Adaptive indexing for relational keys

Goetz Graefe
and
Harumi Kuno
To index, or not to index …

If you build the index, how many times does the index need to be used to justify the cost of creation?

This ratio changes with workload, data, and system configuration.
Load versus query performance

- Multiple indexes
- No indexes or statistics
- Adaptive indexing

Choose!
Database Cracking

• **Cracking the database store.** Martin L. Kersten, Stefan Manegold. CIDR 2005.

• **Database Cracking.** Stratos Idreos, Martin L. Kersten, Stefan Manegold. CIDR 2007.

• **Updating a cracked database.** Stratos Idreos, Martin L. Kersten, Stefan Manegold. SIGMOD 2007.

• **Self-organizing tuple reconstruction in column stores.** Stratos Idreos, Martin Kersten, Stefan Manegold. SIGMOD 2009.
Database Cracking

- **Like** an adaptive, incremental, in-memory quicksort
- Focus on active key ranges, e.g., ‘d’–‘m’

\[
\text{hbncoyulzqutgjwvdokimreapxsafsi}
\]

where ... between ‘d’ and ‘i’

\[
\text{bcaa,hegdiefi,noyulzqutjwvokmrpxs}
\]

where ... between ‘f’, ‘m’

\[
\text{bcaa,ede,hgifi,ljkm,noyuzqutwvorpxs}
\]
Adaptive Merging

• “Self-selecting, self-tuning, incrementally optimized database indexes” (EDBT ’10): demonstrates how to create and incrementally refine indexes as a side effect of processing range queries.

• “Adaptive indexing for relational keys” (SMDB ’10): extends adaptive merging approach to work with point (equality) queries.
Adaptive Merging

• **Like** an adaptive, incremental, external merge sort

```
hbnecoyulzqutgjwvdokimreapxafsi
```

where ... between ‘d’ and ‘i’

```
bcehnouy gjlqtuwz deikomrv aafipsx
```

where ... between ‘f’, ‘m’

```
bcenoyu qtuwz deorv aapsx fghiiijklm
```
What about equality queries?

Thus far, adaptive indexing research has focused on range queries

| Range queries | Metrics | Non-keys | "<", "between"

... but many queries use equality predicates.

| Equality queries | Dimensions | Keys | "=", "in"

For example: Which lineitems belong to orders whose zip codes fall between 90000 and 93510?
How are equality queries different?

- Range of 1 key $\rightarrow$ more queries needed to refine index on focus area.
- Obvious focus area may not exist.

Question: will it impact the number of queries needed to converge on a fully-refined index for the keys of interest to a workload?
Recommended techniques

• Both cracking and merging
  – Round query ranges

• Database cracking
  – Put keys in own partitions
  – Sort minimal partitions

• Adaptive merging
  – Partitioned B-trees
  – Merged indexes
  – Hot partitions
Evaluation

Simulation

$10^7$ records with distinct values

$10^5$ records can fit in workspace

Workload of random key queries
Record comparisons

Adaptive merging (with partitioned B-tree):

- Each record in 17-18 comparisons during run generation; 6-7 comparisons when merged.
- Each record copied and moved once during run generation, once during merging

Database cracking:

- Many fewer comparisons for initial query: each query like a single step of quicksort.
- Many more queries needed to converge.
Records accessed

Metric: number of records accessed > number of records accessed in presence of a full index

– Assumption: data on disk
– Overhead of index refinement
Impact of keys on convergence

Compare three methods:

- Database cracking (original)
- Database cracking (improved for key queries)
- Adaptive merging

Metric: records accessed above and beyond number of records accessed used when full index exists and is used.
25K queries, $10^7$ distinct key values

Records unnecessarily touched by scan

Database cracking

Improved database cracking

Adaptive merging

Full index = no overhead
Master index with distinct keys (e.g., orders with zip codes in LA County)
Detail index with 99 records per key (e.g., line items for orders from LA)

Database cracking

Improved database cracking

Adaptive merging
Adaptive merging / master-detail clustering (100 records per key)
Summary / Ongoing efforts

• All three methods apply to non-key columns (e.g., measures) and range predicates as well as key columns (e.g., dimensions) and equality predicates
• Rounding boundary keys (optional for range queries) seems required for equality queries
• Adaptive merging for key columns enables self-tuning and self-managing master-detail clustering
• Stay tuned for: collaborative exploration of cracking and merging with Stratos Idreos and Stefan Manegold of CWI 😊
Why index adaptively?

What if the workload changes every 1M queries?
What if most queries focus on a certain key range?
What if we need fast queries upon newly-loaded data?
What if we never access most of the data?
What if we don’t fully understand our workload?

... plus, adaptive indexing complements automatic tuning wizards, physical design, index tuning