεία η συσχέτιση πεδίων βελτιώνει τον βαθμό συμπίεσης, αυξάνοντας όμως παράλληλα τον χρόνο εκτέλεσης. Αυτό συμβαίνει λόγω του παραπάνω κόστους υπολογισμού των συσχετιζόμενων πεδίων που γίνεται στην

πρώτη σελίδα του SPAC αρχείου, και των προσθέσεων τιμών στα λεξικά που είναι πιο συχνή στις πρώτες σελίδες.

Πείραμα 5. Συσχετισμοί πεδίων (μεγάλοι πίνακες)

Το τελευταίο πείραμα εφαρμόστηκε και πάλι στον πίνακα lineitem του TPC-H, στα ίδια 8 πεδία αλλά σε περισσότερες πλειάδες ούτως ώστε το αρχικό μέγεθος του πίνακα να είναι 430.3 MB

Τα αποτελέσματα παρουσιάζονται στον παρακάτω πίνακα :



Σχήμα 16. Διάγραμμα σύγκρισης συσχετίσεων πεδίων σε πίνακες πολλών σελίδων

Από τον παραπάνω πίνακα προκύπτει ότι υπάρχουν στήλες που η συσχέτισή τους μπορεί να δώσει μεγαλύτερα ποσοστά συμπίεσης (έως 11% επιπλέον συμπίεση στο συγκεκριμένο

πείραμα) αλλά και πολύ μεγαλύτερες ταχύτητες (έως 30% γρηγορότερη συμπίεση) σε σχέση με την συμπίεση του απλού αλγόριθμου (SPAC).

Το συμπέρασμα που προκύπτει από τα 2 τελευταία πειράματα είναι ότι το cSPAC μπορεί να είναι πιο αποδοτικό (σε συμπίεση και χρόνο) όσο μεγαλύτερος είναι ο πίνακας που συμπίεζεται. Αυτό ισχύει αφενός γιατί το SPAC κερδίζει σε συμπίεση από το γεγονός ότι γράφει λεξικά διαφορών. Στο πρώτο πείραμα ο πίνακας απαιτεί 2 σελίδες του αρχείου SPAC με αποτέλεσμα η μια σελίδα να είναι με πλήρη λεξικά και η επόμενη με λεξικά διαφορών, ενώ στο δεύτερο πείραμα έχουμε περισσότερες σελίδες με λεξικά διαφορών. Παράλληλα το cSPAC σκοπεύει να κερδίζει σε ταχύτητα και βαθμό συμπίεσης από τους πίνακες δεικτών που αφαιρεί. Σε ένα μικρό πίνακα, στις πρώτες σελίδες οι αρκετές προσθαφαιρέσεις τιμών στα λεξικά, καθώς και ο υπολογισμός των συσχετιζόμενων πεδίων προσθέτει πολυπλοκότητα, ενώ σε μεγάλους πίνακες από ένα σημείο και μετά τα λεξικά δεν τροποποιούνται πολύ ενώ από άποψη πολυπλοκότητας και συμπίεσης αποφεύγεται ο υπολογισμός και η αποθήκευση n πινάκων δεικτών για κάθε $n+1$ συσχετιζόμενα πεδία.

Η εύρεση ενός απληστου αλγόριθμου για τον υπολογισμό των πιο αποδοτικών διαμερίσεων ως προς συμπίεση/ταχύτητα αφήνεται ως μελλοντική δουλειά.

7. ΠΡΟΤΑΣΕΙΣ ΓΙΑ ΤΟ ΜΕΛΛΟΝ

1. Εύρεση άπληστου αλγορίθμου για την επιλογή των καλύτερων διαμερίσεων πεδίων στον αλγόριθμο cSPAC. Η εύρεση κατάλληλων διαμερίσεων μπορεί να οδηγήσει σε καλύτερη συμπίεση αλλά και σε πολύ καλύτερους χρόνους εκτέλεσης όπως είδαμε στα πειράματα.
2. Συσχέτιση στηλών που αναπαριστώνται με λεξικά και δείκτες, με στήλες που αναπαριστώνται με όλες τις τιμές και τα διπλότυπα. Οι στήλες αυτές όπως είδαμε σε προηγούμενο κεφάλαιο είναι στήλες με λίγα ή καθόλου διπλότυπα (κλειδιά) , και συμπίεζονται αταξινόμητες. Η ιδέα είναι ότι αν μπορούμε να τις ταξινομήσουμε πριν συμπειστούν αυτό θα δώσει υψηλότερα ποσοστά συμπίεσης. Εάν όμως ταξινομηθούν μοναδικά θα πρέπει να διατηρείται και ένας πίνακας δεικτών οπότε το όφελος της συμπίεσης εξαλείφεται. Εάν μπορούν να ενωθούν με στήλες με ελάχιστες μεμονωμένες τιμές τότε μπορούν να ταξινομηθούν και οι δύο ως προς την μεγαλύτερη κολώνα να χρησιμοποιηθεί ένας πίνακας δεικτών και να συμπειστούν πολύ καλύτερα.
3. Εύρεση και ένωση συσχετιζόμενων λεξικών που περιέχουν τιμές από διαφορετικά πεδία (SdicC , SPAC). Είναι δυνατόν να υπάρχουν διαφορετικά πεδία σε έναν πίνακα με κοινές τιμές. Η ένωσή τους θα μειώσει τον αριθμό διπλότυπων τιμών από διαφορετικά πεδία οδηγώντας σε καλύτερη συμπίεση.
4. Υλοποίηση δυναμικού αλγορίθμου που να επιλέγει σε κάθε μπλοκ τον καταλληλότερο αλγόριθμο συμπίεσης (δίνοντας έμφαση σε ταχύτητα ή συμπίεση) ανάλογα με την κατάσταση του δικτύου την δεδομένη στιγμή.
5. Εφαρμογή του μετασχηματισμού MTF (Move-to-front transform) [34] στους δείκτες (SdicC , SPAC, cSPAC) για επίτευξη μεγαλύτερων ποσοστών συμπίεσης στους δείκτες.
6. Έρευνα για κατάλληλη ταξινόμηση των δεδομένων. Η ταξινόμηση των δεδομένων παίζει πολύ σημαντικό ρόλο στην ταχύτητα και στο βαθμό συμπίεσης. Επίσης στα σχεσιακά δεδομένα η σειρά των πλειάδων δεν παίζει σημασία και αυτό μας δίνει την δυνατότητα να μπορούμε να την αλλάζουμε προς όφελος της συμπίεσης. Σε έναν

πίνακα όμως αν ταξινομείται ένα πεδίο επηρεάζονται και τα υπόλοιπα με βάση αυτό. Σημαντική είναι λοιπόν η εύρεση ενός άπληστου αλγορίθμου που να υπολογίζει την κατάλληλη σειρά με την οποία πρέπει να ταξινομούνται τα πεδία ώστε να επιτυγχάνεται η βέλτιστη συμπεριφορά.

7. Συμπίεση με απώλειες σε πεδία επιλεγμένα από το χρήστη. Σε πολλές περιπτώσεις μικροδιαφορές που υπάρχουν ανάμεσα σε τιμές δεν μας ενδιαφέρουν, πχ σε ένα πεδίο που περιέχει θερμοκρασίες. Ίσως η θερμοκρασία 19.778844 °C να μπορεί να θεωρηθεί ίσοδύναμη με την 19.778845 °C σε κάποιες περιπτώσεις. Η εξομάλυνση αυτών των διαφορών μπορεί να δώσει υψηλές δυνατότητες συμπίεσης στους υπάρχοντες αλγορίθμους.

8. ΣΥΜΠΕΡΑΣΜΑΤΑ

Τα συμπεράσματα που προέκυψαν από αυτή την διπλωματική εργασία συνοψίζονται παρακάτω :

- Η αύξηση του όγκου των δεδομένων απαιτεί καινοτόμες μεθόδους για την αποτελεσματική διαχείρησή τους.
- Στα συστήματα κατανεμημένης επεξεργασίας σημαντικό ρόλο παίζει το κόστος μεταφοράς δεδομένων.
- Η συμπίεση των δεδομένων είναι ένας τρόπος μείωσης του όγκου τους, ώστε να μειώνεται το κόστος μεταφοράς.
- Η συμπίεση σχεσιακών δεδομένων για ερωτήματα αναλυτικής επεξεργασίας είναι ένα σημαντικό θέμα στην επιστημονική κοινότητα και αυτό αποδुकνείται από το πλήθος επιστημονικών δημοσιεύσεων στον χώρο αυτό.
- Στην παρούσα διπλωματική εργασία παρουσιάζονται αλγόριθμοι που μπορούν να οδηγήσουν σε υψηλούς βαθμούς συμπίεσης των σχεσιακών δεδομένων ώστε να βελτιώνεται το κόστος μεταφοράς στο δίκτυο.
- Θα πρέπει οι αλγόριθμοι συμπίεσης να είναι ελαστικοί ώστε να προσαρμόζουν το επίπεδο συμπίεσης και την ταχύτητα συμπίεσης/αποσυμπίεσης ανάλογα με την τρέχουσα κατάσταση του δικτύου και την ταχύτητα με την οποία μπορούν να μεταφέρονται δεδομένα μέσα από αυτό.
- Σημαντική είναι η διάταξη των δεδομένων καθώς θα πρέπει να δίνεται η δυνατότητα εκτέλεσης ερωτημάτων απευθείας στα συμπιεσμένα δεδομένα αποσυμπιέζοντας μόνο τα τμήματα που απαιτούνται ανάλογα με το ερώτημα.
- Υπάρχουν πολλές επιπλέον προοπτικές στο θέμα αυτό που παρουσιάζονται στην προηγούμενη ενότητα και αφήνονται ως προτάσεις για το μέλλον.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	Ελληνικός όρος
analytical processing	αναλυτική επεξεργασία
data transfer	μεταφορά δεδομένων
distributed systems	συστήματα κατανεμημένης επεξεργασίας
data layout	διάταξη δεδομένων
Stream	Ροή
Input	Είσοδος
Output	Έξοδος
row store	αποθήκευση ανά σειρά
column store	αποθήκευση ανά στήλη
global dictionary	καθολικό λεξικό
local dictionary	τοπικό λεξικό
Serialization	Σειριοποίηση

ΣΥΝΤΜΗΣΕΙΣ - ΑΡΚΤΙΚΟΛΕΞΑ - ΑΚΡΩΝΥΜΙΑ

OLAP	Online analytical processing
PAX	Partition Attributes Across
BWT	Burrows-Wheeler Transform
LZ	Lempel-Ziv
SdicC	Sorted Dictionary per Column
SPAC	Structured partition attribute compression
cSPAC	correlation SPAC
MTF	Move to front

ΑΝΑΦΟΡΕΣ

- [1] <http://cacs.usc.edu/education/cs653/Simon-Exascale-LBL13.pdf>
- [2] <http://www.postgresql.org/docs>
- [3] <http://www.sqlite.org/docs.html>
- [4] <http://dev.mysql.com/doc>
- [5] <https://www.monetdb.org/Documentation>
- [6] DB2 V3 Performance Topics, GG24-4284-00,
<http://www.redbooks.ibm.com/redbooks/pdfs/gg244284.pdf>
- [7] H. Kimura, V. Narasayya, and M. Syamala. Compression aware physical database design. PVLDB, 4(10):657–668, 2011
- [8] M. Poss and D. Potapov. Data Compression in Oracle. In Proc. VLDB, Berlin, Germany, 2003.
- [9] Anastassia Ailamaki , David J. DeWitt , Mark D. Hill , Marios Skounakis, Weaving Relations for Cache Performance, Proceedings of the 27th International Conference on Very Large Data Bases, p.169-180, September 11-14, 2001
- [10] Yongqiang He , Rubao Lee , Yin Huai , Zheng Shao , Namit Jain , Xiaodong Zhang , Zhiwei Xu, RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems, Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, p.1199-1208, April 11-16, 2011
- [11] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, R. Murthy, "Hive - A Warehousing Solution Over a Map-Reduce Framework," In Proc. of Very Large Data Bases, vol. 2 no. 2, August 2009, pp. 1626-1629.
- [12] Gzip compression algorithm, <http://www.gnu.org/software/gzip/>
- [13] Kestelyn, J. "Introducing Parquet: Efficient Columnar Storage for Apache Hadoop." <http://blog.cloudera.com/blog/2013/03/introducing-parquet-columnar-storage-for-apache-hadoop/>
- [14] Cloudera Impala. <http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html>.
- [15] Snappy. [http://en.wikipedia.org/wiki/Snappy_\(software\)](http://en.wikipedia.org/wiki/Snappy_(software))
- [16] A. Floratou, U. F. Minhas, and F. Ozcan. Sql-on-hadoop: Full Circle Back to Shared-Nothing Database Architectures. Proceedings of the VLDB Endowment, 7(12), 2014
- [17] ZIV, J., AND LEMPEL, A. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 23 (1977), 337–343.
- [18] ZIV, J., AND LEMPEL, A. Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory 24 (1978), 530–536.
- [19] Thomas. C. Bell, J. G. Cleary, and I. H. Witten. Text Compression, Prentice-Hall, Englewood Cliffs, NJ, 1990

- [20] D. A. Huffman "A method for the construction of minimum redundancy codes",
Proceedings IRE, vol. 40, pp.1098 -1101 1962
- [21] Rissanen, J.J. Generalized Kraft inequality and arithmetic coding. IBM 1. Res. Dev. 20
(May 1976), 198-203. Another early exposition of the idea of arithmetic coding.
- [22] Bzip2 / Bzip. <http://www.bzip.org>
- [23] I. H. Witten , R. M. Neal and J. G. Cleary "Arithmetic Coding for data compression",
Commun. ACM, pp.520 -540 1987
- [24] R. Bose *Information Theory, Coding and Cryptography*, 2002 :Tata McGraw-Hill
- [25] Lzma. http://en.wikipedia.org/wiki/Lempel-Ziv-Markov_chain_algorithm
- [26] G. Manzini "An analysis of the Burrows-Wheeler transform", *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pp.669 -677 1999 [online] Available:
www.imc.pi.cnr.it/~manzini/tr-99-13/
- [27] Python. <https://www.python.org/>
- [28] <http://jmoiron.net/blog/python-serialization/http://jmoiron.net/blog/python-serialization/>
- [29] D. Marpe , T. Wiegand and G. J. Sullivan "The H.264/MPEG4 advanced video coding
standard and its applications", *IEEE Commun. Mag.*, vol. 44, no. 8, pp.134 -144
2006
- [30] Συνδυαστική Βελτιστοποίηση Σημειώσεις Β. Ζησιμόπουλος Ιανουάριος 2007.
<http://cgi.di.uoa.gr/~vassilis/co/comb-opt.pdf>
- [31] Herald Kllapi, Lefteris Stamatogiannakis, Manolis Tsangaris, Yannis Ioannidis,
"TRIEME: Sailing through Flows of Big Data ", *HDMS, Athens, Greece, July 2014*,
2014
- [32] Madis. <http://code.google.com/p/madis/>
- [33] TPC-H. <http://www.tpc.org/tpch/>
- [34] Bentley JL, Sleator DD, Tarjan RE, Wei VK. A locally adaptive data compression
scheme. *Communications of ACM* 1986;**29**(4):320–330.