Hector: A Framework to Design Scheduling Strategies in Persistent Key-Value Stores

L.-C. Canon[1], A. Dugois[1,2], L. Marchal[2] and E. Rivière[3]

[1] FEMTO-ST, Univ. Franche-Comté, Besançon, France
[2] LIP, ENS Lyon, Inria, Lyon, France
[3] ICTEAM, UCLouvain, Louvain-la-Neuve, Belgium

August 9, 2023 – 52nd International Conference on Parallel Processing (ICPP 2023)

Introduction to Key-Value Stores

Key-value store

NoSQL database where each data item is identified by a unique key.

Examples: Dynamo, Cassandra

OPERATIONS `get(k)` `put(k,v)` `scan(k1,k2)`

Introduction to Key-Value Stores

Key-value store

NoSQL database where each data item is identified by a unique key.

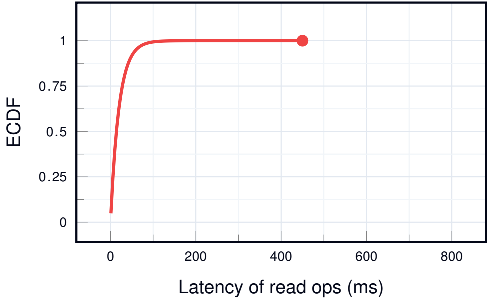
Examples: Dynamo, Cassandra

OPERATIONS `get(k)` `put(k,v)` `scan(k1,k2)`

Introduction to Key-Value Stores

Must be

- Highly scalable
- Highly available
- **Blazing fast**



Tail latency problem

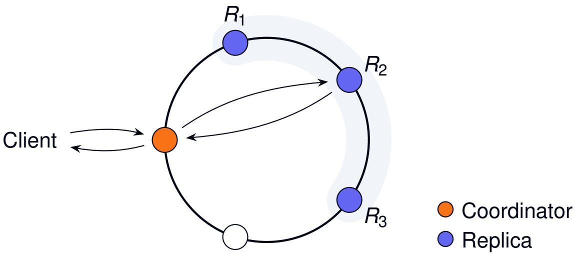
1 service request = many read operations.

$<$ 1% slow ops = degraded QoS for most users.

Scheduling in Apache Cassandra

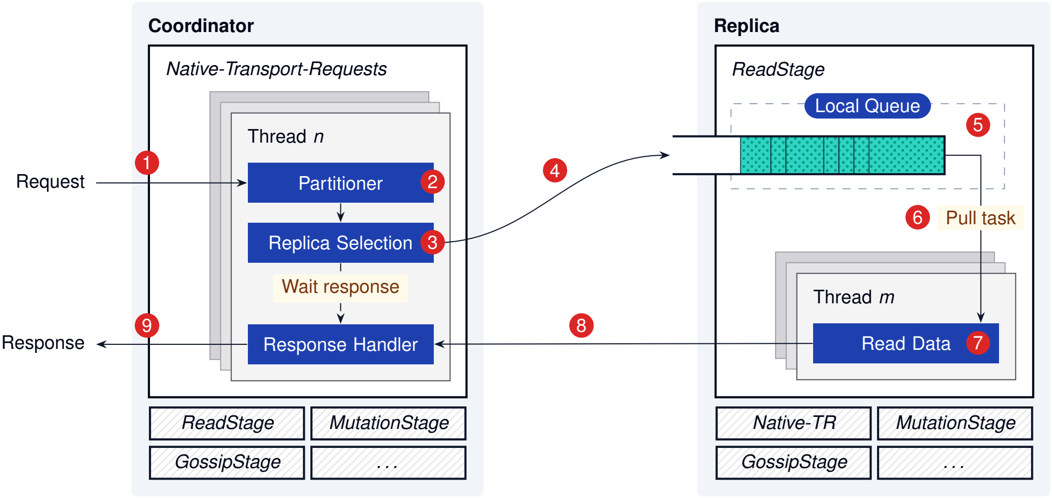
Request Execution

1. A node receives a client request
2. The read operation is forwarded to appropriate storage server
3. The server performs the read
4. The result is sent back to the client



Scheduling in Apache Cassandra

Request Execution



Canon et al. Hector: A Framework to Design Scheduling Strategies in Persistent Key-Value Stores. ICPP 2023.

Scheduling in Apache Cassandra

Prior Work

Suresh et al. *C3: Cutting tail latency in cloud data stores via adaptive replica selection.* (2015)

Reda et al. *Rein: Taming tail latency in key-value stores via multiget scheduling.* (2017)

Jaiman et al. *Héron: taming tail latencies in key-value stores under heterogeneous workloads.* (2018)

Jiang et al. *Haste makes waste: The On–Off algorithm for replica selection in key–value stores.* (2019)

Jaiman et al. *TailX: Scheduling heterogeneous multiget queries to improve tail latencies in key-value stores.* (2020)

Scheduling in Apache Cassandra

Observations & Challenges

	Observations	Challenges
Feature-related	• 2 critical steps • Need info on cluster • Need info on workload	• Poor replica selection • Poor local scheduling • No info provider
API-related		
Evaluation-related		

Scheduling in Apache Cassandra

Observations & Challenges

	Observations	Challenges
Feature-related	• 2 critical steps • Need info on cluster • Need info on workload	• Poor replica selection • Poor local scheduling • No info provider
API-related	• Huge codebase • No unified API	• Difficult to extend • Error-prone
Evaluation-related		

Scheduling in Apache Cassandra

Observations & Challenges

	Observations	Challenges
Feature-related	• 2 critical steps • Need info on cluster • Need info on workload	• Poor replica selection • Poor local scheduling • No info provider
API-related	• Huge codebase • No unified API	• Difficult to extend • Error-prone
Evaluation-related	• No common baseline • Different assumptions	• No easy comparison • No easy reproducibility

Scheduling Framework Hector

Hector is a fully-integrated scheduling framework built in Apache Cassandra.

`https://github.com/anthonydugois/hector`

Apache Cassandra 4.2

Modular components

No conflict

Simple API

Single config file

...

Introducing Hector

Modular Components

Replica Selection

Select the “best” replica to serve a request

Metadata

Local Scheduling

Schedule local operations in optimized order

Cluster State

Propagate info on the cluster state to help scheduling

Workload Oracle

Gather information on the workload to help scheduling

Evaluation

1. Setup environment
2. Define scheduling settings
   - Optional: adapt implementations
3. Run experiments
4. Go to Step 2

Example of config file

```
replica_selector :
  — class_name: hector.C3ScoringSelector
    parameters:
      — concurrency_weight: '5.0'

local_read_queue:
  — class_name: hector.FIFOReadQueue

state_feedback:
  — PENDING_READS
  — SERVICE_TIME
```

Default schedulers in Apache Cassandra

Replica Selection

Dynamic Snitching (DS)
Periodically compute a score based on latency history; select the replica with lowest score

Local Scheduling

First Come First Served (FCFS)
Process operations in order of arrival

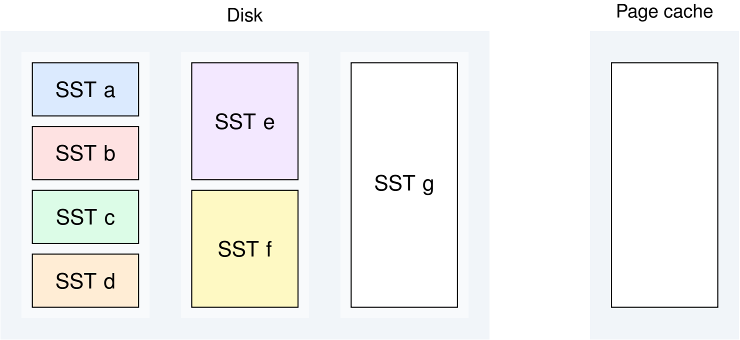
Suresh et al. *C3: Cutting tail latency in cloud data stores via adaptive replica selection.* (2015)

Replica Selection (+ Cluster State)	**C3** Continuously compute a score based on latency history, queue size, pending operations; select the replica with lowest score

Introducing Hector

Schedulers

Idea: leverage the Linux page cache to reduce the number of disk accesses

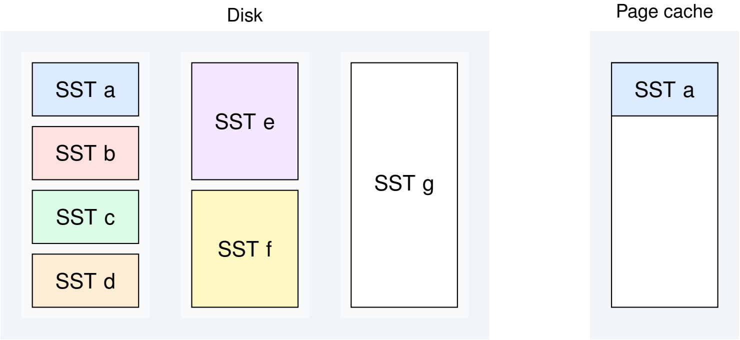


SST = Sorted String Table file

Introducing Hector

Schedulers

Idea: leverage the Linux page cache to reduce the number of disk accesses

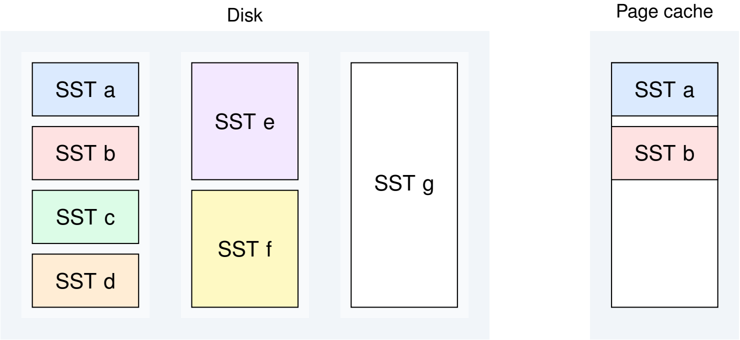


SST = Sorted String Table file

Introducing Hector

Schedulers

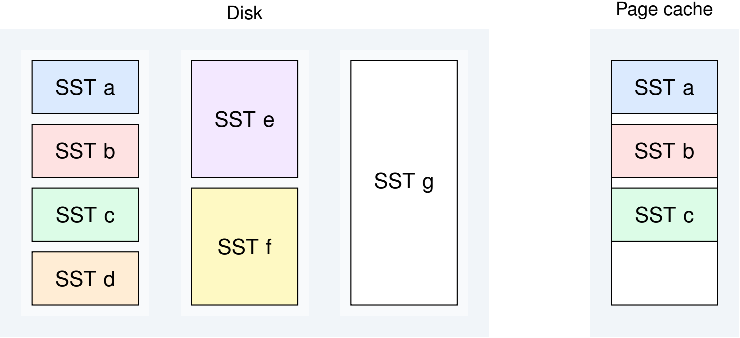
Idea: leverage the Linux page cache to reduce the number of disk accesses



SST = Sorted String Table file

Introducing Hector
Schedulers

Idea: leverage the Linux page cache to reduce the number of disk accesses

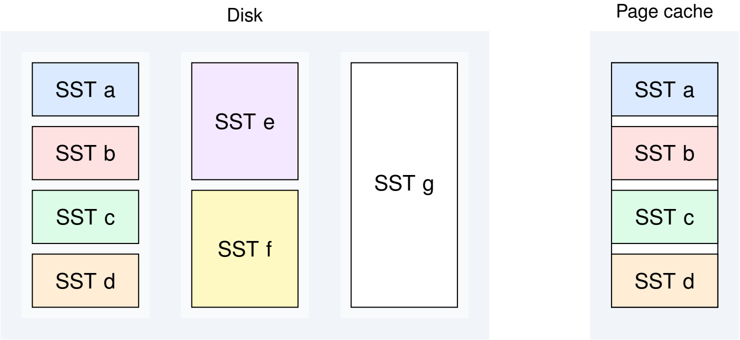


SST = Sorted String Table file

Introducing Hector

Schedulers

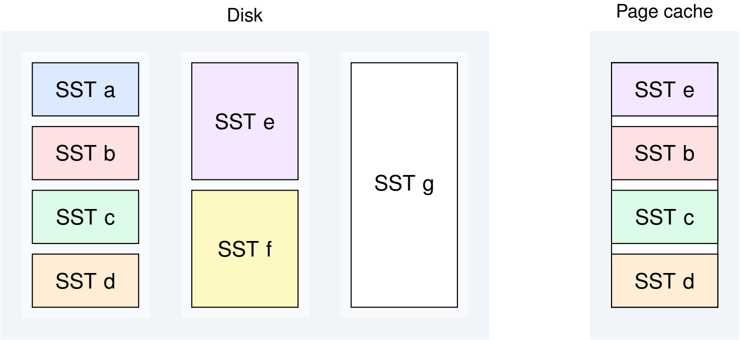
Idea: leverage the Linux page cache to reduce the number of disk accesses



SST = Sorted String Table file

Introducing Hector

Idea: leverage the Linux page cache to reduce the number of disk accesses

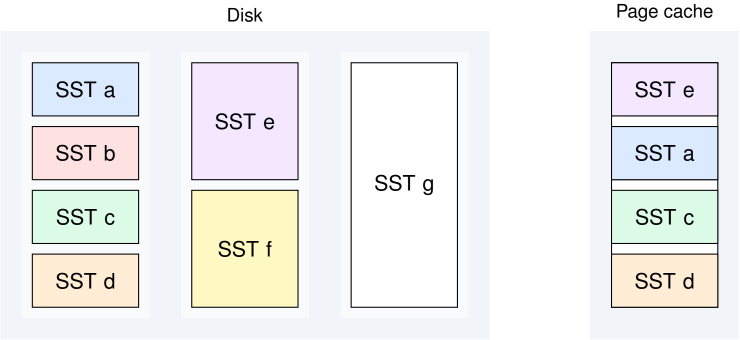


SST = Sorted String Table file

Canon et al. Hector: A Framework to Design Scheduling Strategies in Persistent Key-Value Stores. ICPP 2023.

Introducing Hector
Schedulers

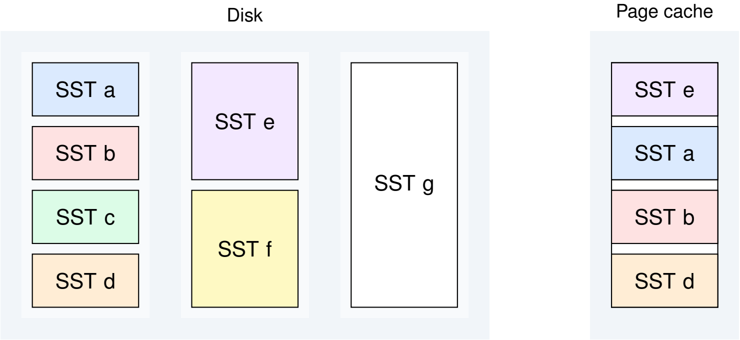
Idea: leverage the Linux page cache to reduce the number of disk accesses



SST = Sorted String Table file

Introducing Hector
Schedulers

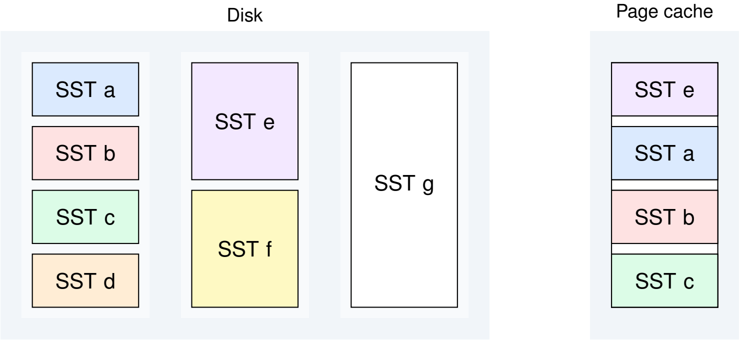
Idea: leverage the Linux page cache to reduce the number of disk accesses



SST = Sorted String Table file

Introducing Hector
Schedulers

Idea: leverage the Linux page cache to reduce the number of disk accesses



SST = Sorted String Table file

Replica Selection
(+ Workload Oracle)

Popularity-Aware (PA)
According to popularity of keys, favor page cache hit (low popularity) or load balancing (high popularity)

Idea: assign priorities to operations.

Local Scheduling (+ Metadata)	**Random Multi-Level (RML)** Process operations in order of priority

Idea: assign priorities to operations.

Local Scheduling **Random Multi-Level (RML)**

(+ Metadata) Process operations in order of priority

In this talk: “**fast**” operation = **high** priority

Experimental Evaluation

Settings

- Grid'5000 testbed
- 15 identical servers

  18-core Intel Xeon Gold 5220 + 96 GiB RAM + 480 GiB SSD
- 150 GiB of data per server
- 5 benchmark clients
- Synthetic workload
- Production settings

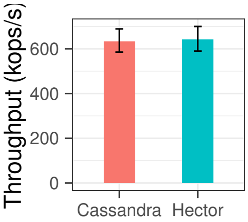
Hector Overhead

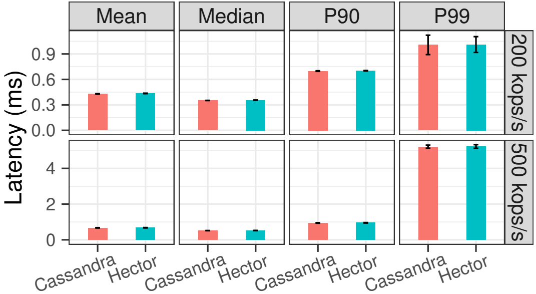
Cache-Locality Effects

Heterogeneous Scheduling

Experimental Evaluation

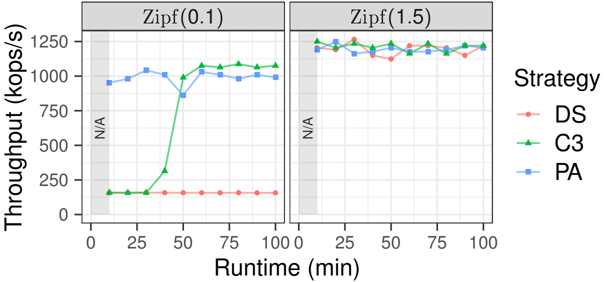
Hector Overhead





Experimental Evaluation

Cache-Locality Effects

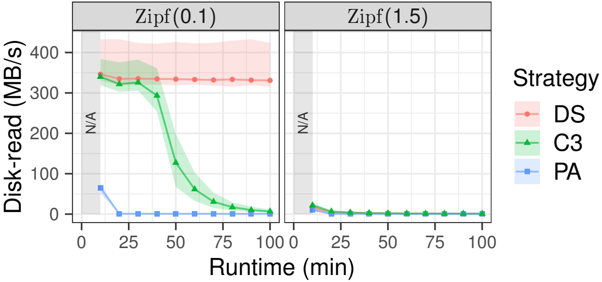


Settings

- Item size: 1 kB
- $\mathrm{Zipf}(0.1)$: quasi-uniform
- $\mathrm{Zipf}(1.5)$: heavily-skewed

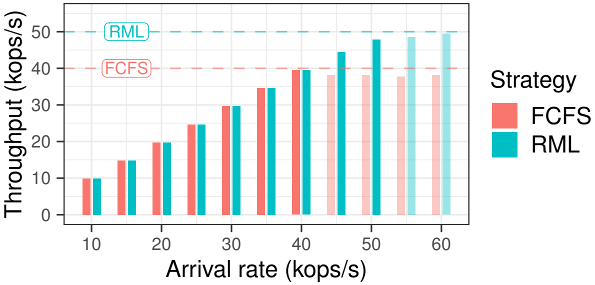
Experimental Evaluation

Cache-Locality Effects



Experimental Evaluation

Heterogeneous Scheduling

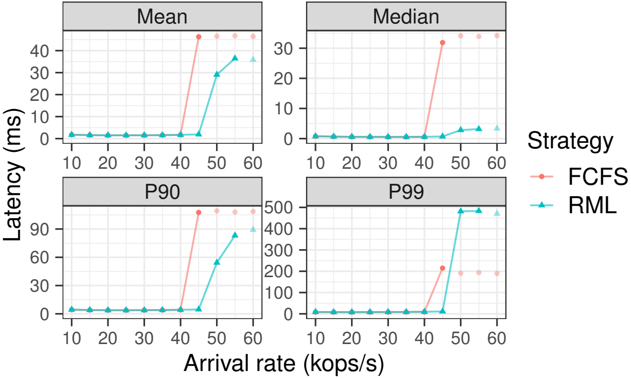


Settings

- Small item size: 1 kB
- Large item size: 1 MB
- Small/large ratio: 3:1
- Uniform popularity

Experimental Evaluation

Heterogeneous Scheduling



Conclusion

Hector benefits

- Easier implementation
- Comparisons over baseline
- Testing new ideas
- No overhead

Future work

- Support multi-get operations
- Exhaustive evaluation campaign

Conclusion

Thank you for your attention!

`https://github.com/anthonydugois/hector`