ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ,
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Δομή Λεξικού
για Ασύρματα Δίκτυα

Τσακιρίδης Φώτιος

Επιτροπή: Μποζάνης Παναγιώτης, Επιβλέπων
Κατσαρός Δημήτριος

ΠΕΡΙΛΗΨΗ

Η κατανάλωση ενέργειας και η αποδοτηκότητα πρόσβασης είναι δύο βασικοί στόχοι και ανταγωνιστικοί στην ασύρματη εκπομπή δεδομένων. Για να αντιμετωπιστεί το ενεργειακό πρόβλημα στην ακολουθιακή αναζήτηση μαζί με τα δεδομένα έχουν προστεθεί δείκτες (index). Στην εργασία αυτή προτείνουμε ένα παραμετροποιήσημο αλγόριθμο εκπομπής δεδομένων τον **Interpolation Index**. Ο αλγόριθμος έχει δυνατότητα να βελτιστοποιήσει το χρόνο αναζήτησης κρατώντας σταθερό τον χρόνο συντονισμού (tuning time) και αντίστροφα.

1.ΕΙΣΑΓΩΓΗ

Η ραγδαία ανάπτυξη της ασύρματης τεχνολογίας μαζί με την ανάπτυξη των έξυπνων κινητών συσκευών οδήγησαν στην εξάπλωση των υπηρεσιών πληροφοριών , οι οποίες παρέχουν πρόσβαση σε πληροφορίες σε μεγάλο πλήθος πελατών, παντου και κάθε ώρα μεσω σημείων πρόσβασης και ασύρματων καναλιών. Επιπρόσθετα η μεγάλη πρόοδος στην κατασκευή πολύ μικρών κυκλωμάτων καθώς και κυκλωμάτων μικρής κατανάλωσης, συνδιαζόμενα με ισχυρές μπαταρίες μικρού μεγέθους έχουν κάνει την κατασκευή ασύρματων αισθητήρων πραγματικότητα.

Ο χρόνος προσπέλασης είναι ένας κοινός στόχος σε πολλά συστήματα όπως βασεις δεδομένων, αλλα η κατανάλωση ενέργειας είναι ένας ζωτικός στόχος στα ασύρματα δίκτυα. Αυτό συμβαίνει διότι οι κινητοί πελάτες έχουν περιορισμένα αποθέματα ενέργειας. Για καλύτερη κατανάλωση ενέργειας οι κινητοί πελάτες υποστηρίζουν δύο καταστάσεις λειτουργίας , την ενεργή κατάσταση και την κατάσταση ύπνου όπου έχουμε την μικρότερη κατανάλωση ενέργειας. Η αναλογία κατανάλωσης ανάμεσα στις δύο καταστάσεις είναι τεράστια. Για παράδειγμα ένας κόμβος αισθητήρας σε κατάσταση ύπνου καταναλώνει 7-20 φορές λιγότερη ενέργεια.

Όπως φαίνεται ο χρόνος αναζήτησης και ο χρόνος συντονισμού είναι δύο όροι τελείως ανταγωνιστικοί. Ο κινητός πελάτης για να πάρει μια πληροφορία που θέλει πρέπει να ακούει συνεχώς στο κανάλι σε κατάσταση ενεργή. Έτσι αναγκάζεται να διαβάσει πληροφορίες που δεν τον ενδιαφέρουν μέχρι να βρεί την πληροφορία που θέλει σπαταλώντας την ενέργεια του. Η λύση είναι να συνδιάσουμε τον χρόνο αναζήτησης και τον χρόνος συντονισμού για να έχουμε το καλύτερο αποτέλεσμα. Οι πελάτες πρέπει να μπορούν να μπούν σε κατάσταση ύπνου και να συγχρονιστούν ξανά με τον εξυπηρέτη όταν είναι να πάρουν την πληροφορία που τους ενδιαφέρει. Για να γίνει αυτό πρέπει να υπαρχει σε κάθε εκπεμπόμενη πληροφορία ένα ευρετήριο. Ο κινητός πελάτης αφου συγχρονιστεί με το εξυπηρέτη και διαβάζει το ευρετήριο της πληροφορίας που εκπέμπεται εκείνη την ώρα και μπαίνει σε κατάσταση ύπνου μέχρι την στιγμή που θα έρθει η πληροφορία που θέλει.

2.ΠΡΟΚΑΤΑΡΤΙΚΑ

2.1 Βασική ιδέα

Έχουμε ένα γενικό σύστημα εκπομπής δεδομένων, όπου ο εξυπηρέτης κυκλικά εκπέμπει μια συλλογή από ταξινομημένες πληροφορίες σε ένα ασύρματο κανάλι. Οι κινητοί πελάτες συντονίζονται με το κανάλι εκπομπής και ξεκινάνε την διαδικασία αναζήτησης της πληροφορίας που θέλουνε. Ο εξυπηρέτης για να βοηθήσει την διαδικασία αναζήτησης προσθέτει στα εκπεμπόμενα αντικείμενα ευρετήρια δημιουργώντας ένα κύκλο εκπομπής. Κάθε εκπομπή είναι οργανωμένη ως μια ακολουθία από ίδιου μεγέθους αντικείμενα (buckets) που είναι η μικρότερη μονάδα πληροφορίας που έχει πρόσβαση ο κινητός πελάτης. Κάθε αντικείμενο είναι κατηγοριοποιημένο είτε σε αντικείμενο δεδομένων είτε σε αντικείμενο ευρετηρίου. Μερικές φορές ένα αντικείμενο ευρετηρίου μπορεί να περιέχει και δεδομένα και ονομάζεται υβριδικό.

Ανάλογα με την συχνότητα εκπομπής δεδομένων η εκπομπή μπορεί να οργανωθεί σε επίπεδη , όταν κάθε αντικείμενο εκπέμπεται μόνο μια φορά, και σε skewed, αν κάθε αντικείμενο εκπέμπεται περισσότερο από μια φορές . Η επιλογή ανάμεσα στα δύο εξαρτάται από το εάν είναι γνωστά ή όχι τα μοτίβο προσπέλασης του χρήστη. Οι εκπομπές μπορεί να ορισμένες ως clustered ή non clustered. Μία εκπομπή ονομάζεται clustered όταν όλα τα αντικείμενα με την ίδια τιμή ιδιότητας εμφανίζονται διαδοχικά, αλλιώς δεν είναι clustered. Άρα η clustered εκπομπή σχετίζεται με την επίπεδη εκπομπή όταν το χαρακτηριστικό είναι πρωτέυον. Από την άλλη τα non-clustered σχετίζονται είτε με την επίπεδη εκπομπή είτε με την skewed εκπομπή.

2.2 Συναφείς εργασίες

Η ασύρματη εκπομπή δεδομένων έχει λάβει πολύ ενδιαφέρον τα τελευταία χρόνια. Το (1 , m)-indexing παρουσίαστηκε ως μια μέθοδος καταμερισμού ευρετηρίου σύμφωνα με την οποία η πληροφορία ευρετηρίου εκπέμπεται m φορές κατά την διάρκει κάθε εκπομπής. Το βασικό πρόβλημα με τον (1 , m)-indexing είναι η αναπαραγωγή ολόκληρου του ευρετηρίου m φορές με αποτέλεσμα να μεγαλώνει πολύ τον κύκλο εκπομπής και κατά συνέπεια τον μεσο χρόνο αναζήτησης. Ένας αλλός αλγόριθμος είναι ο distributed indexing. Οι πληροφρίες είναι οργανωμένες με ένα B+ δένδρο. Επίσης υπαρχει και ο flexible index όπου οι ταξινομημένες ακολουθίες δεδομένων διαμοιρασμένες σε αρκετά ίδιου μεγέθους τμήματα.Στο ξεκίνημα κάθε τμήματος είναι τοποθετημένα ένα γενικό και ένα τοπικό ευρετήριο. Το γενικό ευρετήριο σε κάθε τμήμα έχει ένα λογαριθμικό αριθμό από m ζευγάρια (κλειδί , δείκτης) τα οποία χωρίζουν τα τα τμήματα σε σε επιπλέον m+1 υπο τμήματα και κατά συνέπεια εμπεριέχει ένα εσωτερικό κατάλογο. Έπειτα προτάθηκε από τους Seifert και Hung ο flexible distributed indexing ο οποίος χωρίζει την εκπομπή σε ίσου μεγέθους τμήματα, τέτοια ώστε να μην ξεπεραστεί ένα δοσμένο όριο χρόνου συντονισμού σε συνδιασμό με πολύπλεξη ενός πυκνού B+ δένδρου. Έπειτα προτάθηκε ο Exponential index τον οποίο αναλύουμε παρακάτω.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1-1	4
2-3	8
4-7	12
8-15	2

Εικόνα 1

6
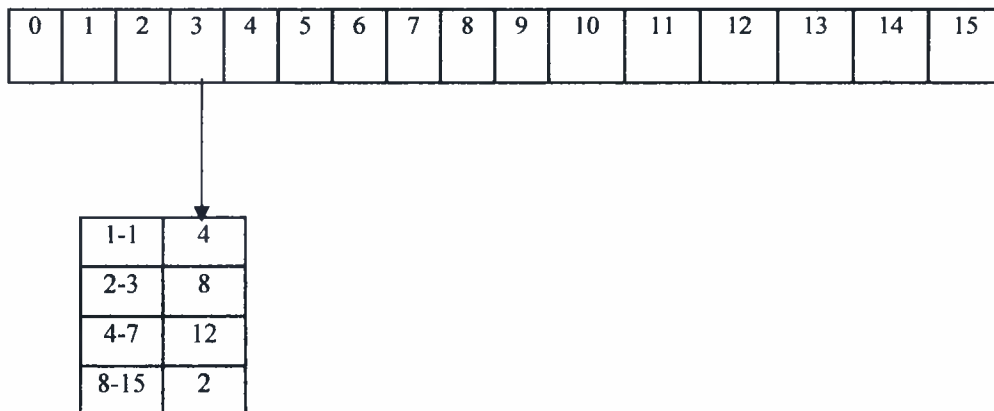
Σύμφωνα με την παραπάνω εικόνα 1 ο exponential index αποτελείται από υβριδικά αντικείμενα τα οποία είναι ταξινομημένα. Κάθε αντικείμενο περιέχει ένα ευρετήριο.Το εύρετήριο αποτελείται από λογαριθμικό πλήθος εγγραφών.Κάθε εγγραφή αποτελείται από ένα ζευγάρι (δεικτης , μεγ.κλειδί).Κάθε δείκτης δείχνει σε ένα αντικείμενο και το μεγ.κλειδί είναι το μέγιστο κλειδί του τμήματος που αντιπροσωπεύει το κλειδί.

r^i $r^{i+1}-1$	Max Key

Εικόνα 2

Η δομή του ευρετηρίου φαίνεται στην εικόνα 2 όπου r είναι η λογαριθμική βάση.Ο χειρότερος χρόνος αναζήτησης είναι $\log_r(n)$ και ο μέσος χρόνος αναζήτησης είναι $\frac{1}{2}\log_r(n)$

3.INTERPOLATION AIR INDEX

Η μέθοδος που προτείνουμε σε αυτην την εργασία εκμεταλλεύεται την ιδέα της εφαρμογής αναδρομικού τεμαχισμού τετραγωνικής απόστασης με την οποία καταφέρνουμε να έχουμε λογ-λογαριθμικό χρόνο. Πρώτα εξετάζουμε την ιδέα πίσω από το λογ-λογαριθμικό σχήμα εφαρμόζοντας το σε ένα πίνακα με αντικείμενα.(Τα αντικείμενα δεν εκπέμπονται αλλά βρίσκονται στην μνήμη του υπολογιστή).

## 3.1 Η ιδέα πίσω από το log-log σχήμα

Υποθέτουμε ότι έχουμε ένα πίνακα αριθμών ταξινομημένων με αύξουσα σειρά $x_1 < x_2 < x_3 < \ldots < x_n$. Οι αριθμοί έχουν ομοιόμορφη κατανομή στο εύρος (x_0 , x_{n+1}) .Ψάχνουμε το στοιχείο y ως εξής.Έστω

$$p = \frac{y - x0}{xn + 1 - x0}$$

το ποσοστό τον κλειδιών που περιμένουμε να είναι μικρότερα από το y . Τότε συγκρίνουμε το y με το $x(p*n)$ και στην περίπτωση που δεν είναι όμοια ψάχνουμε αναδρομικά στο υποπίνακα $x_1, x_2, \ldots x_{[p*n]}$ -1 ($y < x_{[p*n]}$) ή στον υποπίνακα $x_{[p*n]+1}$, $x_{[p*n]}+2, \ldots, x_n$ ($y > x_{[p*n]}$).

Ο χρόνος προσπέλασης είναι log-log(n) στην μέση περίπτωση και γραμικό στην χειρότερη.

3.2 Περιγραφή του Interpolation index
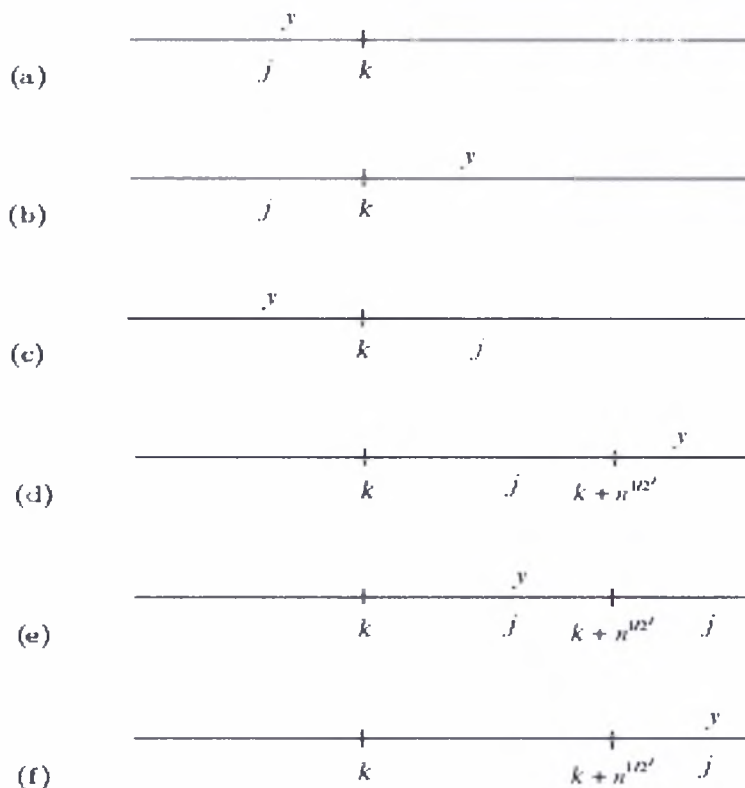
Εικόνα 3

Η εκπομπή στην πρόταση μας αποτελείται από n υβριδικά αντικείμενα $b_1, b_2, \ldots b_n$. Οι ενσωματομένες εγγραφές του ευρετηρίου θα οδηγήσουν την αναζήτηση, εξομειώνοντας τον interpolation search στην ασύρματη εκπομπή.

1. M_i , m_i θα είναι η μέγιστη και η ελάχιστη εγγραφή του ι-οστου αντικειμένου.
2. Αρα κάθε αντικείμενο ανήκει το εύρος [m1 . Mn];
3. m_i και M_n θα είναι επίσης διαθέσιμα σε κάθε αντικείμενο
4. $F_R(y) = \text{Prob} [Y <= y | Y \; \varepsilon \; R]$.

Βασισμένοι στην αποθηκευμένη πληροφορία , το πρωτόκολλο είναι το εξής :
Έστω το k είναι το αντικείμενο στο οποίο έχει συντονιστεί ο πελάτης, y είναι το αντικείμενο που ψάχνει, l το επιπεδο της αναδρομής και j η προβλεπόμενη

θέση που υπολογίζεται με την F. Οταν το y ανήκει στο εύρος κλειδιού είτε του k είτε του k+1 ,ή είναι ίδιο με τη μέγιστη τιμή M_t του ευρετηρίου τότε δεν έχουμε να κάνουμε πολλα να κάνουμε. Στην συνέχεια υποθέτουμε ότι καμία από αυτές τις περιπτώσεις δεν συμβαίνουν.

Κατά την διάρκεια της πρώτης φοράς που ο πελάτης συγχρονίζεται στο κανάλι υπάρχουν τρείς πιθανότητες εκτός από τις γενικές περιπτώσεις.Οταν ταυτόχρονα το j και το y βρίσκονται μπροστά από το k (εικόνα 3α) η υπόλοιπη εκπομπή είναι ασήμαντη. Παρόλα αυτά ελεγχουμε εάν $\frac{j}{\sqrt{n}}$ <= c_d , c_d παράμετρος βελτιστοποίησης.Εαν ισχύει ο κινητός πελάτης κοιμάται μέχρι να έρθει το αντικείμενο 1, και μετα κάνει γραμμικό ψάξιμο με $\sqrt{n}$ άλματα , χρησιμοποιώντας επιπέδου 1 δείκτη , μέχρι να εντοπίσει το αριστερο όριακό αντικείμενο του $\sqrt{n}$ - μεγέθους υποφακέλου που περιέχει το y, στο οποίο επαναλαμβανόμαστε . Στην μέση περίπτωση το y εμπεριέχεται $\pm 2\sqrt{n}$ τμήματα από το j , και το αντικείμενο 1 είναι αρκετα κοντα στο j.Προσαρμοζόμαστε σε αυτή την συντηρητική πολιτική για να αποφύγουμε πιθανή δεύτερη αποτυχημένη εκπομπή. Από την άλλη ο πελάτης μεταβαίνει σε κατάσταση ύπνου και μέχρι το αντικείμενο j - $c_a\sqrt{n}$ αφιχθεί, c_a είναι παράμετρος βελτιστοποίησης. Παλι προτιμάμε να να κάνουμε λίγο πίσω από το j ανταλλάσοντας μια αύξηση του χρόνου συντονισμού ξοδεύοντας λίγο χρόνο αναζήτησης μειώνοτας την πιθανότητα μιας δεύτερης εκπομπής . Σε περίπτωση επιτυχημένης πρόβλεψης εφαρμόζουμε το παραπάνω γραμμικό ψάξιμο για να σκανάρουμε την περιοχή σε μια υπομετάδοση του $\sqrt{n}$. Αν συμβεί και δεύτερη αποτυχημένη εκπομπή κάνουμε γραμμικο ψάξιμο ξεκινόντας από το αντικείμενο 1.

Στην δεύτερη περίπτωση, το j προηγείται του k και y είναι μετα το j και πριν το y (εικόνα 3β).Εδώ μεταβαίνουμε από την κατάσταση υπνου στην κατάσταση λειτουργίας και το ανάστροφο με μεσολάβηση $\sqrt{n}$ τμημάτων μέχρι να βρούμε το το $\sqrt{n}$ τμήμα που θέλουμε και κάνουμε αναδρομή. Όταν το y προηγείται το k και το j βρίσκεται μετά το k (εικόνα 3γ) ο κινητός πελάτης χάνει τη τρέχων εκπομπή. Αυτή η κατάσταση είναι παρόμοια με την

περίπτωση της εικόνας 3 α με το k να αντικαθιστά το j. Ποιο ειδικά αν $\dfrac{k}{\sqrt{n}}$

<= c_d, ο πελάτης κοιμάται μέχρι την άφιξη του αντικειμένου l για να γίνει το γραμμικό ψάξιμο .Αλλιώς ο πελάτης πάει πάλι σε ενεργό κατάσταση όταν το αντικείμενο k - $c_a\sqrt{n}$ αφιχθεί .Αν είναι τυχερός στην πρόβλεψη η γραμμική αναζήτηση αρχίζει. Αλλιώς η εκπομπή δεν έχει χρησιμότητα, οπότε ο πελάτης κοιμάται και κάνει γραμμικό ψάξιμο από το αντικείμενο l.Γυρνόντας στην γενική περίπτωση (l >= 1) ο κινητος πελάτης ήδη ξέρει ότι το y βρίσκεται ανάμεσα από από το αντικείμενο k και min (k + $\sqrt[2l-1]{n}$, n } , το interpolation επιστρέφει ένα ευρετήριο j που ανήκει στο { k , min (k + $\sqrt[2l-1]{n}$, n } , και το μήκος των αλμάτων κατά την διάρκεια του γραμμικού ψαξίματος είναι $\sqrt[2l]{n}$.Υπάρχουν τρείς πιθανότητες . Στην πρώτη το y φαινεται να ανήκει μετα τη θέση του δείκτη επιπέδου l πχ το αντικείμενο k + $\sqrt[2l]{n}$ όσο το j βρίσκεται μεταξυ του k και του k + $\sqrt[2l]{n}$.Μετα κοιμόμαστε μέχρι το αντικείμενο k + $\sqrt[2l]{n}$ μεταδοθεί και ξεκινήσουμε το γραμμικό ψάξιμο.Οταν το το y εντοπίζεται ανάμεσα ανάμεσα του k και k + $\sqrt[2l]{n}$ (εικονα 3e) άσχετα της θέσης του j δεν έχουμε να κάνουμε τιποτα.Το $\sqrt[2l]{n}$ υπο τμήμα στο οποίο πρέπει να κάνουμε αναδρομή έχει βρεθεί.Στην τελευταία περίπτωση (εικονα 3f) και το y και το j βρίσκονται μετά το αντικείμενο k + $\sqrt[2l]{n}$. Αν j - $c_a\sqrt{n}$ < k + $\sqrt[2l]{n}$ ισχύει κάνουμε γραμμικό ψάξιμο στην υπολοιπη εκπομπή από το τμήμα k + $\sqrt[2l]{n}$. Αλλιώς κοιμόμαστε μέχρι το αντικείμενο j - $c_a\sqrt{n}$ αφιχθεί .Αν η προσαρμοσμένη πρόβλεψη αποδειχθεί αληθινή εκτελούμε το γραμμίκο ψάξιμο.Αλλιώς χάσαμε την εκπομπή και το γραμμικό ψάξιμο γίνεται από το ασφαλές αντικίμενο k + $\sqrt[2l]{n}$.

4.ΠΕΙΡΑΜΑΤΑ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑ

Περισσότερες λεπτομέρειες ο αναγνώστης μπορεί να βρεί στο παράρτημα

ΠΑΡΑΡΤΗΜΑ

Interpolating the Air
for Optimizing Wireless Data Broadcast

Fotis Tsakiridis
Computer & Communication
Engineering Department,
University of Thessaly,
Volos, Greece
fotsakir@uth.gr

Panayiotis Bozanis
Computer & Communication
Engineering Department,
University of Thessaly,
Volos, Greece
pbozanis@inf.uth.gr

Dimitrios Katsaros[*][†]
Computer & Communication
Engineering Department,
University of Thessaly,
Volos, Greece
dkatsaro@csd.auth.gr

ABSTRACT

Energy conservation and access efficiency are two fundamental though competing goals in broadcast wireless networks. To tackle the energy penalty from sequential searching, the interleaving of *index* with data items has been proposed. Although, quite important contributions exist in the literature on providing broadcast indexes, they present significant shortcomings. This article proposes a novel parameterized air index, the *interpolation index*, which is a tunable structure able to optimize the latency with the tuning time kept at a given amount, and vice versa. Theoretical and experimental results attest that the novel indexing structure outperforms state-of-the-art air indexing schemes and demonstrate its great flexibility in trading access latency with tuning time.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; E.1. [**Data**]: Data Structures—*Distributed data structures*

General Terms

Design, Experimentation, Measurement, Performance.

Keywords

Indexing, Interpolation Search, Energy Conservation, Latency, Tuning Time, Broadcasting, Cellular Networks, Mobile Computing, Wireless Networks.

1. INTRODUCTION

The rapid advent of wireless technology along with the growing popularity of smart mobile devices, led to the deployment of pervasive information services, which provide access to "ambient" information to large number of audiences (clients), anywhere, and anytime, through access points and a number of wireless channels. Additionally, the advances in miniaturization and the creation of low-power circuits, combined with small-sized batteries have made the development of wireless sensor networks a working reality, boosting the deployment of wireless networks even further and pushing the challenges concerning their development to the limits.

Consider the following scenario encountered in a cellular wireless network (e.g., a PCS), where resource-constraint mobile units within a wireless cell, retrieve information from a relational database, whose contents are repetitively broadcast by a base station serving the cell; in the general case, the information pieces consist of thousands of "projections" (i.e., columns) of relational table rows. The existence of wireless data broadcast service providers, such as Ambient[1], Microsoft[2], and SkyTel[3] confirms the industrial interest in such kind of services and exhibits their feasibility.

In a second scenario from sensor network applications, where a node (assumed to be energy-rich, due to its special duties) with various sensing capabilities (e.g., temperature, humidity, pressure, carbon-dioxide concentration), is able to sense the environment at regular time intervals (different for each measured quantity). These measurements are repetitively broadcast and collected by surrounding energy-starving nodes/sensors, which implement various application protocols, dealing with aspects of temperature only, of gas only, of concentrations only, etc., or with various combinations of them. Such an application scenario could be built on the basis of the architecture described in [5].

In these application scenarios, it is evident that:

- The information "consumers" need to retrieve the data as quick as possible (i.e., with small *access latency*, which is the time elapsed between when the need for a

datum arises in a node and the moment the node gets that datum from the channel).

- The consumers are energy-starving nodes; therefore they should refrain from continuously monitoring the broadcast channel (i.e., pursuit a small *tuning time*, which is the amount of time a node spends while monitoring the channel).
- There could be several thousands of broadcast items, thus scalability in terms of broadcast items is a very important issue.

Access efficiency is a common target in many systems, (e.g., databases), but energy conservation is a vital goal in wireless networks for prolonging the longevity of the sensor network or for guaranteeing as much power-independence as possible for the mobile hosts. To achieve energy savings, mobile nodes support two generic modes of operation, the *active* mode, which is a fully operational state, and the *doze* mode[4], which is a power saving state. The ratio of energy consumption between the two modes is usually an order of magnitude [27]. Similarly, sensor nodes can be in one of three *active* states – transmit, receive, idle – or in *sleep* state; a sensor in the sleep state consumes 7–20 times less energy than when it is in the idle state [6].

As it is easily seen, access latency and tuning time are competing each other: to acquire the requested data as soon as possible, the client must actively listen to the broadcasting channel—retrieving mainly unwanted information — therefore consuming energy, and vice versa. Apparently, every solution for trading off access latency and tuning time must provide some auxiliary information which allows clients to alternate from doze mode to active mode when browsing the broadcast data. Thus, clients can remain in the doze mode most of the time and tune selectively into the broadcast channel only when significant data arrive. This set up calls out for a sort of directory or indexing information which specifies the arrival times of particular items over the broadcast channel. By accessing this index, known as *air index*, mobile clients are able to predict the arrival of desired data. Obviously, this scheme demands additional bandwidth for broadcasting the index. However, its costless scalability feature compensates for the incurred overhead and, so, it became the method of choice.

Traditional database indexing techniques are designed for random access storage media which permit back-tracking during the search operation. On the other hand, broadcasting is equivalent to a sequential access medium: in order to benefit from the indexing information, the client has to wait until the next index entry point is transmitted. The same also holds in case of bad transmitted index items; re-entry to the next broadcasted entry point is the only solution to such "linkage" errors. To cope with this problem, two main approaches were introduced: i) multiple replicated indexes are interleaved with data broadcast [13]; and ii) transmitted data are linked together through index information so that multiple entry points exist during each broadcast cycle [12].

In the following, we deal with the second approach since proven superior to the first one. The most prominent member of this category of air indexes is the *exponential index* [28, 29]. This scheme can be loosely seen as the linearization of a directed acyclic graph (dag), laid over the sorted sequence of data items to be broadcasted, so that

[4]Following the terminology of [13].

the client can simulate the binary search operation, irrespectively of the time point he tunes into. Alternatively, it can be seen as a distributed implementation of skip lists, i.e., a skip list with multiple entry points. As a result, the exponential index shows logarithmic access complexity and resilient behavior to link errors due to bad transmitted packets. Even though this access time complexity is acceptable for a few hundreds of items, still is not satisfactory when dealing with a few thousands of items.

In this paper, we are mainly motivated by the scalability problem of the indexing structures, and improve upon the state-of-the-art indexing method, namely the exponential index. Deviating from the philosophy of this index, we recursively apply a squared-distance partitioning of the linear broadcast order, which, effectively, overlays index search paths of log-logarithmic length in the average case. Additionally, we implement this technique in a distributed fashion, so that it allows index access from any tuning point, permitting the recovery from link errors quickly and easily. This novel air indexing scheme is called the *interpolation air index*. Theoretical and experimental results attest that the novel indexing structure outperforms state-of-the-art air indexing schemes and demonstrate its great flexibility in trading access latency with tuning time.

The rest of the paper is organized as follows. Section 2 reviews basic notions and related work. Section 3 introduces the interpolation air index. In Section 4 we provide both theoretical and experimental evidence on the superiority of our scheme over exponential index. Finally, Section 5 concludes our work.

2. PRELIMINARIES

2.1 Basic notions

We consider a generic data broadcasting system, where a server cyclically broadcasts a collection of totally ordered data items onto a down-link wireless channel. The mobile clients must tune into the broadcast channel and actively must find their way to the required information. In order to aid the search process, the server interleaves auxiliary index items with the actual data items, forming a broadcast cycle, i.e., *bcast*. Every bcast is organized as a sequence of equal sized *buckets*, which constitute the smaller unit of information a mobile client has access to. Each bucket is categorized as being either *data bucket* when it contains a number of (pure) data items or *index bucket*, in case it holds index information. Sometimes, an index bucket may accommodate and some data items, and, then, it is termed as *hybrid* one. In this context, a pointer to a specific bucket is defined as an offset from the bucket containing the pointer to the bucket to which the pointer points to, and designates the number of basic bucket transmission time units one has to wait to start retrieving the pointed bucket. It is widely accepted that every data bucket contains a pointer to the immediately upcoming index bucket. As it is easily seen, the employment of index buckets reduces the tuning time, however it increases the access time.

Figure 1 exemplifies the general data organization of an indexed broadcasted file and the protocol for retrieving its data. When the mobile client tunes into the channel, he reads data bucket d_3. So, he switches to doze mode until the time index bucket I_2 arrives. Then, the index searching phase is commencing, during which, according to the desired
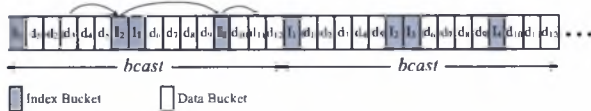
Figure 1: Searching in an indexed broadcast.

query, the client listens to a number of index buckets. In our example, after retrieving I_2, the client downloads I_4 and decides that the required data is contained into data bucket d_{11}. When this bucket is accessed, the actual data retrieval takes place. In the described scenario, the tuning time is 4, while the access latency equals 12. Here we must note that in the literature, the time interval between the point the client initially probes the channel to the point when the next upcoming, relevant to the inquiry, index bucket is read, sometimes is referred to as *probe time*. In our example, the probe time is 3.

Based on data item broadcast frequency, a bcast can be categorized as *flat*, when every item is transmitted exactly once, and *skewed*, if some items occur more than once during the entire schedule. The choice between these two alternatives depends heavily on whether user access patterns are known or not. Bcasts can be further classified as *clustered* and *non-clustered* [13] based on the broadcast schedule of same valued data items. Namely, a broadcast is clustered when all data items with the same value of the attribute appear consecutively; otherwise, it is non-clustered. So, clustered broadcast is equivalent to flat broadcast when the attribute is primary. On the other hand, non-clustered broadcast corresponds to either flat broadcast with respect to a secondary attribute or skewed broadcast of a primary attribute.

2.2 Related work

Air-indexing has received much attention during the last years after its introduction in the seminal papers [12, 13]. In [13] $(1, m)$-indexing was introduced as an index allocation method, according to which the index information is broadcasted m times during each bcast. The main problem with the $(1, m)$-indexing scheme is the replication of the entirety of the index m times, since it prolongs the broadcast cycle and thus the average access time. In the same paper, a tree-based indexing method, called distributed indexing, was also suggested: the data file is associated with a B^+-tree [3], and since the wireless channel is a sequential medium, the formed tree is linearized with a pre-order traversal. Additionally, the first k levels of the index are partially replicated in the broadcast, while the remaining levels are not. The nodes at the replicated levels are repeated at the beginning of the first broadcast of each of their children. Compared to the $(1, m)$-index, the tree-based scheme has lower access time due to its shorter broadcast cycle while its tuning time is analogous to that of the $(1, m)$-index.

In [12] it was exhibited how hash functions can be used for allocating data items to the slots in the broadcast schedule. Since collisions — that is, multiple items are mapped to the same slot — are inevitable, the authors adopted the linear probing method [4] for collision resolution. Namely, overflow items are relocated into succeeding slots, pushing forward every item originally hashed to them, and, thus, penalize them with an extra tuning time of one slot. The

hash-based scheme incurs minimal overhead, compared to the over indexing techniques, since only the hash function is broadcast together with data. However, one has to examine the entirety of the implicit partitions, corresponding to areas of overflowed items, to find the desired item. This may incur high tuning time, especially for large partitions. This approach was extended in [30] to the case of skewed bcasts, by introducing the *MHash air-index*. MHash maps data items to broadcast slots using a two-argument hash function, so that the broadcast schedule is free of unoccupied slots, while the spacing between instances of each data item and the bandwidth allocation follow the underlying distribution.

In [12], the flexible indexing scheme was also proposed, according to which, the sorted sequence of data items is partitioned into several equal-sized segments. At the beginning of each segment, a global index and a local index are accommodated. The global index at a segment contains a logarithmic number of (key, pointer) pairs to guide the search towards succeeding segments. On the other hand, the local index holds m (key, pointer) pairs that split further the hosting segment into $m + 1$ subsegments, and, thus, comprises an inside directory. This approach was generalized in [28], by introducing the exponential index. Specifically, the sizes of the indexed segments increase exponentially by a base of $r \geq 1$, r being a system parameter. This approach was further studied in error-prone environments [29].

Seifert and Hung [24] suggested the flexible distributed indexing scheme which employs partitioning of the broadcast program into a number of equal-sized data segments, such that a given limit on the tuning time will not be exceeded, in conjunction with multiplexing of a dense B+tree-like index on the data items. Similar to $(1, m)$-indexing, index information is broadcast multiple times during a bcast. However, the broadcasted sequence is not indexed at its entire; every index tree refers to the data of immediately upcoming segments, with occasionally exceptions when the tuning time is bounded by a desired limit. Both [25, 2], considered unbalanced tree structures to optimize broadcast schedule for non-uniform data access, while Tan and Yu [26] studied scheduling policies for skewed bcasts.

Signatures have been successfully employed to facilitate retrieval in several types of databases and was adapted to broadcast environments in [15]. Data items are organized into groups and a single signature is generated for each group. During bcast, each signature is transmitted before its corresponding group of data items. A client firstly generates a query signature based on the same assumptions as the server did, and then by listening to the signatures selectively switches between doze and active mode. A hybrid among the signature method and the distributed index tree was proposed in [9] and applied it also to the multi-attribute indexing case [10, 11]. Remotely related to the present article, are the air indexing schemes that have been proposed for the case of multiple broadcasting channels [1, 8, 14, 22], and for querying spatial data, e.g., [16, 31].

For the case of wireless sensor networks, since the majority of research has focused for the moment on topics like routing, clustering, sleep scheduling, localization, medium access control, the issue of indexing has received much less attention and the literature has solely developed distributed indexes that reside on the sensor nodes and are not broadcast. These indexes comprise (in one form or another) adaptations of the traditional disk-based indexes, with special
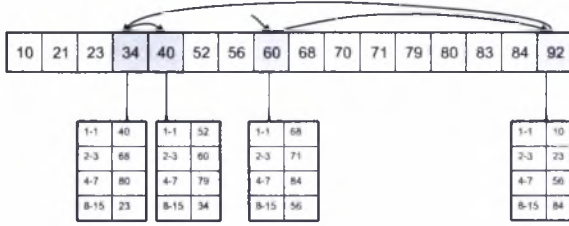
Figure 2: Searching with the exponential index.

care to achieve only local (to the extend possible) communication during their creation or maintenance, and small storage overhead. The GHT [23] is based on a (geographic) hashing scheme, DIM [17], DIFS [7] and DIST [19] are based on the quadtree structure, and TSAR [5] is based on Skip Graphs (a generalization of Skip Lists for distributed environments). None of these indexes is broadcast over wireless channels and they all assume global ordering for the data they index. We conclude this section by presenting more details about the exponential index, since it comprises the state-of-the-art in indexing for single broadcasting channel environments.

The Exponential index.

According to this scheme [28, 29], the bcast consists of hybrid buckets, which try to simulate the exponential searching technique [18] over the periodical broadcast. The index part of each bucket consists of logarithmic number of entries. Every entry consists of a pair $(pntr, maxKey)$ and indexes a segment of buckets, where $pntr$ is a pointer to the first bucket of the involved segment, and $maxKey$ is the maximum key value in the last bucket of this segment. Specifically, the first entry refers to a single bucket segment, the one immediately following, and every i-th entry refers to the segment of 2^{i-1} buckets that are from 2^{i-1} to $2^i - 1$ slots ahead; that is, the sizes increase exponentially by base of 2. It follows immediately that, firstly, the pointer can be inferred from the entry IDs, and, secondly, the key range of the buckets indexed by the i-th entry is bounded by the $maxKey$ values of the $(i-1)$-th and i-th entries.

Figure 2 provides an instance of exponential index searching, where, for simplicity, every bucket contains only one data element of integer type. Assume that a client needs to retrieve element 40 and he tunes into the broadcast at the 8-th slot. Since 40 belongs to the range of values covered by the fourth entry, he switches to doze mode until the 8th succeeding bucket arrives. By examining its index, he decides to be active again after 4 buckets. Then he figures out that the element he searches is one bucket ahead.

The authors suggested two generalizations: i) The sizes of segments increase exponentially by any base $r \geq 1$. Therefore, the i-th index entry refers to buckets that are from $\left\lfloor \frac{r^{i-1}-1}{r-1} + 1 \right\rfloor$ to $\left\lfloor \frac{r^i-1}{r-1} \right\rfloor$ away; and ii) To reduce the indexing overhead, the bcast is partitioned into data chunks of I buckets, and the exponential index is build on a per-chunk basis. Thus, in each chunk, the first bucket is hybrid, while the rest $I - 1$ are pure data buckets. To aid intra-chunk search, the hybrid bucket contains also a local index, of $I - 1$ entries, so that the i-th local entry holds the maximum key of the i-th data bucket within the local chunk. It is easy one

to see that the intra-chunk tuning time is either 1, for the first bucket in the chunk, or 2, for every pure data bucket. Given the size B of each pure data bucket, in number of data items it holds, and the size B' of a hybrid bucket, measured in number of data items it can accommodate, a formula is provided in [29] for calculating the smallest permissable value for base r.

In an error-free broadcasting environment, the average access latency is $I/2 + N/2$, N the size of bcast, the worst case tuning time is $\left\lfloor \log_{\frac{1}{r-1}}(C - 1) \right\rfloor + 1$, C the number of data chunks, the average tuning time equals to $\frac{(I-1)}{I} + \frac{B(I-1)}{B(I-1)+B'} + \frac{1}{C}\sum_{l=0}^{C-1} t(l)$, $t(l)$ being the tuning time for a data chunk that is l chunks away from the current chunk, while the index space overhead per chunk is $O(I + \log_r C)$. The authors in [29] also proved analytical formulae for both average access latency and average tuning time in error-prone environments.

3. INTERPOLATION AIR INDEXING

We described earlier how the exponential index conducts the tuning into the channel in order to get the desired data in such a fashion that the resulting performance is logarithmic in the number of broadcast data. The method proposed in the present article exploits the idea of recursively applying a squared-distance partitioning of the linear broadcast order, which, effectively, overlays index search paths of log-logarithmic length, in the average case. This idea is further improved by implementing it in a distributed fashion to allow for multiple "entry" points from the broadcast. Firstly, we exemplify the idea behind this log-logarithmic scheme by applying it in a ordinary array of items (the word 'ordinary' means that the items are not broadcast, but reside in a computer's main memory).

3.1 The idea behind the log-log scheme

Suppose that there is an array of totally ordered numeric values (keys) $x_1 < x_2 < \cdots < x_n$, drawn independently from a uniform distribution over the range (x_0, x_{n+1}). Searching by interpolation for the item y in the array proceeds as follows: Let $p = \frac{y - x_0}{x_{n+1} - x_0}$, that is the percentage of the keys expected to be less than y. Then, we compare y to $x_{\lceil p*n \rceil}$, and, in case of inequality, we search recursively either subarray $x_1, x_2, \ldots, x_{\lceil p*n \rceil - 1}$ $(y < x_{\lceil p*n \rceil})$ or subarray $x_{\lceil p*n \rceil + 1}, x_{\lceil p*n \rceil + 2}, \ldots, x_n$ $(y > x_{\lceil p*n \rceil})$.

The access time is $\log \log n$ in the average case, and linear in the worst case [20]. This approach, with the same time bounds, can be employed in every totally ordered data set whose cumulative distribution function is known.

A further improvement to this scheme can be achieved [21], namely the binary interpolation search, which modifies the actions after comparing y to $x_{\lceil p*n \rceil}$, as follows: If $y > x_{\lceil p*n \rceil}$, then y is successively compared with $x_{\lceil p*n + i\sqrt{n} \rceil}, i = 1, 2, \ldots$, to locate the smallest i such that $y < x_{\lceil p*n + i\sqrt{n} \rceil}$. If $y < x_{\lceil p*n \rceil}$, then y is successively compared with $x_{\lceil p*n - i\sqrt{n} \rceil}, i = 1, 2, \ldots$ In either case, the located pertinent subarray of size $\sqrt{n}$ is recursively searched.

It can be shown that the average access time is bounded by $2.03 \log \log n$, but its worst case time complexity is $\sqrt{n} + O(\sqrt[4]{n})$. It can be relatively easily showed that the worst case complexity of this approach can be further improved to $2 \log n$, if one employs exponential search to determine

i with $\log i$ comparisons [18], without affecting the average case performance.

3.2 Description of the proposed Interpolation index

The bcast in our proposal consists of n hybrid buckets, $b_1, b_2, \ldots, b_n$. The embedded index entries will guide the search, simulating the interpolation search in the wireless set up. In the following, i) M_i, m_i will denote the maximum and the minimum data entry, respectively, of i-th bucket — therefore, every data element belongs to the range $[m_1, M_n]$; iii) m_1 and M_n will be also available to every bucket; and iv) $\mathcal{F}_\mathcal{R}(y) = \mathbf{Prob}[Y \leq y | Y \in \mathcal{R}]$.

In the first version, apart from m_1 and m_n, the index part of k-th bucket, $k = 1, 2, \ldots, n$, will consist of pairs $(pntr, maxKey)$, where $pntr$ is an offset and $maxkey$ is the maximum data entry of the bucket $pntr$ slots ahead. The 0-th entry refers to the immediately upcoming bucket, while the i-th entry holds $m_{k+n^{2-i}}$, namely, the maximum data entry $n^{\frac{1}{2^i}}$ slots ahead; every such entry is characterized as i-th level. It follows immediately that:

LEMMA 1. *The space overhead of every accommodated index structure is $1 + \log \log n$ entries.*

Please note that the offsets are immediately inferred given the sequence id of the hosting bucket and, therefore, are not stored.

Based on stored information, the access protocol is as follows (please cf. Fig. 3): Let k be the bucket the client currently tuned into, y the search item, l the level of recursion, and j the probing position, calculated using $\mathcal{F}$. When y belongs to the key range of either k or $k + 1$, or equals to one of the maximum values M_i of the indexed buckets, then we do not have much to do. So, in the sequel we assume that none of these cases occurred.

During the first time the client tunes into (i.e., $l = 1$), three possibilities, besides the general case (see below) may happen. When both j and y lie ahead of k (Fig. 3(a)), the rest of the bcast is useless. Thus, we check whether $\frac{j}{\sqrt{n}} \leq c_d$, c_d a tuning distance parameter. If so, the client dozes until bucket 1 arrives, and then he conducts linear search with $\sqrt{n}$ long jumps, using level 1 pointers, until he locates the left delimiting bucket of the $\sqrt{n}$-sized subfile containing y, to which we recur; since, on the average, y lies within approximately $\pm 2\sqrt{n}$ slots from j ([21]), and bucket 1 is close enough to j, we adopt this conservative policy, to avoid a possible second missed bcast. Otherwise, the client switches to doze mode until bucket $j - c_a\sqrt{n}$ arrives, c_a being an interpolation adjustment parameter. Again, we prefer to "step" back a little from j, trading-off an increase of tuning time with sparing some latency time, by decreasing the probability of a second consecutive unutilized bcast. In case of successful prediction, we start employing the above mentioned linear search to narrow down the area of interest in a sub-transmission of $\sqrt{n}$ extent. If, however, a second missed bcast does happen, we employ the linear search starting from bucket 1.

In the second case, j precedes k and y succeeds both of them (Fig. 3(b)); we simply start switching between doze and active mode, with $\sqrt{n}$ intervening slots, until we locate the desired $\sqrt{n}$-sized interval, to which we recur. When y precedes k and j succeeds k (Fig. 3(c)), the client missed
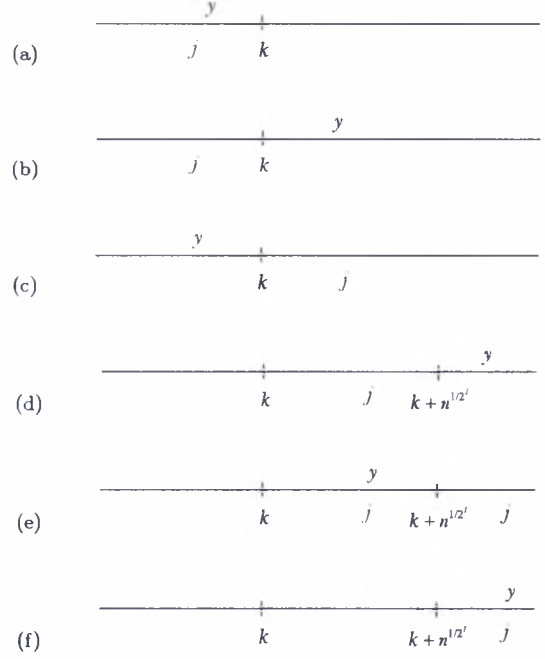

Figure 3: Relative positions of k, j and l: (a)-(c) can happen only when $l = 1$, (d)-(f) general recursive cases ($l \geq 1$).

the current bcast. This situation is similar to the one of Figure 3(a), with k substituting j. Specifically, if $\frac{k}{\sqrt{n}} \leq c_d$, the client dozes until the arrival of bucket 1 to conduct the linear search. Else, the client goes back to active mode when bucket $k - c_a\sqrt{n}$ arrives. If he is fortunate in his prediction, linear searching commences; otherwise, the bcast is useless, he dozes and applies the linear search starting from bucket 1.

Turning now to the general case ($l \geq 1$), the client already knows that y is lying between buckets k and $\min\{k + 2^{l-1}\sqrt{n}, n\}$, the interpolation obviously returns an index $j \in [k, \min\{k + 2^{l-1}\sqrt{n}, n\}]$, and the length of jumping during linear searching equals to $\sqrt[2^l]{n}$. There are also tree possibilities. In the first one, y seems to belong after the position the level l pointer points to, i.e., bucket $k + \sqrt[2^l]{n}$, while j lies between k and $k + \sqrt[2^l]{n}$ (Fig. 3(d)). Then, we doze until bucket $k + \sqrt[2^l]{n}$ is transmitted and we start the linear search procedure. When y is located between k and $k + \sqrt[2^l]{n}$, (Fig. 3(e)), irrespectively of j's position, we have nothing to do; the $\sqrt[2^l]{n}$ subfile, to which we must recur, has been found. In the last case (Fig. 3(f)), both y and j lie after bucket $k + \sqrt[2^l]{n}$. If $j - c_a\sqrt[2^l]{n} < k + \sqrt[2^l]{n}$ holds, we linearly search the rest of the bcast from slot $k + \sqrt[2^l]{n}$. Else, we doze until bucket $j - c_a\sqrt{n}$ arrives. If the adjusted prediction is proven to be true, we execute the linear search; otherwise, we missed the bcast and linear searching is performed from "secure" bucket $k + \sqrt[2^l]{n}$.

The described protocol is presented in the appendix, while Figure 4 illustrates a searching instance, where we assume that $c_d = 1.2$, $c_a = c_x = 1$, and 40 is the key we search. We initially tune into bucket 8. Since $40 < 60$, we interpolate
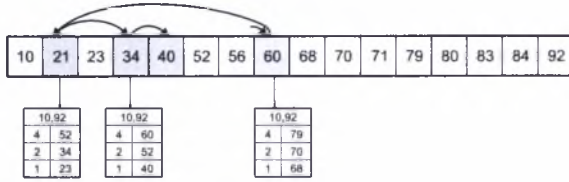
Figure 4: An instance of searching for item 40 with interpolation index.

and get $j = \left\lceil \frac{40-10}{92-10} \cdot 16 \right\rceil = 6$. Since $\frac{6}{\sqrt{16}} = 1.5 > 1.2$, we decide to switch into doze mode until the bucket $6 - 1 \cdot \sqrt{16} = 2$ arrives, where we have $21 < 40$, and, thus, we must continue linear searching in upcoming buckets. Since level 1 pointer (52) bounds 40, we recur in level 2. We apply once again interpolation which give us $j = 2 + \left\lceil \frac{40-21}{52-21} \cdot 4 \right\rceil = 5$. Because 40 is located after $2 + \sqrt{16} = 4$, we sleep until bucket 4 is transmitted, where we figure out that 40 belongs to the following bucket 5.

As far as tuning time is concerned, the following Lemma holds, whose proof is omitted due to space limitations:

LEMMA 2. *The average tuning time is* $O(\log \log n)$.

4. PERFORMANCE EVALUATION

This section provides a detailed study of the performance of the proposed index against the state-of-the-art scheme of exponential index [28, 29], using similar system parameters in order to conduct fair comparisons. The database size n ranges from 1000 to 1000000 items. Additionally, the database is characterized as small one when N between 1000 and 50000; otherwise, is considered as a big one. In this article, similar to [28, 29] we present the results for uniform access pattern; the conducted experiments for skewed (Zipfian) access patterns favor the interpolation index even more. We have also investigated different combinations of bucket capacity B, ranging thus the item size. In the sequel, we report the results for $B = 10$ and $B = 100$; intermediate values for B led to the same conclusions. Finally, the tuning time and access latency, are both measured in terms of number of buckets.

4.1 Tuning the Interpolation index

Our first experiment investigated the impact of parameters c_d and c_a on the performance of the interpolation index. Due to space limitations, we consider only the case $c_a = c_d = C$. Figure 5 depicts our findings when $N = 30\,000$, B is either 10 or 100, and C varies between 0 (i.e., the index 'blindly' trusts the interpolation estimation) and 1.5 (we are a bit conservative and adjust the estimation by going back 1.5 steps; we have arrived to analogous conclusions for various values of N, while being more conservative, that is, $C > 1.5$ proven to deteriorate the index performance.

Figure 5(a) shows that every value greater or equal to 0.3 is fine for bringing the average latency time to 50% of the beast. On the other hand, $C = 0.9$ appears to be the best value for achieving the overall best tuning time performance. This value is used for the rest of the experimental section, and as we will see, certainly beats the exponential index, proving that our proposal is a very easily configurable indexing scheme.

4.2 Comparison with the Exponential index

In this section, we compare our proposal to the exponential index. The investigation is twofold: we explore both the time as well as the space overhead of the two proposals.

4.2.1 Access latency and tuning time

Firstly, we explored the tuning time performance of both schemes. The comparison was performed on the basis of tuning the Interpolation air index to achieve the same access time with that achieved by the exponential index and then comparing their tuning time performance. The independent parameter was the number of broadcast items. The performance of the indexes was compared for both small and large databases, and for the case where the parameter r of exponential index was set to $r = 2$ and $r = 3$. The obtained results are depicted in Figures 6(a)-(d). Confirming the conclusions of [29], we observe that the tuning time of exponential index increases as the number of broadcast items gets larger. The increase is more steep for large number of items. On the contrary, the tuning time incurred by the Interpolation index is almost constant for small database sizes and increases moderately for large databases, as a result of its log log performance.

In Figures 7(a)-(c) we performed the reverse experiment: we kept the tuning time the same for both schemes and examined the access latency. We can see that our scheme achieves the same average access latency performance, irrespectively of the data base size, while the exponential index is slightly worst for small data sets. We refrained from showing the results for the case of large databases and with the

(a)

(b)
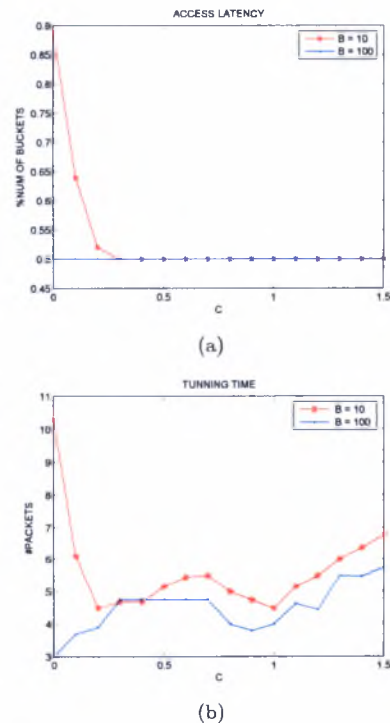
Figure 5: Investigation of the C parameter when $N = 30\,000$: (a) access latency; and (b) tuning time w.r.t. the C parameter.
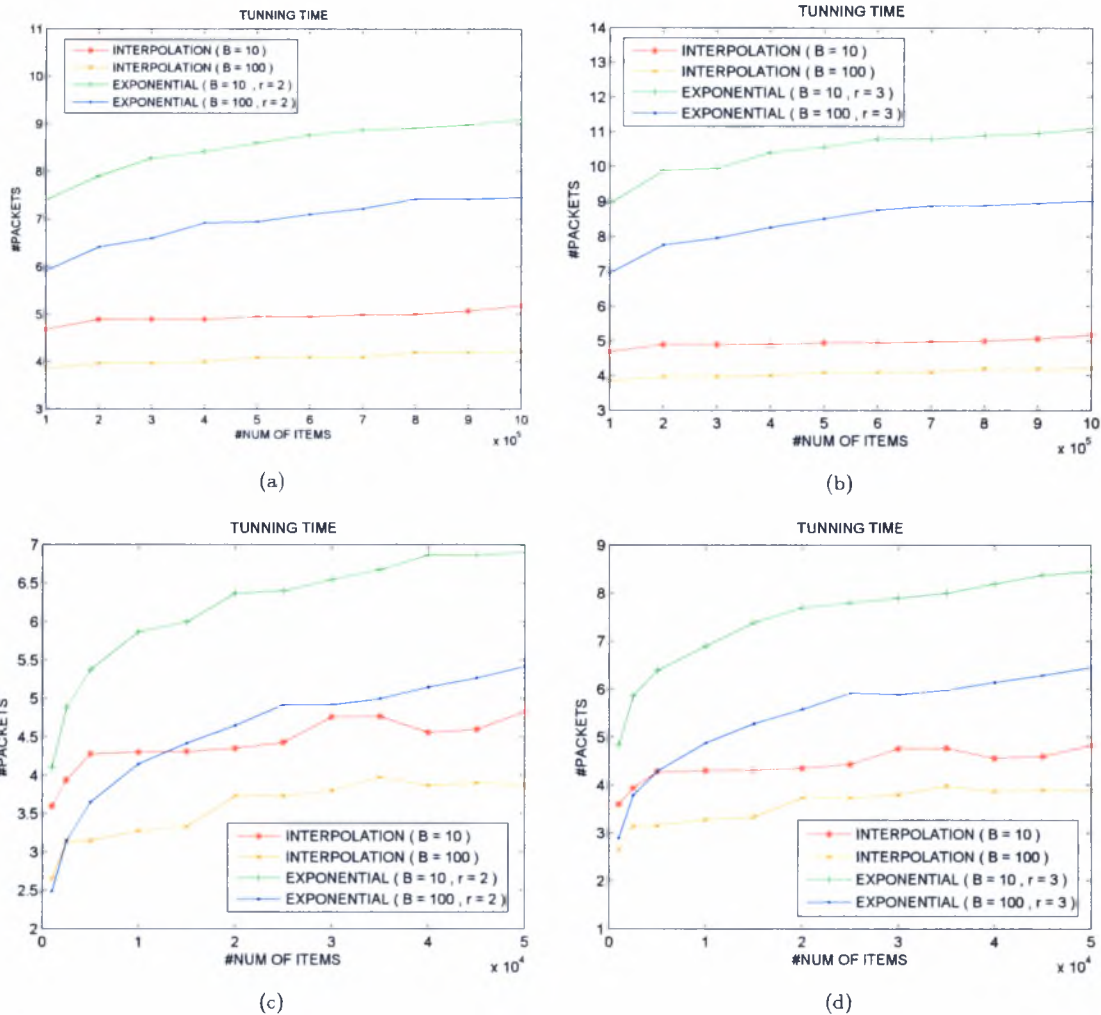
Figure 6: Average Tuning Time: (a) big data base with $r = 2$; (b) big data base with $r = 3$; (c) small data base with $r = 2$; and (d) small data base with $r = 3$.

exponential index's $r = 3$, because the results are completely analogous.

4.2.2 Index overhead investigation

In this final presented experiment, we show the results of the interrelation among the tuning time and the index overhead which is measured in "pointers" per transmitted bucket. Following the same policy as before, we performed the investigation by keeping constant for both competing indexes the first "quantity" and measuring the other, and vice versa. Firstly, we evaluated the average index overhead for the same tuning time; to achieve the same tuning time, we adjusted the index base r of the exponential index. The results are illustrated in Figure 8(a)-(b). It is obvious that on the average, the overhead of exponential index is four times larger than the respective of the interpolation index! This implies that the length of the broadcasting program generated by the exponential index is much larger than the program generated by the interpolation index.

Then, we forced both schemes to employ the same in-

dex space overhead and investigated the tuning time performance. Figures 9(a)-(b) confirm that interpolation air index undoubtedly outperforms exponential index by almost an order of magnitude.

5. DISCUSSION AND CONCLUSION

Modern wireless broadcasting systems, like Microsoft's SPOT technology or emerging applications of wireless sensor networks, involve the broadcasting of several thousands of items over wireless channels. Given the fact that mobile nodes of cellular systems or sensor nodes are energy-starving devices, it is mandatory that the access to the transmitted information is as energy-conserving as possible. This requirement calls for the deployment of distributed air indexes able to scale up to larger than ever numbers of items, and tunable so as to be able to tradeoff energy-conservation for access latency when needed.
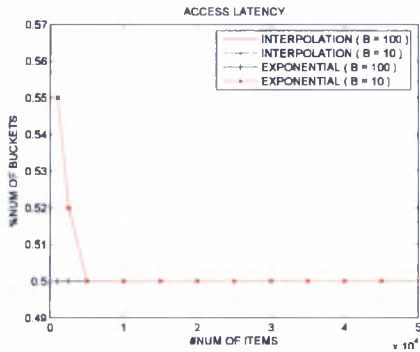
This article is motivated by the aforementioned requirement, and proposes the *interpolation air index*, a very easily configurable and efficient air indexing scheme. This scheme
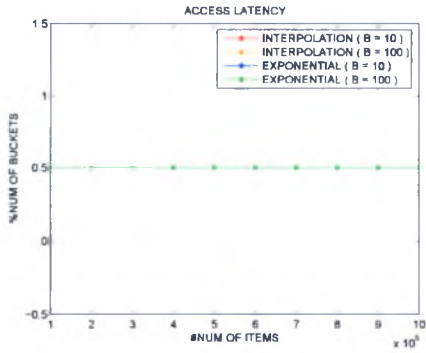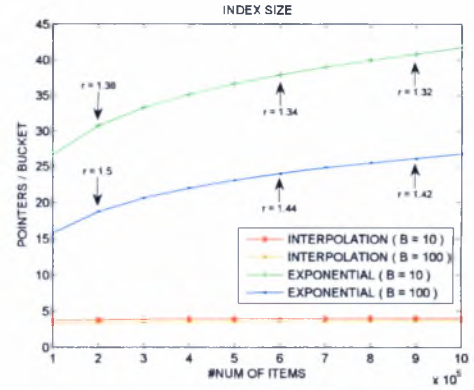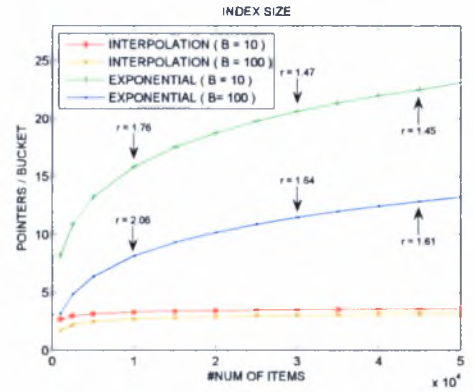
(a)
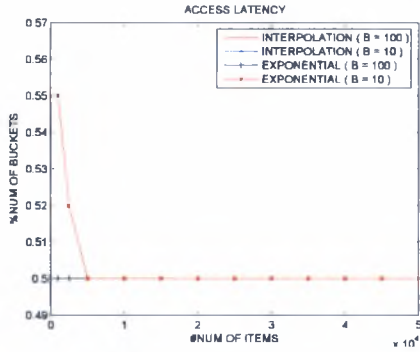
(b)

(c)

Figure 7: Average access latency: (a) big data base; (b) small data base with $r = 2$; and (c) small data base with $r = 3$.


(a)

(b)

Figure 8: Average index overhead when both schemes exhibit the same tuning time performance: (a) big data base; and (b) small data base.

There are a number of issues that are left for future work. Firstly, we plan to investigate how skewed data can be accommodated in our scheme. Secondly, it is very interesting to explore its performance in multi-channel data broadcast environments.

exhibits a linear structure, suiting the broadcast environment very well. The index space overhead is log-logarithmic per transmitted bucket, while the tuning time is log-logarithmically proportional to the broadcast size. Additionally, the access latency and tuning time of the interpolation index can be simply adjusted by a single parameter. To evaluate the suitability and behavior of the proposed novel air index, we investigated its performance against the state-of-the-art exponential index [29]. The experimental results attest that our index outperforms the exponential index both in tuning time and space overhead, while achieving the same access latency.

6. REFERENCES

[1] D. Amarmend, M. Aritsugi, and Y. Kanamori. An air index for data access over multiple wireless broadcast channels. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, page 135, 2006.

[2] M.-S. Chen, K.-L. Wu, and P. S. Yu. Optimizing index allocation for sequential data broadcasting in wireless mobile computing. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):161–173, 2003.

[3] D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[5] P. Desnoyers, D. Ganesan, and P. Shenoy. TSAR: A two tier sensor storage architecture using interval skip graphs. In *Proceedings of the ACM International*
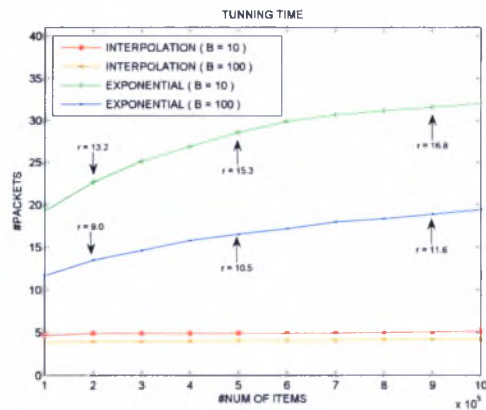
(a)
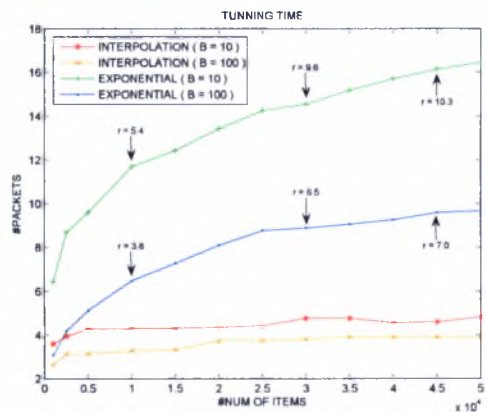
(b)

Figure 9: Average tuning time when both schemes exhibit the same index space overhead: (a) big data base; and (b) small data base.

Conference on Embedded Networked Sensor Systems (SenSys), pages 39–50, 2005.

[6] L. M. Feeney. Energy efficient communication in ad hoc wireless networks. In S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, editors, *Mobile Ad Hoc Networking*, pages 301–327. IEEE/Wiley, 2004.

[7] B. Greenstein, S. Ratnasamy, S. Shenker, R. Govindan, and D. Estrin. DIFS: A distributed index for features in sensor networks. *Ad Hoc Networks*, 1(2–3):333–349, 2003.

[8] C.-H. Hsu, G. Lee, and A. L. P. Chen. Index and data allocation on multiple broadcast channels considering data access frequencies. In *Proceedings of the Conference on Mobile Data Management (MDM)*, pages 87–93, 2002.

[9] Q. L. Hu, W.-C. Lee, and D. L. Lee. Indexing techniques for wireless data broadcast under data clustering and scheduling. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 351–358, 1999.

[10] Q. L. Hu, W.-C. Lee, and D. L. Lee. Power

conservative multi-attribute queries on data broadcast. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 157–166, 2000.

[11] Q. L. Hu, W.-C. Lee, and D. L. Lee. A hybrid index technique for power efficient data broadcast. *Distributed and Parallel Databases*, 9(2):151–177, 2001.

[12] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Power efficient filtering of data on air. In *Proceedings of the International Conference on Extending Data Base Technology (EDBT)*, pages 245–258, 1994.

[13] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air: Organization and access. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):353–372, 1997.

[14] S. Jung, B. Lee, and S. Pramanik. A tree-structured index allocation mathod with replication over multiple broadcast channels in wireless environments. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):311–325, 2005.

[15] W.-C. Lee and D. L. Lee. Using signature techniques for information filtering in wireless and mobile environments. *Distributed and Parallel Databases*, 4(3):205–227, 1996.

[16] W.-C. Lee and B. Zheng. DSI: A fully distributed spatial index for location-based wireless broadcast services. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 349–358, 2005.

[17] X. Li, Y.-J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 63–75, 2003.

[18] K. Mehlhorn. *Data Structures & Algorithms, I: Sorting and Searching*. Springer-Verlag, Berlin, 1984.

[19] A. Meka and A. K. Singh. DIST: A distributed spatio-temporal index structure for sensor networks. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 139–146, 2005.

[20] Y. Perl, A. Itai, and H. Avni. Interpolation search — A log log N search. *Communications of the ACM*, 21(7):550–553, 1978.

[21] Y. Perl and E. M. Reingold. Understanding the complexity of interpolation search. *Information Processing Letters*, 6(6):219–222, 1977.

[22] K. Prabhakara, K. Hua, and J. Oh. Multi-level multi-channel air cache designs for broadcasting in a mobile environment. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 167–176, 2000.

[23] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-centric storage in sensornets with GHT: A geographic hash table. *ACM Mobile Networks and Applications*, 8(4):427–442, 2003.

[24] A. Seifert and J.-J. Hung. FlexInd: A flexible and parameterizable air-indexing scheme for data broadcast systems. In *Proceedings of the International Conference on Extending Data Base Technology (EDBT)*, volume 3896 of *Lecture Notes on Computer*

Science, pages 902–920. Springer, 2006.

[25] N. Shivakumar and S. Venkatasubramanian. Efficient indexing for broadcast based wireless systems. *ACM/Baltzer Mobile Networks and Applications*, 1(4):433–446, 1996.

[26] K. L. Tan and J. X. Yu. Energy efficient filtering of nonuniform broadcast. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 520–527, 1996.

[27] M. A. Viredaz, L. S. Brakmo, and W. R. Hamburgen. Energy management on handheld devices. *ACM Queue*, 1(7):44–52, 2003.

[28] J. Xu, W.-C. Lee, and X. Tang. Exponential index: A parameterized distributed indexing scheme for data on air. In *Proceedings of the ACM/USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 153–164, 2004.

[29] J. Xu, W.-C. Lee, X. Tang, Q. Gao, and S. Li. An error-resilient and tunable distributed indexing scheme for wireless data broadcast. *IEEE Transactions on Knowledge and Data Engineering*, 18(3):392–404, 2006.

[30] Y. Yao, X. Tang, E.-P. Lim, and A. Sun. An energy-efficient and access latency optimized indexing scheme for wireless data broadcast. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1111–1124, 2006.

[31] B. Zheng, W.-C. Lee, and D. L. Lee. Spatial queries in wireless broadcast systems. *ACM/Kluwer Wireless Networks*, 10(6):723–736, 2004.

APPENDIX

AUXILIARY FUNCTION — SearchI

1. $\text{SEARCHI}(y, x, z, l)$
2. //*probing of interpolated value x;z is the 'retry' bucket*
3.   **if** ($x \mathrel{!}=$ current bucket)
4.   doze until bucket x arrives;
5.   **if** ($y < m_x$)
6.   switch to doze mode until bucket z arrives;
7.   **if** (($m_z \le y \le M_z$) || ($M_z \le y \le M_{z+1}$))
8.   //*y belongs to z or $z + 1$*
9.   search y among the data elements of bucket z
10.   or $z + 1$ and **exit**;
11.   **if** (($y == M_p$) && (p among indexed buckets))
12.   switch to doze mode until bucket p arrives;
13.   **return** the result of search and **exit**;
14. $\text{SEARCHL}(y, z, l)$;
15.   **else**
16.   **if** (($m_x \le y \le M_x$) || ($M_x \le y \le M_{x+1}$))
17.   //*y belongs to x or $x + 1$*
18.   search y among the data elements of bucket x
19.   or $x + 1$ and **exit**;
20.   **if** (($y == M_p$) && (p among indexed buckets))
21.   switch to doze mode until bucket p arrives;
22.   **return** the result of search and **exit**;
23. $\text{SEARCHL}(y, x, l)$;
24. **end of** SEARCHI

AUXILIARY FUNCTION — SearchL

1. $\text{SEARCHL}(y, x, l)$ // *linear search with $n^{\frac{1}{2^l}}$ jumps*
2. $b = x + n^{\frac{1}{2^l}}$;
3.   **while** ($M_b < y$)
4.   //*known from the index of the current bucket $b - n^{\frac{1}{2^l}}$*
5.   switch to doze mode until bucket b arrives;
6. $b = b + n^{\frac{1}{2^l}}$;
7. $\text{INTERPOLATIONSEARCH}(y, b - n^{\frac{1}{2^l}}, l + 1)$;
8. **end of** SEARCHL

CLIENT ACCESS PROTOCOL FOR INTERPOLATION INDEX

Algorithm INTERPOLATIONSEARCH(y, k, l)

Input: The key item y to be searched, k the current bucket the client tuned into, and l the level of recursion

Output: The data bucket r such that $y \in [m_r, M_r]$, and may contain y

1. **if** (($m_k \le y \le M_k$) || ($M_k \le y \le M_{k+1}$)) //$y \in \{k, k+1\}$
2.   search y among the data elements of bucket k or $k + 1$;
3.   **exit**;
4. **if** (($y == M_p$) && (p among indexed buckets))
5.   switch to doze mode until bucket p arrives;
6.   **return** the result of search and **exit**;
7. $j = (l > 1 ? \lceil n^{\frac{1}{2^{l-1}}} F_{[m_k, M_{k+n^2-l+1}]}(y) \rceil : \lceil n F_{[m_1, M_n]}(y) \rceil)$;
8. **if** ($j <= k$)
9.   **if** ($y < m_k$) //*this may happen only if $l = 1$*
10.   **if** ($\lceil \frac{j}{2\sqrt[l]{n}} \rceil \le c_d$)
11.   //*within heuristic distance parameter c_d*
12. $\text{SEARCHI}(y, 1, 1, l)$;//*we search from bucket 1*
13.   **else** //*we will try the estimation, in case of failure*
14.   //*we will resume on bucket 1*
15. $\text{SEARCHI}(y, j - c_a \sqrt[2^l]{n}, 1, l)$;
16.   **else** //*$y > M_k$, linear search from k*
17. $\text{SEARCHI}(y, k, k, l)$;
18. **else** //*$j > k$*
19.   **if** ($y < m_k$)
20.   **if** ($\lceil \frac{k}{2\sqrt[l]{n}} \rceil \le c_d$)
21.   //*within heuristic distance parameter c_d*
22. $\text{SEARCHI}(y, 1, 1, l)$;
23.   **else** //*we will try the estimation*
24. $\text{SEARCHI}(y, \max\{k - c_a \sqrt[2^l]{n}, 1\}, 1, l)$;
25.   **else** // $y > M_k$
26.   **if** ($M_k < y < M_{k + \sqrt[2^l]{n}}$) //*the right subfile is found*
27.   //*recursive searching*
28. $\text{INTERPOLATIONSEARCH}(y, k, l + 1)$;
29.   **else** // $y > M_{k + \sqrt[2^l]{n}}$
30.   **if** ($j \le k + \sqrt[2^l]{n}$) //*linear search from $k + \sqrt[2^l]{n}$*
31. $\text{SEARCHL}(y, k + \sqrt[2^l]{n}, l)$;
32.   **else** //$(j > k + \sqrt[2^l]{n}) \wedge (y > M_{k + \sqrt[2^l]{n}})$
33.   **if** ($j - c_d \sqrt[2^l]{n} > k + \sqrt[2^l]{n}$)
34.   //*we will try the estimation*
35. $\text{SEARCHI}(y, j - c_d \sqrt[2^l]{n}, k + \sqrt[2^l]{n}, l)$;
36.   **else** //*linear search from $k + \sqrt[2^l]{n}$*
37. $\text{SEARCHL}(y, k + \sqrt[2^l]{n}, l)$;
end of INTERPOLATIONSEARCH