class 17

updates

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/
**UPDATE** table_name
**SET** column1=value1,column2=value2,...
**WHERE** some_column=some_value

**INSERT INTO** table_name
**VALUES** (value1,value2,value3,...)
traditional applications
e.g., banking

how many times per day do you send update queries to your bank account
the world has changed a little bit by now...
still we spy Facebook more than the # of photos we upload or # of our twitter posts, etc…
not just about user data: everything is data!
monitor CPU utilization

monitor memory hierarchy utilization

monitor clicks (frequency, locations, specific links, sequences)
“Three things are important in the database world: performance, performance, and performance”

Bruce Lindsay, IBM
ACM SIGMOD  Edgar F. Codd Innovations award 2012

ture for both reads & writes
GREAT PERFORMANCE FOR READS AND WRITES & VARIABLE READ/WRITE RATIO

hybrid transactional analytical processing
the next big challenge
insert new entry \((a, b, c, d, \ldots)\) on table \(x\)

update \(N\) columns, \(K\) trees, statistics, \(\ldots\)
TWO CHALLENGES
EFFICIENCY & ROBUSTNESS

hardware/software/* failures,
writes involve >1 steps,
>>1 queries
A deletes (id)

A inserts (id, value)

**Project:**
1 query at a time, no failures
A. deletes
(id)

A. inserts
(id, value)

select(A,v1,v2)

Project:
1 query at a time, no failures
select(A,v1,v2)

Project:
1 query at a time, no failures
select(A,v1,v2)

Project:
1 query at a time, no failures
**Project:**
1 query at a time, no failures

**select(A,v1,v2)**

```plaintext
A
\[\text{scan}\]
A.\text{del}
\[\text{diff}\]
A.\text{ins}
\[\text{scan}\]
A.\text{ins}
}(id,value)
(id)

pos
pos2

\[\text{ins}\]
\[\text{res}\]
\[\text{union}\]
```
**transaction**

any database query that should be seen as a single task

```
plan(
    read X
    write Z
    ...
)
```
Atomicity
Consistency
Isolation
Durability

Jim Gray, IBM, Tandem, DEC, Microsoft
ACM Turing award
ACM SIGMOD Edgar F. Codd Innovations award
Atomicity -> all or nothing
Consistency -> correct
Isolation -> >1 queries
Durability -> persistent
update all rows
where A=v1 & B=v2
  to (a=a/2, b=b/4, c=c-3, d=d+2)
update all rows
where A=v1 & B=v2
to (a=a/2, b=b/4, c=c-3, d=d+2)

search (scan/index)
to find row to update

select+project actions
update all rows where $A=v_1$ & $B=v_2$
to $(a=a/2, b=b/4, c=c-3, d=d+2)$

list of rowIDs (positions) (sort)

search (scan/index) to find row to update

select+project actions

CS165, Fall 2017
Stratos Idreos
update all rows
where $A=v_1$ & $B=v_2$
to $(a=a/2, b=b/4, c=c-3, d=d+2)$

list of rowIDs (positions)
(sort)

search (scan/index)
to find row to update

select+project actions

we know what to update but nothing happened yet
update all rows where $A=v_1 \land B=v_2$ to $(a=a/2, b=b/4, c=c-3, d=d+2)$

if problem (power/abort) before we write all pages we are left with an inconsistent state
classic example

joe owes mike 100$

both joe and mike have a Bank of Bla account

possible actions

joe -100  →  mike + 100

mike + 100  →  joe - 100

what if there is a failure here?
classic example

joe owes mike $100

both joe and mike have a Bank of Bla account

possible actions

joe -100 \rightarrow mike + 100

mike + 100 \rightarrow joe - 100

what if there is a failure here?

actually
1) read mike; 2) mike + 100; 3) write new mike;
update all rows where A=v1 & B=v2 to (a=a/2, b=b/4, c=c-3, d=d+2)

WAL: write ahead log
keep persistent notes as we go
so we can resume or undo
update all rows where $A=v_1 \& B=v_2$ to $(a=a/2, b=b/4, c=c-3, d=d+2)$

WAL: write ahead log
keep persistent notes as we go
so we can resume or undo

read page in L1
update
persist to L2

what do we need to remember (log)
update all rows where $A=v_1$ & $B=v_2$ to $(a=a/2, b=b/4, c=c-3, d=d+2)$

WAL: write ahead log
keep persistent notes as we go so we can resume or undo

what do we need to remember (log)

**restart:** get set of positions again, and either resume from last written page or undo all previously written pages
update all rows where \( A=v_1 \) & \( B=v_2 \) to \((a=a/2, b=b/4, c=c-3, d=d+2)\)

read page in L1
update
persist to L2

(update in place/out of place)
problems when failure
in-place updates: the cardinal sin

The Transaction Concept:
Virtues and Limitations
Jim Gray, Tandem TR 81.3, 1981

(update in place/out of place)
problems when failure
in-place updates: the cardinal sin

The Transaction Concept: Virtues and Limitations
Jim Gray, Tandum TR 81.3, 1981

RUMA has it: Rewired User-space Memory Access is Possible!
Felix Martin Schuhknecht, Jens Dittrich, Ankur Sharma
memory

non volatile memory

SSD

disk

other machines

log
**TWO CHALLENGES**

**EFFICIENCY & ROBUSTNESS**

hardware/software/* failures,
writes involve >1 steps,
>>1 queries
ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging
C. Mohan, Donald J. Haderle, Bruce G. Lindsay, Hamid Pirahesh, Peter M. Schwarz
ACM Transactions on Database Systems, 1992

C Mohan, IBM Research
ACM SIGMOD Edgar F. Codd Innovations award 1993

(rumor has it he has his own Facebook server)
(also check his noSQL and blockchain lectures)
query 1
parse
optimize
plan(
read X
write Z
...
)

query 2
parse
optimize
plan(
read Z
write Z
...
)
... read X
... get_rlock(X)
... read X
... release_rlock(X)
...

... write X
... get_wlock(X)
... write X
... release_wlock(X)
... commit
A survey of B-tree logging and recovery techniques
Goetz Graefe
2 phase locking

1. get all locks
2. do all tasks
3. commit
4. release all locks
(and variations)
2 phase locking

1. get all locks
2. do all tasks
3. commit
4. release all locks
(and variations)

be positive!
optimistic concurrency control
classic example

joe owes mike 100$

both joe and mike have a Bank of Bla account

what about logging and recovery during e-shopping
e.g., shopping cart, wish list, check out

Quantifying eventual consistency with PBS
Peter Bailis, Shivaram Venkataraman, Michael J. Franklin,
Joseph M. Hellerstein, Ion Stoica
Communications of the ACM, 2014
TWO CHALLENGES
EFFICIENCY & ROBUSTNESS

hardware/software/* failures,
writes involve >1 steps,
>>1 queries
remember: **it all starts with how we store the data**

**IDEAL**

a data structure that is best for reads, best for writes and does not require too much metadata
Jim Gray, IBM, Tandem, DEC, Microsoft
ACM Turing award
ACM SIGMOD Edgar F. Codd Innovations award
random access & page-based access

same for writes!

need to only read x… but have to read all of page 1

data value x

page1  page2  page3  …

data move

CPU
registers
on chip cache
on board cache
memory
disk
update value x to y in page p of array z

Level 1

Level 2

page to update

what if >1 updates (no locking for now)
buffer >>1 updates to this page before pushing to L2

page to update
1. read input into stream buffer, hash and write to respective partition buffer
2. when input buffer is consumed, bring the next one
3. when a partition buffer is full, write to L2
e.g., from disk to flash
ideal write granularity is different

what do you think changed in algorithms?
Designing Access Methods: The RUM Conjecture
M. Athanassoulis, M. Kester, L. Maas, R. Stoica, Radu, S. Idreos, A. Ailamaki, M Callaghan
International Conference on Extending Database Technology (EDBT), 2016

you can’t always get what you want
a perfect access method for reads (point queries)
a perfect access method for reads (point queries) but with no memory overhead

binary search to find(x)

- sorted

- reads

- updates

- memory
a perfect access method for writes (point writes)

update(x)

read log

reads
updates
memory
to structure or not to structure

- insert $v$, delete $v$, update $v$ to $v'$

- no order
  - fixed-width & dense

- sorted
  - fixed-width & dense

- sorted
  - fixed-width with holes
NoSQL key-value store & CS265 research/systems projects

great for inserts/updates

great for “recent” reads

optimize as much as possible querying “old” data
all writes go to a small buffer updates are fast reads can find recent updates fast
all writes go to a small buffer
updates are fast
reads can find recent updates fast

data is flushed to L2 when buffer is full
all writes go to a small buffer
updates are fast
reads can find recent updates fast

data is flushed to L2 when buffer is full

data is ordered on ingestion order
no in place updates
getting old data is a full scan
we may have duplicates to clean up
multi-level L2 structure
push to next level when full
remove duplicates with every merge
to improve reads:
each level may be sorted, it may be a b-tree
have a bloom filter, etc

worst case scenario: have to scan the whole thing: empty queries
to improve reads: each level may be sorted, it may be a b-tree have a bloom filter, etc

worst case scenario: have to scan the whole thing: empty queries
why noSQL

complex
legacy
tuning
expensive
...

simple
clean
just enough
...

as apps become more complex

as apps need to be more scalable

newSQL

Mesa: Geo-Replicated, Near Real-Time, Scalable Data Warehousing
Ashish Gupta et al.
International Conference on Very Large Databases (VLDB), 2014

F1: A Distributed SQL Database That Scales
Jeff Shute et al.
International Conference on Very Large Databases (VLDB), 2013
A Self-Designing Key-Value Store

monkey @SIGMOD2017  daslab.seas.harvard.edu/crimsondb
A Self-Designing Key-Value Store

10x faster

monkey @SIGMOD2017  daslab.seas.harvard.edu/crimsondb
A Self-Designing Key-Value Store

10x faster

CS265 research and systems projects

monkey @SIGMOD2017  daslab.seas.harvard.edu/crimsondb
& check out sections 11, 12, 13

textbook: chapters 16, 17, 18

Positional update handling in column stores
Sándor Héman, Marcin Zukowski, Niels J. Nes, Lefteris Sidirourgos, Peter A. Boncz
In Proc. of the ACM SIGMOD Inter. Conference on Management of Data, 2010

Updating a cracked database
Stratos Idreos, Martin Kersten, Stefan Manegold
updates

DATA SYSTEMS

prof. Stratos Idreos