Broadcast Protocols to Support Efficient Retrieval from Databases by Mobile Users

By Anindya Datta, *et al.*

Presented by Matt Miller

February 20, 2003

Outline

- Motivation
- Problem Statement
- Related Work
- Key Contributions
- Protocol Description
- Evaluation
  - Analytical
  - Simulation
- Potential Improvements
- Conclusions

Motivation

- Base stations have database items which mobile devices would like to read over a wireless channel. Communication is asymmetric.
- Item content changes frequently. Users want to continually acquire recent versions.
- Some items may be more popular.
- Users are mobile.
- Mobile devices may be energy-constrained.

Problem Statement

- Design a server protocol to determine which data items to include in the broadcast and for how long
- Design a client protocol which cooperates with the server protocol to allow retrieval of an item whenever it is broadcast while minimizing energy consumption

Related Work

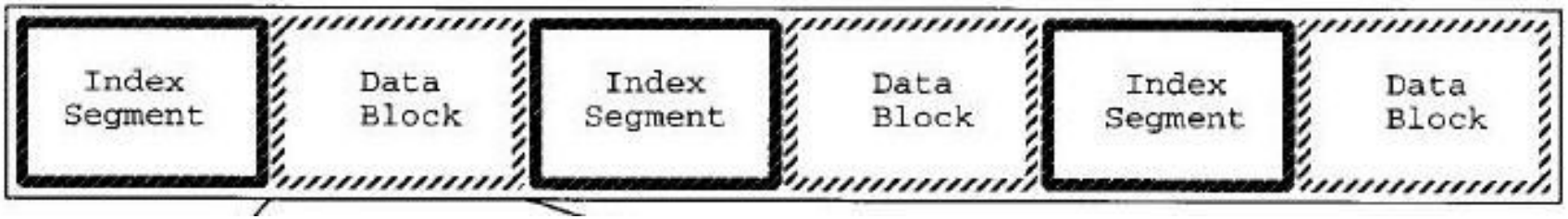
- Broadcast Organization
  - Given the content, how can it be organized efficiently (e.g., index structures)
  - Complementary to proposed work
- Broadcast Disks
  - Items broadcast at different frequencies to emulate disks with varying access times
  - Content constructed based on known access probabilities
  - Content set is static

Key Contributions

- Creates a mixed mode of access
- Alters broadcast content to take advantage of popularity while avoiding excessive starvation
- Integrates client-side caching
- Analytical models for protocol
- Protocol simulation with detailed energy model

Server Broadcast Structure

- Uses a (1,*m*) organization proposed by Imielinkski *et al*.
- *m* index segments are uniformly distributed in the current broadcast. Each segment is identical and tells what data is in the broadcast and the locations.
- A data block is located between each pair of index segments. This implies *m* data blocks per broadcast.
- Index segments implemented with a B-Tree structure for efficient searching.



Server Broadcast Structure (2)

- Information in each bucket:
  - Offset to next index segment
  - Offset to end of broadcast
  - Indication of whether data was modified since last broadcast
  - Offset to next bucket of DC
  - When the DC is scheduled to be dropped from the broadcast (EDT)
- Information in index buckets:
  - Key value (specifies DC)
  - Offset to first bucket of DC
  - Offset to first dirty bucket of DC
  - EDT

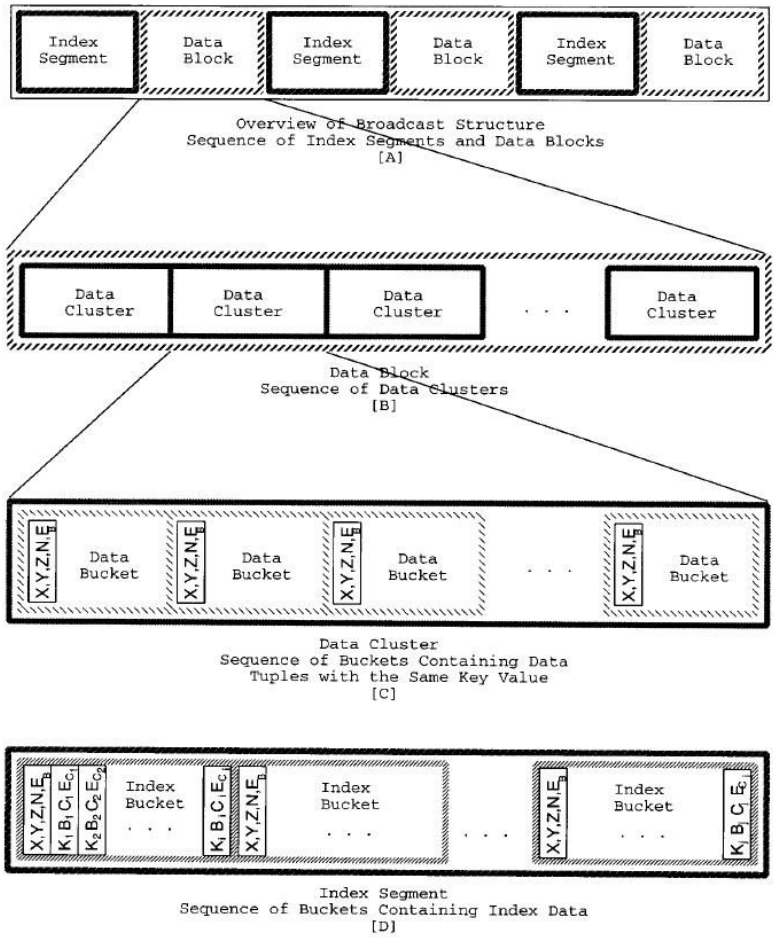


Fig. 3. Broadcast structure.

Broadcast Content

Considers two possibilities:

1. Constant Broadcast Size (CBS): Every broadcast is fixed length. If too few items, dead air is broadcast. If too many items, contention algorithm determines set.
2. Variable Broadcast Size (VBS): Every requested item is included.

CBS Contention

- Server includes items with the highest priorities.
- Items included in order of popularity.
- Priority function

  PF: Number of clients are currently interested in an item

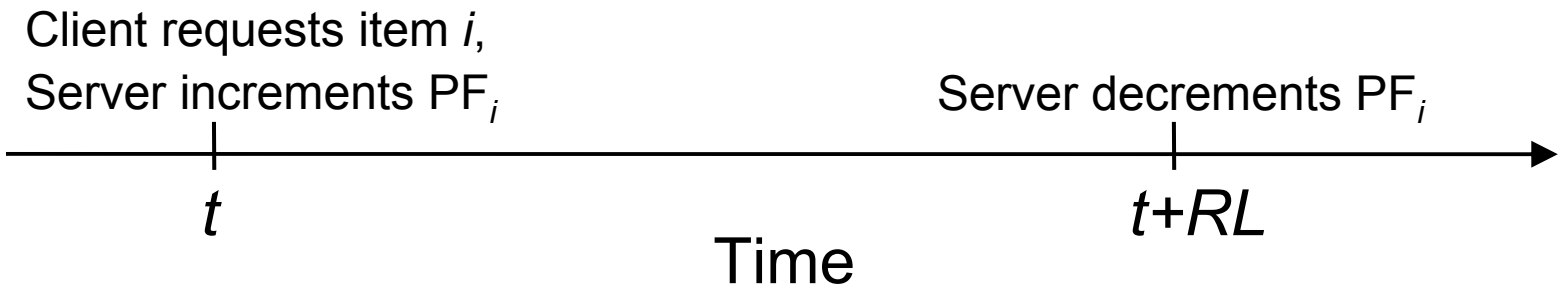
  IF: Number of consecutive broadcasts an item has not been included

  ASF: How many broadcasts the average wait time (AWC) for interested clients has exceeded the specified desired wait time (DWC)

$$Priority = IF^{ASF} \times PF$$

Client Protocol

- On initial probe, if the desired item is will not be in the next broadcast, make a request. Server assumes client will remain in cell for *RL* time units.
- If a client's desired data item appears in consecutive broadcasts, only download the buckets which were modified.
- If the client cannot download the item in consecutive broadcasts, the entire item must be downloaded.



Metrics

- Access Time (AT): Time from when a request is first made until the download of the item is complete
- Tuning Time (TT): Time a client actively listens to the broadcast
- Normalized Energy Expenditure (NEE): Amount of energy clients use per data downloaded. Only used in simulations.

Analysis

- Does not consider IF to make analysis mathematically tractable
- Assumes clients are only interested in one item (DCI)
- Calculates probabilities of various scenarios:
  - Does download start with index or data?
  - Is the DCI in the current broadcast?
  - If so, did the client miss the DCI?

Analysis (2)

- From the probabilities, the expected TT and AT are calculated.
- Basic pattern for TT:
  - Time to read first bucket broadcast on initial probe.
  - Time to do a logarithmic search on index segment. Multiply this by the number of broadcasts until the DCI is included.
  - Time to download the DCI.

Analysis (3)

- Basic pattern for AT:
  - Time from initial probe until the first index segment
  - Time until either DCI or next broadcast
  - Time for each broadcast the DCI is not included
  - Time to download DCI

Analysis Prediction

- CBS: Both AT and TT, relatively constant with some decrease at higher loads due to redundancy.
- VBS: TT will increase due to larger index segments. AT increases significantly over CBS. Increase is more gradual at high loads due to redundancy.

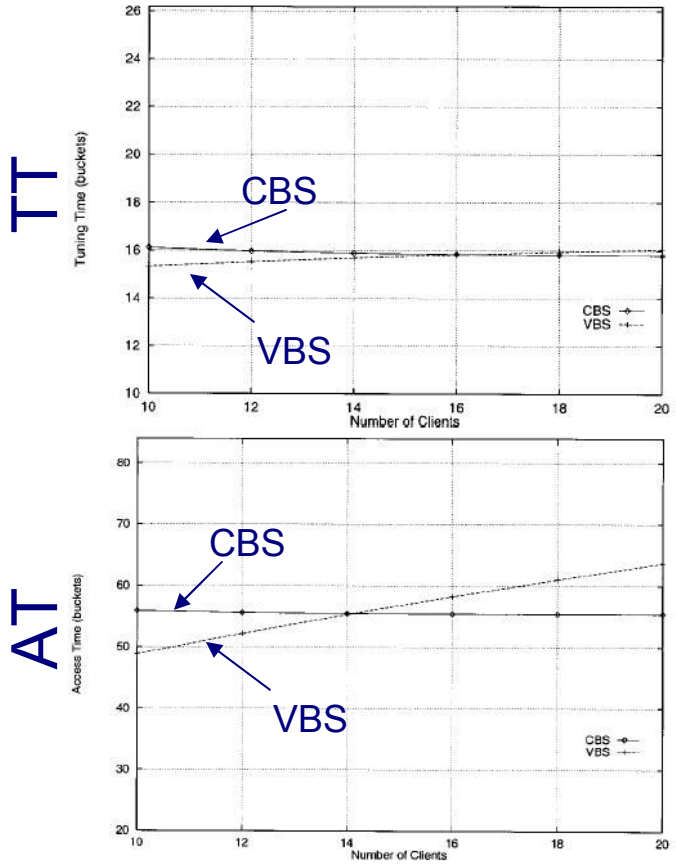


Fig. 5. Experimental results:(a) TT for VBS and CBS; (b) AT for VBS and CBS.

Number of Clients

Simulation Setup

- Considers IF in priority computation
- Items are hot or cold to indicate popularity. Data separated for these two classes.
- Measures energy from four sources:
  1. Hard Drive: R/W, idle, sleep
  2. CPU: on, sleep
  3. Display: on, off
  4. Wireless card: transmit, receive, sleep

Simulation Setup (2)

- To test client-side modification, compares CBS versus a protocol which always downloads the entire DCI on consecutive broadcasts (referred to as IVB)
- To calculate NEE the following equation is averaged for *L* tracked clients:

$$\frac{\textit{total energy consumed}}{(\textit{size of DCI} \times \textit{number of times downloaded})}$$

Simulation Results

- Client-side caching mainly helps when the load is low and the item is hot
- Effects of client caching are negligible compared to choice of server content
- VBS always better at low loads (no dead air) and if no locality is present (every request filled each broadcast)
- CBS is better when locality is present and load is moderate to high because hot items will appear more frequently at the expense of cold items

Simulation Results (2)

- AT is a better measure of NEE than TT because idle time dominates. Therefore, only the denominator causes significant differences to the metric and it is a function of the latency between DCI downloads.
- Changing the broadcast size, client mobility, database size and item update rate will shift and scale the basic trends by affecting inclusion contention, load, redundancy and effectiveness of caching.

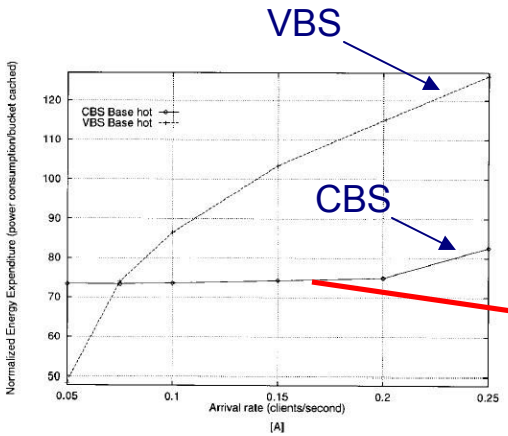
Simulation Results (3)

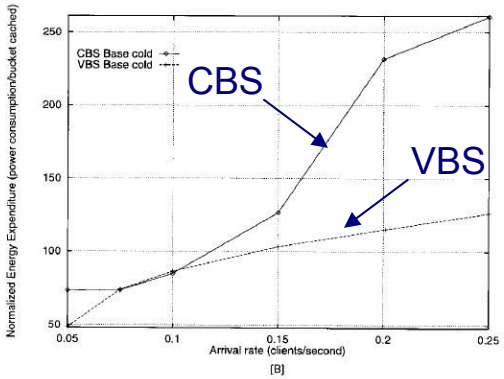
- VBS
  - Same for hot and cold items
  - Always lower NEE at low loads because no dead air
  - Slope will decrease at high loads do to increased redundancy
- CBS
  - Hot items will have constant NEE until cold items can occasionally replace hot items. At high loads, the NEE will increase rapidly because many cold items exceed their DWC.
  - After constant NEE, cold items show rapid increase since there is much contention among the cold items. The slope will become more gradual at high loads as cold items exceeding their DWC will contend more with hot items.

Simulation Results (4)

HOT NEE

COLD NEE





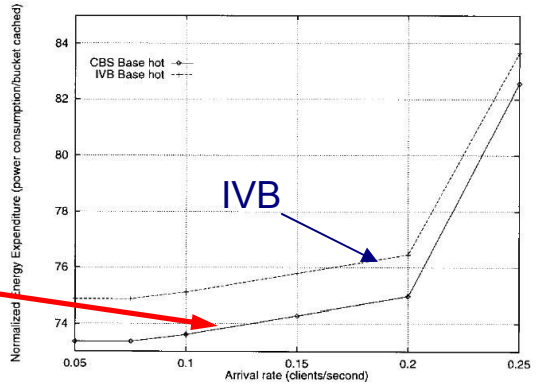


Fig. 9. Experimental results: (a) NEE curves for CBS and VBS hot clients; (b) NEE curves for CBS and VBS cold clients.

Client Arrival Rate

Client Arrival Rate

Potential Improvements

- Protocol
  - Integrate security for subscriptions
  - More analytical justification for priority computation
- Potentially Influential Factors
  - Uplink contention
  - Overhead of switching the WLAN card on/off frequently
  - NEE is not a good metric if a device's energy is at critical level
- Simulation
  - More realistic popularity model (e.g., Zipf)
  - More realistic energy model
  - The effects of a large variance for RL (simulation variance is less than 0.5% of the mean)
  - Measure the initial latency of a user request

Conclusions

- Neither protocol is strictly better
  - If locality is present, CBS protocol is usually better
  - If load is low, VBS is always better
- The effects of client-side caching are insignificant compared to the choice of broadcast content.
- AT is a better metric than TT for estimating NEE because idle time dominates energy and latency between downloads becomes the significant factor.

Bonus Slides

Analytical Notation

Table I. Notation

Notation	Meaning
d	size of data to broadcast (in items)
S	selectivity of a data item (in buckets)
n	capacity of a bucket (tuples per bucket)
m	number of index segments or data blocks per broadcast
B	size of data block (buckets per block) = d/nm
I	size of index segment (in buckets) = $2d/Sn$
T_B	size of the broadcast in CBS = $m(I + B)$
b	number of DCI = d/Sn
C_{CBS}	capacity of broadcast in CBS (in DCI)
l	size of an index segment (in buckets) = $\log_{n+1} I$
L_1	expected distance to DCI from the end of index segment (buckets)
L_2	expected distance to next broadcast from end of index segment (buckets)
p_h	probability that a client requests a *hot* item
p_c	probability that a client requests a *cold* item
$p_{h,\,in}$	probability that a *hot* item is in broadcast
$p_{c,\,in}$	probability that a *cold* item is in broadcast
P_{in}	probability that an item is in broadcast
h	fraction of *hot* items in the d
c	fraction of *cold* items in the d
p_0	probability that a client does not request any DCI
k	total number of DCIs in the database

Table II. Probabilistic Events

Event	Meaning
M	DCI is missed completely
$\bar{M}$	DCI is downloaded completely (i.e., not missed at all)
PM	DCI is missed partially
D_i	DCI is in the i^{th} data block
TI_i	tune in to i^{th} index segment first
TD	tune in to a data block first
TI	tune in to an index segment first
PH	client requests a *hot* item
PC	client requests a *cold* item

Table III. Event Probabilities

Events	Probabilities
$P\{\overline{M},\ \overline{TI}\}$	$(m - 1)/(2m + S)$
$P\{\overline{M},\ \overline{TD}\}$	$((m - 1)/2m)(S/(2m + S))$
$P\{M,\ TI\}$	$(m + 1)/(2m + S)$
$P\{M,\ TD\}$	$(((m - 1)/2m) + ((B - S)/mB))S/(2m + S)$
$P\{PM\}$	$(S/mB)S/(2m + S)$

Simulation Parameters

Table V. Broadcast and Cell Parameters in the Simulation

Notation	Baseline Value
DatabaseSize	3000 items
MeanRecordsPerItem	150 records
StdRecordsPerItem	6 records
MinRecordsPerItem	3 records
MaxRecordsPerItem	300 records
DesiredWaitCount	3 broadcasts
PercentHot	20%
PercentChooseHot	80%
DataRecordSize	40 bytes
IndexRecordSize	5 bytes
DataHeaderSize	8 bytes
IndexHeaderSize	8 bytes
BucketSize	128 bytes
IndexSegmentsPerBroadcast	4
BroadcastSize	1600 seconds
BucketsPerSecond	18.87 seconds
MeanChangeFrac	0.02
StdChangeFrac	0.001
MinChangeFrac	0
MaxChangeFrac	1
TransmissionRate	19.2 kbps
ArrivRate	variable
MeanResidenceLatency	8000 seconds
StdResidenceLatency	26.67 seconds
IdlePeriod	30 seconds
SpinUp	2 seconds

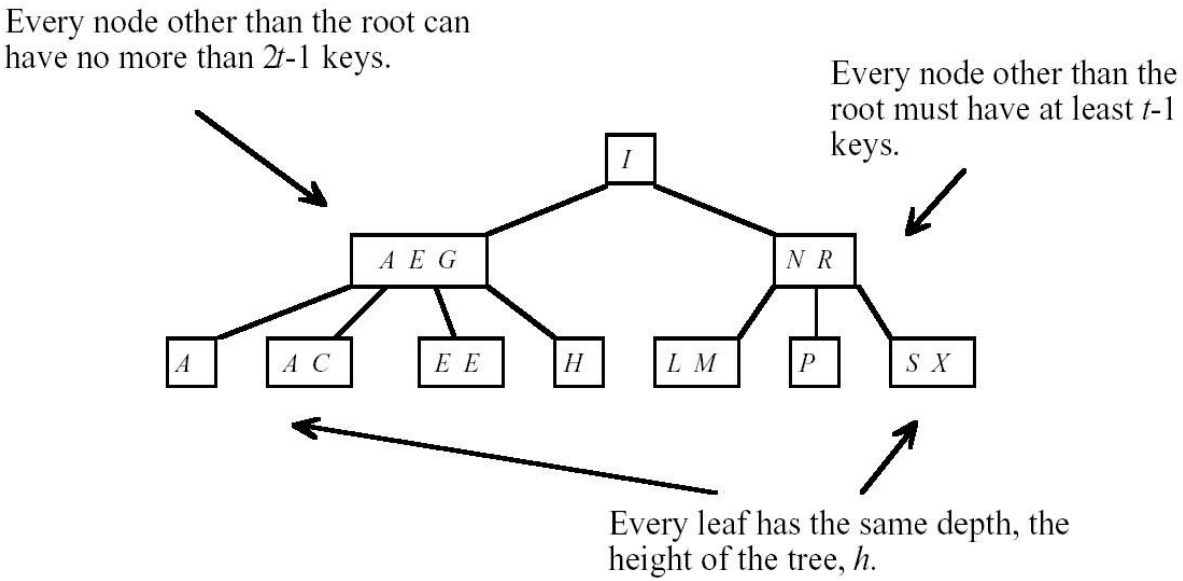
Table IV. Component States and Power Consumption

Component	State	Rate of Consumption (in watts)
CPU	Active	0.25
CPU	Sleep	0.00005
Disk	Write	0.95
Disk	Idle	0.65
Disk	Sleep	0.015
Display	Active	2.5
Display	Off	0
MDC	Transmit	0.4 – 0.6
MDC	Receive	0.2
MDC	Sleep	0.1

Energy Consumption Rates (Watts)

- CPU
  - Active: 0.25
  - Sleep: 0.00005
- Hard Drive
  - R/W: 0.95
  - Idle: 0.65
  - Sleep 0.015
- Display
  - On: 2.5
  - Off: 0.0
- WLAN Card
  - Transmit: 0.4
  - Receive: 0.2
  - Sleep: 0.1
- Current WLAN Spec
  - Transmit: 1.4
  - Receive: 1.0
  - Idle: 0.83
  - Sleep: 0.13

B-Tree Structure



- Every node (other than root) has between *t-1* and *2t-1* keys
- A node with *n* keys must have *n+1* children

CBS Contention Detailed

- Server maintains counter, PF, of how many clients have requested item within specified time limit (RL).
- Server maintains counter, IF, of how many consecutive broadcasts a requested item has not been included. This counter is calculated with respect to requests made in the last RL time units. It has a minimum value of one.

CBS Contention Detailed (2)

- For IF, the server only needs to maintain the timestamp of the first request made in each broadcast period since priority computation is done at the beginning of a broadcast and two requests arriving in the same broadcast are estimated to expire in the same broadcast.
- It only has to maintain this timestamp for each previous broadcast for which it estimates the requesting client has not left.

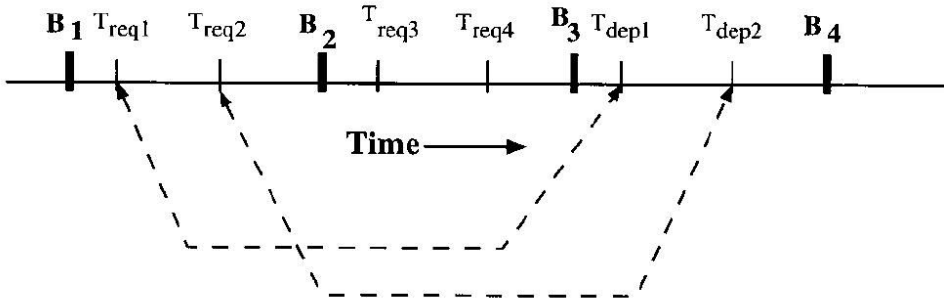


Fig. 4. Computation of the ignore factor.

CBS Contention Detailed (3)

- An adaptive exponential scaling factor, ASF, is initialized to one and incremented for each broadcast the AWC is greater than DWC for an item.
- Priority is then computed:

$$Priority = IF^{ASF} \times PF$$

CBS Contention Detailed (4)

- Intuitively, PF will dominate when all items are can be included every couple intervals. In this case, popular items will always be included and less popular items will alternate for inclusion.
- The IF will allow items to gain priority over more popular items when it has been passed over multiple times.
- The ASF term will force the ignoring of an item to dominate quickly when clients are waiting longer than the threshold.