Data Systems that are Easy to Design, Tune and Use

Stratos Idreos
every bit matters
it all starts with how we store data

every bit matters

no fixed decisions
from static to dynamic designs
today
today

WANTED
Data Scientists

Stratos Idreos

tomorrow
soon everyone will need to be a “data scientist”

hmm, my data is too big :(
data exploration

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

data has always been big

volume  velocity  variety  veracity
Daily data vs. years

[IBMbigdata]
daily data

years

[IBM big data]

data system design, set-up, tune, use

[Stratos Guess]

data * skills

years
data systems that are easy to:

(\text{years})

design & build
data systems that are easy to:

- design & build
- set-up & tune

(years)

(months)
data systems that are **easy** to:

- design & build
- set-up & tune
- use

(years)

(months)

(hours/days)
too many preparation options lead to complex installation

expert users - idle time - workload knowledge
users/applications
declarative interface
ask what you want

DBA

db system
need to choose the proper system & workloads/applications change rapidly
be able to query the data immediately & with good performance
be able to query the data immediately & with good performance

raw data → knowledge

schema    storage    load    indexing → query
tune = create proper indices offline
performance 10-100X
tune = create proper indices offline
performance 10-100X

but it depends on workload!
which indices to build?
on which data parts?
and when to build them?
storage  load  indexing  query
sample workload
sample workload          analyze


timeline
timeline

storage  load  indexing  query

sample workload  analyze  create indices  query

timeline
The timeline of indexing in complex and time consuming process:

1. Storage
2. Load
3. Indexing
4. Query
5. Sample workload
6. Analyze
7. Create indices
8. Query

Timeline arrow points from storage to query.
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
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
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

relation1/table1

A  B  C  D  ...

...  ...

...  ...

...  ...

Database Cracking CIDR 2007
column-store database

a fixed-width and dense array per attribute
Q1:
select R.A
from R
where R.A > 10
and R.A < 14
Q1: select R.A from R where R.A > 10 and R.A < 14
Q1: select R.A from R where R.A > 10 and R.A < 14
Q1: select R.A from R where R.A > 10 and R.A < 14
Q1: 
select R.A 
from R 
where R.A > 10 
and R.A < 14
Q1: select R.A from R where R.A > 10 and R.A < 14
Q1: select R.A from R where R.A > 10 and R.A < 14

<table>
<thead>
<tr>
<th>column A</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
Q1: select R.A from R where R.A > 10 and R.A < 14
Q1: select R.A from R where R.A > 10 and R.A < 14

piece1: A <= 10

piece2: 10 < A < 14

Database Cracking CIDR 2007
Q1: select R.A from R where R.A>10 and R.A<14

piece1: A<=10

piece2: 10<A<14

piece3: A>=14
Q1:
select R.A
from R
where R.A>10
and R.A<14
Database Cracking CIDR 2007
gain knowledge on how data is organized

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

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

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

result
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  

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

piece1: A <= 10

piece2: 10 < A < 14

piece3: A >= 14

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
### 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**

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

- **piece1:** $A \leq 10$
- **piece2:** $10 < A \leq 14$
- **piece3:** $A \geq 14$

---

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

**Database Cracking CIDR 2007**
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
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

the more we crack, the more we learn

Database Cracking CIDR 2007
select [15,55]
select [15, 55]
select [15,55]

10  20  30  40  50  60
select [15,55]

10  20  30  40  50  60

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]
set-up

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

Response time (secs)

Query sequence (x1000)
set-up

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

almost no
initialization overhead
set-up

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

almost no
initialization overhead

continuous improvement
set-up

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

almost no
initialization overhead

continuous improvement
set-up

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

continuous adaptation

Cumulative average response time (secs)

Query sequence

Full Index

Scan

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

---

Cumulative average response time (secs)

1 10 100 1000 10000

Query sequence

Full Index
Scan
Crack

Database Cracking CIDR 2007
Stratos Idreos

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

10 100 1000 10000
1 10 100 1000
0.1 0.01 0.001 0.0001
0 1 10 100 1000
Cumulative average response time (secs)
Query sequence

Full Index
Scan
Crack

continuous adaptation
<table>
<thead>
<tr>
<th>table1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
</tbody>
</table>
table1

A B C D

...
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 databases

- basics (CIDR07)
  - updates (SIGMOD07)
    - >1 columns (SIGMOD09)
      - storage-restrictions (SIGMOD09)
        - robustness (PVLDB12)
          - algorithms (PVLDB11)
            - benchmarking (TPCTC10)
              - multi-cores (SIGMOD15)
                - hadoop (Yale/Saarland)
        - adaptive storage (SIGMOD14)
          - concurrency control (PVLDB12)
            - time-series (SIGMOD14)
              - adaptive storage (SIGMOD14)
                - hadoop (Yale/Saarland)
          - b-trees (HP Labs)
cracking tangram

base data

as queries arrive...

Table 1

A  B  C  D

Table 2

A  B  C  D
Cracking tangram

Base data

Table 1

Table 2

As queries arrive...

Table 1

Table 2

Stratos Idreos

Harvard School of Engineering and Applied Sciences
cracking tangram

base data

A
B
C
D

table 1

as queries arrive...

A
B
C
D

partial materialization

A
B
C
D

table 1

table 2

A
B
C
D

A
B
C
D

table 2
cracking tangram

base data

A  B  C  D

as queries arrive...

A  B  C  D

partial materialization

partial indexing

A  B  C  D

A  B  C  D

A  B  C  D

A  B  C  D

A  B  C  D

A  B  C  D

A  B  C  D

A  B  C  D

A  B  C  D

base data

table 2

A  B  C  D

table 1

A  B  C  D

table 2

A  B  C  D
cracking tangram

base data

as queries arrive...

partial materialization
partial indexing
continuous adaptation

Stratos Idreos
cracking tangram

base data

as queries arrive...

partial materialization
partial indexing
continuous adaptation
storage adaptation
cracking tangram

base data

A
B
C
D

table 1

as queries arrive...

A
B
C
D

table 1

partial materialization

A
B
C
D

partial indexing

continuous adaptation

storage adaptation

A
B
C
D

table 2

A
B
C
D

table 2
cracking tangram

base data

A
B
C
D

table 1

as queries arrive...

table 1

A
B
C
D

A
B
C
D

partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

A
B
C
D

table 2

A
B
C
D

table 2

A
B
C
D
cracking tangram

As queries arrive...

Partial materialization
Partial indexing
Continuous adaptation
Storage adaptation
No tuple reconstruction
Adaptive alignment
Stratos Idreos

base data
table 1

as queries arrive...
table 1

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

table 2

table 2
cracking tangram

As queries arrive... partial materialization
partial indexing
continuous adaptation
storage adaptation
no tuple reconstruction
adaptive alignment
sort in caches

Table 1

Table 2
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

Stratos Idreos
cracking tangram

base data

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

as queries arrive...

| A | B | C | D |

partial materialization
partial indexing
continuous adaptation
storage adaptation
no tuple reconstruction
adaptive alignment
sort in caches
lightweight locking

| A | B | C | D |

q1

q2
cracking tangram

Table 1

- Base data
- As queries arrive...

Table 2

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

---

Stratos Idreos
Response time (milli secs)

MonetDB  Sel. Crack  MySQL
Presorted  Sid. Crack  Presorted

TPC-H Query 15

Sideways Cracking, SIGMOD 09
Stratos Idreos

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

Response time (milli secs)

TPC-H Query 15

normal MonetDB
selection cracking

Sideways Cracking, SIGMOD 09
Stratos Idreos

TLQ-H Query 15

Response time (milli secs)

MonetDB - Red dots
Presorted - Magenta circles
Sel. Crack - Green triangles
Sid. Crack - Blue triangles
MySQL - Teal dots
Presorted - Orange circles

Query sequence

presorted MonetDB preparation cost 3-14 minutes

normal MonetDB

selection cracking

Sideways Cracking, SIGMOD 09
presorted MonetDB preparation cost 3-14 minutes

Sideways Cracking, SIGMOD 09
Sideways Cracking, SIGMOD 09

Stratos Idreos

presorted MonetDB preparation cost 3-14 minutes

MonetDB with sideways cracking

normal MonetDB

selection cracking

TPC-H Query 15

Response time (milli secs)

Query sequence
presorted MonetDB preparation cost 3-14 minutes

normal MonetDB

selection cracking

MonetDB with sideways cracking

Stratos Idreos

Sideways Cracking, SIGMOD 09
Stratos Idreos

MonetDB  Sel. Crack  MySQL
Presorted  Sid. Crack  Presorted

Response time (milli secs)

TPC-H Query 15

normal MonetDB
selection cracking
MonetDB with sideways cracking

presorted MonetDB preparation cost 3-14 minutes

Sideways Cracking, SIGMOD 09
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
adaptive loading (NoDB, CIDR11/SIGMOD12)
adaptive loading (NoDB, CIDR11/SIGMOD12)

adaptive storage (H20, SIGMOD14)
adaptive loading (NoDB, CIDR11/SIGMOD12)

adaptive storage (H20, SIGMOD14)

adaptive time series indexing (ADS, SIGMOD14)
data systems that are easy to design
(storage, data flow, algorithms, tuning, etc)
e.g., column-stores:
first ideas in 80s,
first advanced architectures in 90s,
first rather complete designs in early 2000s,
industry adoption 2010+
still no indexing, cost based optimizations, …
conflicting goals  moving target
(hardware and requirements change continuously and rapidly)

application requirements

performance

budget

hardware

energy profile
data systems design (and research) is kind of an art
disk  memory  flash  ...
self-designing data systems

data+queries+hardware

->

data system
self-designing data systems

data+queries+hardware -> data system

easy to design

adapt to environment
adaptivity across architecture borders

GENOME-Synthesizer
Gene pool (reuseable modules)

Workload & feature list
Hardware description

Custom tuned architecture
Monitor & detect workload changes

row-store
hybrid store
key-value store
column-store
data systems that are easy to use

data systems that are easy to use

show me something interesting

Queriosity

--- DATA

dbTouch
data systems today
allow us to answer queries fast

data systems tomorrow
should allow us to find fast which queries to ask

db

db explore
instead of making fixed decisions

every query is treated as an advice on how data should be stored
thank you!

Martin Kersten
Stefan Manegold
Goetz Graefe
Harumi Kuno
Anastasia Ailamaki
Themis Palpanas
Eleni Petraki

Ioannis Alagiannis
Miguel Branco
Renata Borovica
Erietta Liarou
Felix Halim
Ronald Yap
Panos Karras

Kostas Zoumpatianos
Manos Athanassoulis
Lukas Maas
Abdul Wasay
Mike Kester
Dhruv Gupta

Stratos Idreos