Database Architectures for Big Data Exploration

Stratos Idreos
Every two days we create as much data as much we did from dawn of humanity to 2003

[Eric Schmidt]
make it easy to turn data into knowledge

2.5 exabytes

daily data

years

2012

Every two days we create as much data as much we did from dawn of humanity to 2003

[Eric Schmidt]

[IBMbigdata]

[Eric Schmidt]

[Domop]
Big Data V’s

- volume
- velocity
- variety
- veracity

data exploration

not always sure what we are looking for (until we find it)
today

what is an index?

wanted
Data Scientists
today

what is an index?

WANTED
Data Scientists

tomorrow
everybody will need to be a “data scientist”
it is time for a paradigm shift in how we design databases systems
database systems great...
declarative processing, back-end to numerous apps
database systems great...
declarative processing, back-end to numerous apps

but databases have become too heavy and blind!
database systems great...
declarative processing, back-end to numerous apps

but databases have become too heavy and blind!

timeline
database systems great...
declarative processing, back-end to numerous apps

but databases have become too heavy and blind!
**database systems great...**
declarative processing, back-end to numerous apps

**but databases have become too heavy and blind!**
**database systems great...**
declarative processing, back-end to numerous apps

**but databases have become too heavy and blind!**
but databases have become too heavy and blind!

expert users - idle time - workload knowledge
data systems tailored for data exploration
no workload knowledge

no installation steps

data systems tailored for data exploration
no workload knowledge

no installation steps

minimize data-to-query time

data systems tailored for data exploration
3 ideas

adaptive indexing

adaptive loading

dbTouch

Martin Kersten, Stefan Manegold, Felix Halim, Panagiotis Karras, Roland Yap, Goetz Graefe, Harumi Kuno, Eleni Petraki, Themis Palpanas, Kostas Zoumpatianos, Anastasia Ailamaki, Ioannis Alagiannis, Renata Borovica, Miguel Branco, Ryan Johnson, Erietta Liarou
3 ideas

- adaptive indexing
- adaptive loading
- dbTouch

Martin Kersten, Stefan Manegold, Felix Halim, Panagiotis Karras, Roland Yap, Goetz Graefe, Harumi Kuno, Eleni Petraki, Themis Palpanas, Kostas Zoumpatianos, Anastasia Ailamaki, Ioannis Alagiannis, Renata Borovica, Miguel Branco, Ryan Johnson, Erietta Liarou
indexing

load  tune  query

\textit{tune} = create proper indices offline

performance 10-100X
indexing

\[ \text{tune} = \text{create proper indices offline} \]

\[ \text{performance 10-100X} \]

\[ \text{but it depends on workload!} \]

\[ \text{which indices to build?} \]
\[ \text{on which data parts?} \]
\[ \text{and when to build them?} \]
sample workload
load \hspace{1cm} \textbf{tune} \hspace{1cm} \textbf{query}

sample workload \hspace{1cm} analyze

timeline
load       tune       query

sample workload       analyze       create indices

timeline
timeline

load  tune  query

sample workload  analyze  create indices  query
complex and time consuming process
human administrators + auto-tuning tools

sample workload  analyze  create indices  query

timeline

complex and time consuming process
Big Data V’s

- volume
- velocity
- variety
- veracity

what can go wrong?

- not enough space to index all data
- not enough idle time to finish proper tuning
- by the time we finish tuning, the workload changes
- not enough money - energy - resources
what can go wrong?

- not enough space to index all data
- not enough idle time to finish proper tuning
- by the time we finish tuning, the workload changes
- not enough money - energy - resources
database cracking
Database cracking

- Idle time
- Workload knowledge
- External tools
- Human control
database cracking

auto-tuning database kernels
incremental, adaptive, partial indexing

idle time
workload knowledge
external tools
human control
database cracking

auto-tuning database kernels
incremental, adaptive, partial indexing

idle time
workload knowledge
external tools
human control
database cracking

auto-tuning database kernels
incremental, adaptive, partial indexing

idle time
workload knowledge
external tools
human control
database cracking

auto-tuning database kernels
incremental, adaptive, partial indexing

every query is treated as an advice on how data should be stored
column-store database
a fixed-width and dense array per attribute
column-store database

a fixed-width and dense array per attribute
full indexing example

Q1:
select R.A
from R
where R.A>10
and R.A<14
full indexing example

Q1:
select R.A
from R
where R.A>10
and R.A<14
full indexing example

Q1: select R.A from R where R.A>10 and R.A<14

column A

sort

1 2
3 4
6 7
8 9
11 12
13
14
16
19
full indexing example

Q1: select R.A from R where R.A > 10 and R.A < 14
full indexing example

Q1: select R.A from R where R.A>10 and R.A<14

column A

sort

binary search

result
full indexing example

Q1: select R.A from R where R.A>10 and R.A<14

column A

<table>
<thead>
<tr>
<th>13</th>
<th>16</th>
<th>12</th>
<th>19</th>
<th>14</th>
<th>11</th>
<th>8</th>
<th>6</th>
</tr>
</thead>
</table>

sort

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
</table>

binary search

<table>
<thead>
<tr>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>16</th>
<th>19</th>
</tr>
</thead>
</table>

result

time + knowledge
cracking example

Q1:
select R.A
from R
where R.A>10
    and R.A<14
cracking example

Q1:
select R.A from R
where R.A>10 and R.A<14

column A

| 13 | 16 | 4 | 9 | 2 | 12 | 7 | 1 | 19 | 3 | 14 | 11 | 8 | 6 |

piece1: A<=10

| 4 | 9 | 2 | 7 | 1 | 3 | 8 | 6 | 13 | 12 | 11 | 16 | 19 | 14 |
cracking example

Q1: select R.A from R where R.A > 10 and R.A < 14

column A

piece1: A <= 10

piece2: 10 < A < 14
cracking example

Q1:
select R.A from R
where R.A > 10
and R.A < 14
cracking example

Q1:
select R.A
from R
where R.A>10
and R.A<14

column A

<table>
<thead>
<tr>
<th>13</th>
<th>16</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- piece1: A<=10
- piece2: 10<A<14
- piece3: A>=14

result: 16
Q1: select R.A from R where R.A > 10 and R.A < 14

Q2: gain knowledge on how data is organized
gain knowledge on how data is organized

column A

Q1: select R.A from R where R.A>10 and R.A<14

dynamically/on-the-fly within the select-operator
cracking example

Q1:
select R.A
from R
where R.A > 10
and R.A < 14

Q2:
select R.A
from R
where R.A > 7
and R.A <= 16

dynamically/on-the-fly within the select-operator
cracking example

Q1:
select R.A
from R
where R.A > 10
and R.A < 14

dynamically/on-the-fly within the select-operator

Q2:
select R.A
from R
where R.A > 7
and R.A <= 16
### Q1:
select R.A from R where R.A > 10 and R.A < 14

### Q2:
select R.A from R where R.A > 7 and R.A <= 16

**column A**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>14</td>
</tr>
</tbody>
</table>

- **piece1:** A <= 10
- **piece2:** 10 < A <= 14
- **piece3:** A >= 14

**dynamically/on-the-fly within the select-operator**
cracking example

Q1:
select R.A
from R
where R.A > 10
and R.A < 14

Q2:
select R.A
from R
where R.A > 7
and R.A <= 16

dynamically/on-the-fly within the select-operator
cracking example

<table>
<thead>
<tr>
<th>column A</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 16 4 9 2 7 1 3 8 6</td>
</tr>
</tbody>
</table>

Q1:
select R.A  
from R  
where R.A>10  
and R.A<14

dynamically/on-the-fly within the select-operator

Q2:
select R.A  
from R  
where R.A>7  
and R.A<=16
cracking example

Q1:
select R.A
from R
where R.A > 10
and R.A < 14

Q2:
select R.A
from R
where R.A > 7
and R.A <= 16

dynamically/on-the-fly within the select-operator
Q1: 
select R.A 
from R 
where R.A > 10 
and R.A < 14

Q2: 
select R.A 
from R 
where R.A > 7 
and R.A <= 16

dynamically/on-the-fly within the select-operator
select [15,55]
select [15,55]
select [15,55]
select [15,55]

10  20  30  40  50  60

select [15,55]
touch at most two pieces at a time

pieces become smaller and smaller

select [15,55]
implemented in \textit{monetdb}

open-source column-store

database kernel

optimizer
reconstruct
update
select
join
aggr

code footprint
monetdb 2M
continuous adaptation

set-up

100K random selections
random selectivity
random value ranges
in a 10 million integer column

Response time (secs)

Query sequence (x1000)
continuous adaptation

set-up
100K random selections
random selectivity
random value ranges
in a 10 million integer column

almost no
initialization overhead
continuous adaptation

set-up
100K random selections
random selectivity
random value ranges
in a 10 million integer column

almost no
initialization overhead

continuous improvement
continuous adaptation

set-up

100K random selections
random selectivity
random value ranges
in a 10 million integer column

almost no
initialization overhead

continuous improvement

Response time (secs)

Query sequence (x1000)
continuous adaptation

set-up
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column
continuous adaptation

set-up

10K random selections
selectivity 10%
random value ranges
in a 30 million integer column
Continuous adaptation

Set-up
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column

10K queries later,
Full Index still has not
amortized the initialization costs
traditional databases
monolithic/full indexing

workload analysis
index building
query processing

offline indexing

online indexing
traditional databases
monolithic/full indexing

offline indexing
workload analysis
index building
query processing

online indexing

database cracking
partial/adaptive/continuous indexing

adaptive indexing
<table>
<thead>
<tr>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
select R.A from R where R.A > 10 and R.A < 14
select R.A from R where R.A > 10 and R.A < 14

select max(R.A), max(R.B), max(S.A), max(S.B) from R, S
where v1 < R.C < v2 and v3 < R.D < v4
and v5 < R.E < v6 and k1 < S.C < k2 and k3 < S.D < k4 and k5 < S.E < k6
and R.F = S.F
select R.A from R where R.A > 10 and R.A < 14

select max(R.A), max(R.B), max(S.A), max(S.B) from R, S
where v1 < R.C < v2 and v3 < R.D < v4
and v5 < R.E < v6 and k1 < S.C < k2 and k3 < S.D < k4 and k5 < S.E < k6
and R.F = S.F
cracking tangram

**base data**

**As queries arrive...**

<table>
<thead>
<tr>
<th>Table 1</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
cracking tangram

**base data**

**table 1**

| A | B | C | D |

**table 2**

| A | B | C | D |

**as queries arrive...**

**table 1**

| A | B | C | D |

**table 2**

| A | B | C | D |
cracking tangram

**base data**

**table 1**

A
B
C
D

**table 2**

A
B
C
D

**as queries arrive...**

**table 1**

A
B
C
D

**table 2**

A
B
C
D

*partial materialization*
cracking tangram

**base data**

- table 1
  - A B C D

- table 2
  - A B C D

**as queries arrive...**

- table 1
  - A B C D

- table 2
  - A B C D

**partial materialization**

**partial indexing**
cracking tangram

base data

as queries arrive...

partial materialization
partial indexing
continuous adaptation
cracking tangram

**base data**

**table 1**

A B C D

**table 2**

A B C D

**as queries arrive...**

**table 1**

A B C D

partial materialization

partial indexing

continuous adaptation

storage adaptation

**table 2**

A B C D
cracking tangram

**base data**

<table>
<thead>
<tr>
<th>table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
</tbody>
</table>

as queries arrive...

<table>
<thead>
<tr>
<th>table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

**partial materialization**

**partial indexing**

**continuous adaptation**

**storage adaptation**
cracking tangram

base data

as queries arrive...

partial materialization
partial indexing
continuous adaptation
storage adaptation
no tuple reconstruction
cracking tangram

**base data**

**table 1**

A | B | C | D

**table 2**

A | B | C | D

**as queries arrive...**

**table 1**

A | B | C | D

**table 2**

A | B | C | D

- partial materialization
- partial indexing
- continuous adaptation
- storage adaptation
- no tuple reconstruction
- adaptive alignment
cracking tangram

base data

<table>
<thead>
<tr>
<th>table 1</th>
<th>table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

as queries arrive...

<table>
<thead>
<tr>
<th>table 1</th>
<th>table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

partial materialization
partial indexing
continuous adaptation
storage adaptation
no tuple reconstruction
adaptive alignment
cracking tangram

**base data**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>table 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**as queries arrive...**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>table 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>table 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**partial materialization**

**partial indexing**

**continuous adaptation**

**storage adaptation**

**no tuple reconstruction**

**adaptive alignment**

**sort in caches**
cracking tangram

base data

as queries arrive...

partial materialization
partial indexing
continuous adaptation
storage adaptation
no tuple reconstruction
adaptive alignment
sort in caches
crack joins

table 1

A B C D

A B C D

table 2

A B C D

A B C D

A B C D
cracking tangram

**base data**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**as queries arrive...**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Partial materialization
Partial indexing
Continuous adaptation
Storage adaptation
No tuple reconstruction
Adaptive alignment
Sort in caches
Crack joins
Lightweight locking
cracking tangram

**base data**

**table 1**

A B C D

**table 2**

A B C D

**as queries arrive...**

**table 1**

A B C D

A B C D

**table 2**

A B C D

A B C D

- partial materialization
- partial indexing
- continuous adaptation
- storage adaptation
- no tuple reconstruction
- adaptive alignment
- sort in caches
- crack joins
- lightweight locking
- stochastic cracking

query

random
cracking databases

- basics (CIDR07)
- updates (SIGMOD07)
- >1 columns (SIGMOD09)
- storage restrictions (SIGMOD09)
- robustness (PVLDB12)
- algorithms (PVLDB11)
- concurrency control (PVLDB12)
- benchmarking (TPCTC10)
cracking databases

- basics (CIDR07)
- updates (SIGMOD07)
- > 1 columns (SIGMOD09)
- storage-restrictions (SIGMOD09)
- robustness (PVLDB12)
- algorithms (PVLDB11)
- benchmarking (TPCTC10)
- concurrency control (PVLDB12)
- b-trees (HP Labs)
- hadoop (Yale/Saarland)
- joins (coming up)
- holistic indexing (coming up)
- resilience (coming up)
Positional alignment

Lookup

A(i) = A + i * width(A)
position alignment

lookup

\[ A(i) = A + i \times \text{width}(A) \]

query

\[ \text{max}(B) \text{ where } A < 10 \]
position alignment

lookup

$A(i) = A + i \times \text{width}(A)$

query

$max(B)$ where $A < 10$
**position alignment**

**lookup**

\[ A(i) = A + i \times \text{width}(A) \]

**query**

\[ \max(B) \text{ where } A < 10 \]

<table>
<thead>
<tr>
<th>A</th>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>1</td>
</tr>
<tr>
<td>a2</td>
<td>4</td>
</tr>
<tr>
<td>a3</td>
<td>6</td>
</tr>
<tr>
<td>a4</td>
<td>9</td>
</tr>
<tr>
<td>a5</td>
<td></td>
</tr>
<tr>
<td>a6</td>
<td></td>
</tr>
<tr>
<td>a7</td>
<td></td>
</tr>
<tr>
<td>a8</td>
<td></td>
</tr>
<tr>
<td>a9</td>
<td></td>
</tr>
<tr>
<td>a10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
</tr>
<tr>
<td>a2</td>
</tr>
<tr>
<td>a3</td>
</tr>
<tr>
<td>a4</td>
</tr>
<tr>
<td>a5</td>
</tr>
<tr>
<td>a6</td>
</tr>
<tr>
<td>a7</td>
</tr>
<tr>
<td>a8</td>
</tr>
<tr>
<td>a9</td>
</tr>
<tr>
<td>a10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
</tr>
<tr>
<td>b2</td>
</tr>
<tr>
<td>b3</td>
</tr>
<tr>
<td>b4</td>
</tr>
<tr>
<td>b5</td>
</tr>
<tr>
<td>b6</td>
</tr>
<tr>
<td>b7</td>
</tr>
<tr>
<td>b8</td>
</tr>
<tr>
<td>b9</td>
</tr>
<tr>
<td>b10</td>
</tr>
</tbody>
</table>

Sideways Cracking, SIGMOD 09
positional alignment
lookup
A(i) = A + i * width(A)

query
max(B) where A<10

positional join
positional alignment

lookup
\[ A(i) = A + i \times \text{width}(A) \]

query
\[ \max(B) \text{ where } A < 10 \]

positional join

\begin{align*}
A & \quad \quad \quad B \\
\text{a1} \quad \quad \quad \text{b1} \\
\text{a2} \quad \quad \quad \text{b2} \\
\text{a3} \quad \quad \quad \text{b3} \\
\text{a4} \quad \quad \quad \text{b4} \\
\text{a5} \quad \quad \quad \text{b5} \\
\text{a6} \quad \quad \quad \text{b6} \\
\text{a7} \quad \quad \quad \text{b7} \\
\text{a8} \quad \quad \quad \text{b8} \\
\text{a9} \quad \quad \quad \text{b9} \\
\text{a10} \quad \quad \quad \text{b10}
\end{align*}
positional alignment

lookup

\[ A(i) = A + i \times \text{width}(A) \]

query

\( \max(B) \) where \( A < 10 \)

positional join with cracking

\( \text{Sideways Cracking, SIGMOD 09} \)
sideways cracking
sideways cracking
sideways cracking

A

B

C

D
sideways cracking

query

A

B

C

D
sideways cracking

query

A

B

C

D
sideways cracking

query

A

B

C

D
sideways cracking

query

A  B  C  D
sideways cracking

query

A

B

C

D

.
sideways cracking

A

B

C

D

query
sideways cracking

query
sideways cracking

query
sideways cracking

A

B

C

D

query
replace tuple reconstruction with cracking

log crack actions and replay to align columns dynamically
TPC-H

MonetDB - Sel. Crack - MySQL
Presorted - Sid. Crack

Response time (milli secs)

Query sequence
TPC-H

MonetDB  Sel. Crack  MySQL
Presorted  Sid. Crack  Presorted

Response time (milli secs)

Query sequence

Plain MonetDB
Selection cracking

TPC-H Query 15
764
420

Sideways Cracking, SIGMOD 09
TPC-H

MonetDB  Sel. Crack  MySQL
Presorted  Presorted

Response time (milli secs)

Query sequence

Fully tuned MonetDB
Preparation cost ~3 hours

Selection cracking

Plain MonetDB

TPC-H Query 15

Sel. Crack
Sid. Crack

sel. Crack
Sid. Crack

764
420

~3 hours

TPC-H
TPC-H

Preparation cost ~3 hours

Fully tuned MonetDB

Response time (milli secs)

Query sequence

Plain MonetDB

Selection cracking

MonetDB with sideways cracking

MonetDB

Presorted

Sel. Crack

Sid. Crack

MySQL

Presorted

TPC-H Query 15

764

420

~3 hours
TPC-H

MonetDB • Presorted
Sel. Crack • Presorted
MySQL • Presorted
Sid. Crack

Response time (milli secs)

Query sequence

Fully tuned MonetDB
Preparation cost ~3 hours

Plain MonetDB
Selection cracking
MonetDB with sideways cracking

TPC-H Query 15

764
420
TPC-H

MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
Presorted

Response time (milli secs)

Query sequence

Plain MonetDB
Selection cracking
MonetDB with sideways cracking

Fully tuned MonetDB
Preparation cost ~3 hours

TPC-H Query 15

764
420
TPC-H

```
Preparation cost
~3 hours
```

```
Query sequence

Response time (milli secs)

Sel. Crack
MonetDB
Presorted

Sid. Crack
MySQL
Presorted

MonetDB with sideways cracking
Selection cracking
Plain MonetDB
```

```
TPC-H Query 15
```

```
764
420
```

```
TPC-H
```
TPC-H

reducing data-to-query time

Response time (mili secs)

Query sequence

TPC-H Query 15

764
420

Plain MonetDB

Selection cracking

MonetDB with sideways cracking

Fully tuned MonetDB
Preparation cost ~3 hours
cracking on Skyserver (4TB)
(Sloan Digital Sky Survey, www.sdss.org)

cracking answers 160,000 queries while full indexing is still half way creating one index
cracking on Skyserver (4TB)
(Sloan Digital Sky Survey, www.sdss.org)

cracking answers 160,000 queries while full indexing is still half way creating one index

reducing data-to-query time
time-series indexing

time-series={a1,a2,...,an}
typical query: find a time-series which is similar to time series x
state of the art in time series

does not scale - non interactive

35 hours to index and query (10K queries)
ADS: Adaptive Data Series indexing

built the index tree as queries arrive
ADS: Adaptive Data Series indexing

built the index tree as queries arrive
ADS: Adaptive Data Series indexing

built the index tree as queries arrive

Texmex corpus: 1 billion images - 10K queries

Index building and query answering cost (Hours)

- Querying (Total)
- Indexing (Other)
- Indexing (Output)
- Indexing (Input)

Algorithm

iSAX 2.0

ADS+
**ADS: Adaptive Data Series indexing**

built the index tree as queries arrive

[Graph showing index building and query answering cost (Hours) for different algorithms, with iSAX 2.0 and ADS+ compared.]

Texmex corpus: 1 billion images - 10K queries

Indexing (Input), Indexing (Output), Indexing (Other), Querying (Total)

index and process all queries in 2 hours instead of 12
reducing data-to-query time

built the index tree as queries arrive

Indexing (Input)

Indexing (Output)

Indexing (Other)

Querying (Total)

Texmex corpus: 1 billion images - 10K queries

Index building and query answering cost (Hours)

iSAX 2.0

ADS+

Algorithm

index and process all queries in 2 hours instead of 12
loading

load  tune  query
loading

load
tune
query

copy data inside the database
database now has full control
loading

load
tune
query

copy data inside the database

database now has full control

slow process...not all data might be needed all the time
database vs. unix tools

1 file, 4 attributes, 1 billion tuples

- **DB**
- **Awk**

Single query cost (secs):
- 0
- 550
- 1,100
- 1,650
- 2,200

Adaptive Loading, CIDR 11
database vs. unix tools

1 file, 4 attributes, 1 billion tuples

break down db cost
- Loading: 7%
- Query Processing: 93%

single query cost (secs)
- DB: 0, 550, 1,100, 1,650, 2,200
- Awk: 1,650 seconds
**database vs. unix tools**

1 file, 4 attributes, 1 billion tuples

![Single query cost](chart)

- **Loading**: 93%
- **Query Processing**: 7%

---

**Loading is a major bottleneck**
database vs. unix tools

1 file, 4 attributes, 1 billion tuples

single query cost (secs)

loading is a major bottleneck

... but writing/maintaining scripts is hard too
adaptive loading

load/touch only what is needed and only when it is needed
but **raw data access is expensive**

tokenizing - parsing - no indexing - no statistics

**challenge: fast raw data access**
query plan
Adaptive Loading, CIDR 11

query plan

scan
db
Adaptive Loading, CIDR 11

query plan

scan

loading
access raw data adaptively on-the-fly
access raw data adaptively on-the-fly
selective parsing
file indexing
file splitting
online statistics

access raw data
adaptively on-the-fly
Reducing data-to-query time
towards auto-tuning data kernels
towards auto-tuning data kernels

so what’s next?

adaptive (load-store-execute) cracking (+ AI, + OS, +ML)

compression
disk based cracking
multi-core cracking
row-store cracking

...and many more...
interactive data systems
querying
querying

load  tune  query

SQL interface
querying

load  tune  query

SQL interface

correct and complete answers
querying

complex and slow - not fit for exploration

SQL interface

correct and complete answers
just touch the data you need
just touch the data you need

this is not about query building
it is about query processing
dbTouch demo  (ICDE 2014)
what does this mean for db kernels?
select R.a from R

what does this mean for db kernels?
**db**

```sql
select R.a from R
```

**what does this mean for db kernels?**

**dbTouch**

56 38 45 2

**process only what you touch**
explore: touch, observe and react

the system does not have control of the data flow
the user dictates which is the next tuple

hierarchies of samples
incremental and adaptive operators
adaptive indexing - adaptive loading
rethink db kernels: correct Vs. interactive
rethink db kernels: correct Vs. interactive

break down cost for hash join

- Pointer chasing: 70%
- CPU: 30%
an exploration tool
get a quick feeling about your data
focus on interesting areas

HCl + databases
a database system
allows you to answer queries fast

a data exploration system
allows you to find fast which queries to ask
properties of data exploration systems

easy to use
(no tuning, no set-up)

interactive navigation
(no need for correct/complete answers)
3 Ideas for Big Data Exploration

adaptive systems - tailored for exploration

it is not the strongest species that survive, nor the most intelligent, but the ones most responsive to change

[Darwin, Megginson]

Stratos Idreos
3 Ideas for Big Data Exploration

adaptive systems - tailored for exploration

*it is not the strongest species that survive, nor the most intelligent, but the ones most responsive to change*

[Darwin, Megginson]

Thank you!

Stratos Idreos