Schedule:
2 more lectures this week,
2-3 research lectures as of next week (including research projects),
Then discussion papers
This Friday: discussion papers & schedule. Register for presentations.
Schedule:
2 more lectures this week,
2-3 research lectures as of next week (including research projects),
Then discussion papers
This Friday: discussion papers & schedule. Register for presentations.

Systems Projects:
This should be the last week of reading and getting familiar with tools.
Next week you should start coding & start going to labs regularly.
Schedule:
2 more lectures this week,
2-3 research lectures as of next week (including research projects),
Then discussion papers
This Friday: discussion papers & schedule. Register for presentations.

Systems Projects:
This should be the last week of reading and getting familiar with tools.
Next week you should start coding & start going to labs regularly.

Research Projects:
New readings this week. Read carefully until next week. And read projects.
Next week: project assignments.

Stratos Idreos, Konstantinos Zoumpatianos, Brian Hentschel, Michael Kester, Demi Guo. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2018
Design Continuums and the Path Toward Self-Designing Key-Value Stores that Know and Learn.


Optional but read especially if you think about doing a research project in NoSQL
Limousine: Blending Learned and Classical Indexes to Self-Design Larger-than-Memory Cloud Storage Engines.
Subarna Chatterjee, Mark F. Pekala, Lev Kruglyak, and Stratos Idreos.
In Proceedings of the ACM Management of Data 2, 1, Article 47 (February 2024), (SIGMOD), 2024

Subarna Chatterjee, Meena Jagadeesan, Wilson Qin, and Stratos Idreos.
In Proceedings of the Very Large Databases Endowment, (PVLDB), 2022
The Image Calculator: 10x Faster Image-AI Inference by Replacing JPEG with Self-designing Storage Format.
Utku Sirin and Stratos Idreos.
In Proceedings of the ACM Management of Data 2, 1, Article 52 (February 2024), (SIGMOD), 2024

μ-TWO: 3× Faster Multi-Model Training with Orchestration and Memory Optimization Storage, Scheduling, and Networking
Sanket Purandare, Abdul Wasay, Stratos Idreos, Animesh Jain
MLSys 2023, The Annual Conference on Machine Learning and Systems
Preparing for presentations and reviews

review and slides should answer:

what is the problem
why is it important
why is it hard
why existing solutions do not work
what is the core intuition for the solution
solution step by step
does the paper prove its claims
exact setup of analysis/experiments
are there any gaps in the logic/proof
possible next steps

* follow a few citations to gain more background

And final question:
What can we use from this paper
technique or inspiration
towards self-designing systems?
how to prepare slides

no bullets  2 colors  big text  images  animation for examples
how to prepare slides

no bullets  2 colors  big text  images  animation for examples

story

one message per slide  connection from slide to slide
how to prepare slides
no bullets  2 colors  big text  images  animation for examples

story
one message per slide  connection from slide to slide

discussion
add 3-4 interesting discussion points
Debug before presenting

1. Content: Did you answer all reviews questions?

2. Formatting: Did you apply all slide format rules?

3. Feedback: Send to Stratos 2 days before presentation.
Our goal:

Every presentation is better than the previous one

During discussion we will also touch on the presentation in every class

Grading will be relative to the presentation schedule
Reviews

1. Content: Did you answer all reviews questions?

2. Feedback: In every class, we pick 2 reviews to read and give feedback.

3. Grading: we will grade the last three reviews.
What-if we **add bloom filters** in the hash-table buckets?
What-if the workload changes to 90% writes?
What-if we **buy faster CPU X?**
What-if we buy faster CPU X?

~20 SECONDS
(workload: 10 Million entries, 100 queries)
Deep Reinforcement Learning

Neural Net

Cost Estimation/Synthesis

Bayesian Optimization

Genetic Algorithms

Micro-benchmarks train models on different hardware profiles. Given a function $f(x)$, the goal is to find $x$ that minimizes $f(x)$:

$$\text{arg min } f(x)$$

Data Layout and Index Synthesis

Operation Synthesis

Hardware Conscious Synthesis

Concurrency & Updates

Design Continuums

Fast analytical model optimization

Memoization

H/W Pruning

Expert Rules

Decision-making in reinforcement learning involves updating designs iteratively.

Systematically evaluate various designs for each node.

Input

Workload

Hardware

Cons

SLAs

Performance Constraints

- Initial design
- Time threshold
- Distance to optimal

Feedback

Output

Data structure design

Code

Data Flow

Feedback

Final Design

Code Generator

Hardware Profiles

Optimization

AST

SLAs

H/W Pruning

Expert Rules

Memoization

Convergence

Learned Shortcuts

High confidence

Low confidence

Update design

Node by node design process

Design Search

Reinforcement Learning

Deep Reinforcement Learning

Bayesian Optimization

Hardware Profiles

Equality Scan

Range Scan

Binary Search

Micro-benchmarks

Cost Synthesizer

Machine Learning

Equation: $f(x) = ax + b$
Deep Reinforcement Learning

Neural Net

Cost Estimation/Synthesis

Bayesian Optimization

Genetic Algorithms

Input

Output

Workload

Hardware

SLAs

Performance Constraints - Initial design - Time threshold - Distance to optimal

Learned Shortcuts

High confidence

Low confidence

Feedback

Data Layout and Index Synthesis

Overall Designs

Data Access Primitives

Cost Synthesizer

Layout Primitives

Operation Synthesis

Generalized Cost and Algorithm Synthesis

Micro-benchmarks train models on different hardware profiles.

\[
E(x) = \sum_{i=1}^{n} c_i \cdot f(x)
\]

Machine Learning

Hardware Conscious Synthesis (Level 2)

Space Efficiency Optimization

Concurrency & Updates

Put

Get

Delete

Bulk Load

Put

Get

Delete

Bulk Load

Policy Synthesis

Partial design

Update design

Design Continuums

Fast analytical model optimisation

arg \min f(x)

Memoization

H/W Pruning

Expert Rules

Reinforcement Learning

Cost Estimation/Synthesis

Neural Net

Design Search

Iterative Search

Node by node design process

Systematically evaluate various designs for each node

 Otherwise

Is it a design continuum?

Node has partitioning?

Node has Bloom filters?

Node has Zone maps?

Put

Get

Delete

Bulk Load

Put

Get

Delete

Bulk Load

Data Node (Element) Design Space

Linked List

Try

Skip List

B-Tree

Internal

B+Tree

Internal

Design Continuums

Data Access Primitives

Equality Scan

Range Scan

Binary Search

Random Probe

Hardware Profiles

Put

Get

Delete

Bulk Load

Put

Get

Delete

Bulk Load

Layout Primitives

Data Node (Element) Library

Concurrent Design Continuums

Zone Maps

Bloom Filter

Node has partitioning?

Node has Bloom filters?

Node has Zone maps?

Data Structure Design

Overall Designs

Sorted Search

Binary Search

Random Probe

Hardware Profiles

Put

Get

Delete

Bulk Load

Put

Get

Delete

Bulk Load

Learning Shortcuts

Learned shortcuts

Decision tree

Put

Get

Delete

Bulk Load

Put

Get

Delete

Bulk Load

Key Value Data Structures

<k, v>
function CompleteDesign(Q, E, l, currentPath = [], H)
    if blockReachedMinimumSize(H.page_size) then
        return END_SEARCH;
    if !meaningfulPath(currentPath, Q, l) then
        return END_SEARCH;
    if (cacheHit = cachedSolution(Q, l, H)) != null then
        return cacheHit;
    bestSolution = initializeSolution(cost=\infty);
    for E ∈ E do
        tmpSolution = initializeSolution();
        tmpSolution.cost = synthesizeGroupCost(E, Q);
        updateCost(E, Q, tmpSolution.cost);
        if createsSubBlocks(E) then
            Q′ = createQueryBlocks(Q);
            currentPath.append(E);
            subSolution = CompleteDesign(Q′, E, l + 1, currentPath);
            if subSolution.cost != END_SEARCH then
                tmpSolution.append(subSolution);
                Q = Q′;
            end
        end
        if tmpSolution.cost ≤ bestSolution.cost then
            bestSolution = tmpSolution;
        end
    end
    cacheSolution(Q, l, bestSolution);
return bestSolution;
WE CAN AUTOMATICALLY DESIGN/DEBUG/FILL IN
WE CAN AUTOMATICALLY DESIGN/DEBUG/FILL IN

![Diagram showing read, update, and memory processes with corresponding text and images]

- Read
- Update
- Memory

Hybrid B+Tree / Hash Table / Array

- B+TREE ELEMENT
- Point get intensive
- Range intensive
- Only writes
- HASH PARTITIONING
- DATAPAGE (system page size)
- DATAPAGE (large chunks)
- DATAPAGE (system page size)

Synthesis cost (min.) vs. # of inserts

Graph showing the relationship between number of inserts and synthesis cost.
WE CAN AUTOMATICALLY DESIGN/DEBUG/FILL IN
STARS ON THE SKY

POSSIBLE DATA STRUCTURES

(10^24)

(10^32, 2-node)
(10^48, 3-node)
The design space of systems is even larger
The design space of systems is even larger
manually selecting “good” designs

- B-tree based KV-system
- LSM-tree based KV-system
- LSH based KV-system
manually selecting "good" designs
manually selecting "good" designs
manually selecting "good" designs
manually selecting "good" designs
manually selecting "good" designs
selecting "good", manually selecting "good", designing "good", manually designing "good"
write-cost

key retention
value retention
partitioning (range, time, ...)
sub-block location
sub-block (skip-)links
size ratio
merge policy
filters bits per entry
size of buffer/cache
internal k-v layout
key retention
value retention
partitioning (range, time, ...)
sub-block location
sub-block (skip-)links
size ratio
merge policy
filters bits per entry
size of buffer/cache
internal k-v layout
read-cost

manually selecting "good", manually selecting designs
discrete

LSM-tree

great writes
decent reads
space amp

data
data
data

B-tree

index
data

good reads
ok updates
space amp
performance continuum?
performance continuum?
performance continuum?
not LSM-tree, not B-tree
mixed design principles
hybrid performance properties
design continuum

not LSM-tree, not B-tree
mixed design principles
hybrid performance properties
totally different?

Level 1

Level 2

Level 3

... ...

Level N

index (fence+pointer)

data
totally different?

Level 1: index + data

Level 2: index + data

Level 3: index + data

... ... ...

Level N: index + data

index (fence + pointer)

data
>1 B-trees that have not (yet) been merged
>1 B-trees that have not (yet) been merged

B-tree entries have not been propagated to the leaves
navigation transitions
a unified design space
[log, log+hash, LSM-tree*, BεTree, B-Tree, Sorted Array]
a unified design space

[log, log+hash, LSM-tree*, BεTree, B-Tree, Sorted Array]

design principles for >1 structures
a unified design space

[log, log+hash, LSM-tree*, B^TTree, B-Tree, Sorted Array]
a unified design space

[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]
a unified design space
[log, log+hash, LSM-tree*, BεTree, B-Tree, Sorted Array]

Buffer

Filters

Fence Pointers

storage block
node
run boundary
fence pointer

storage block

Y cold levels

L-Y hot levels

Memory

Filter Choice by Mem. Budget

Bloom Filter

Hash Table

(Filter Choice by Mem. Budget)

(Fewer Bits per Key) (Full Key Size)

Filter Choice by Mem. Budget

εTree, B-Tree, Sorted Array]
a unified design space
[log, log+hash, LSM-tree*, BεTree, B-Tree, Sorted Array]

- Filters
- Fence Pointers
- Storage
- Memory

Filter Choice by Mem. Budget

- Bloom Filter (Fewer Bits per Key)
- Hash Table (Full Key Size)
a unified design space
[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]

Buffer

Filters

Fence Pointers

Storage

Memory

- Bloom Filter
- Hash Table

Filter Choice by Mem. Budget

(Fewer Bits per Key) (Full Key Size)

storage block node
run boundary fence pointer

L-Y hot levels

Y cold levels
a unified design space

[log, log+hash, LSM-tree*, BE^Tree, B-Tree, Sorted Array]
a unified design space

[log, log+hash, LSM-tree*, BεTree, B-Tree, Sorted Array]