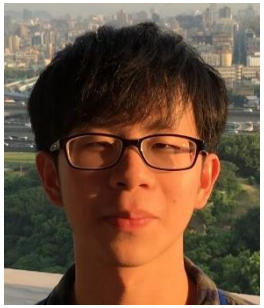
Don’t Look Back, Look into the Future: Prescient Data Partitioning and Migration for Deterministic Database Systems

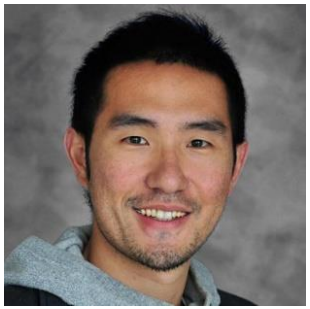
Yu-Shan Lin

Ching Tsai

Tz-Yu Lin



Yun-Sheng Chang



Shan-Hung Wu

National Tsing Hua University, Taiwan

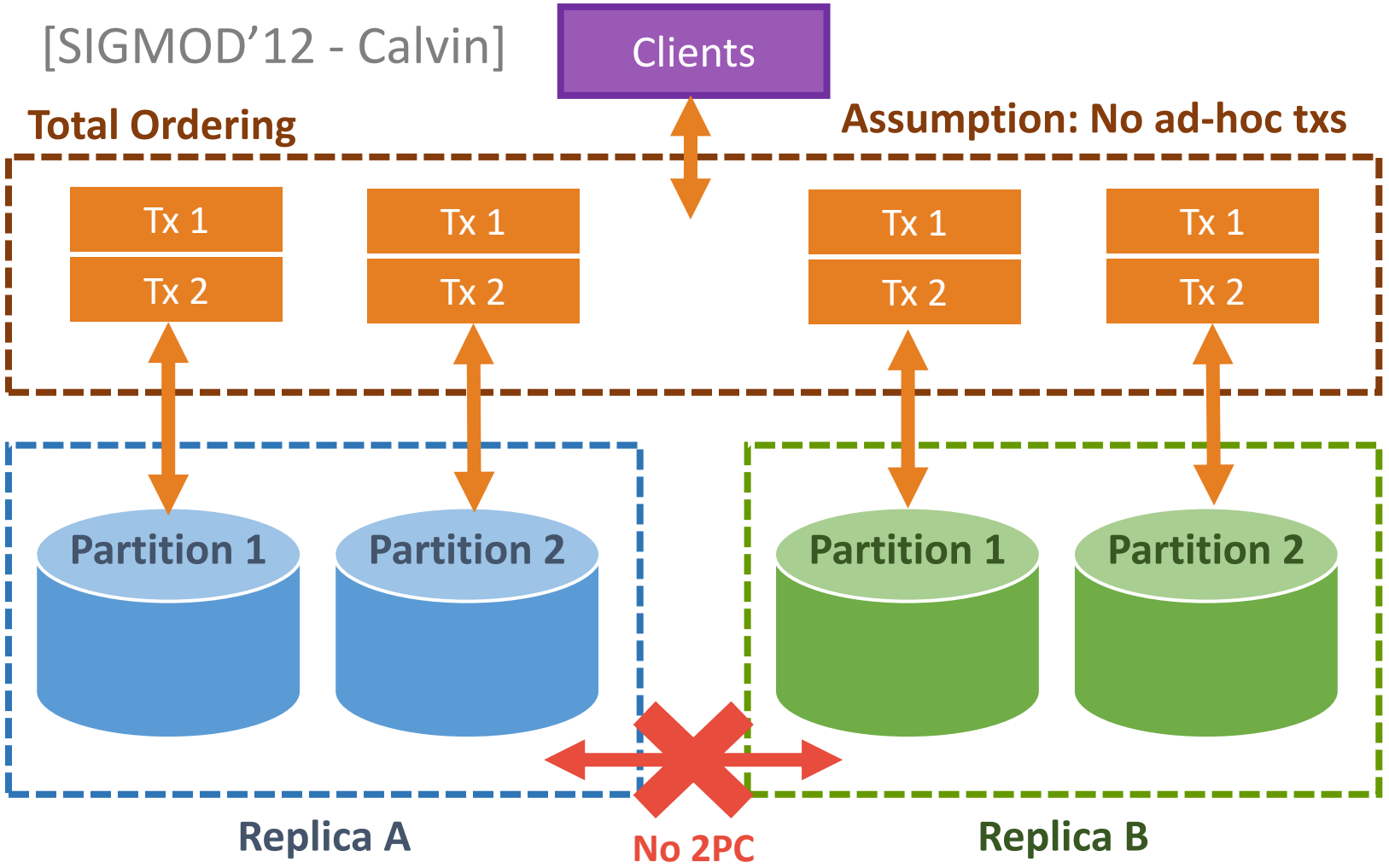
Outline

- Background
  - When a deterministic DBMS meets dynamic workloads
- Related Work
  - Look-Back Approaches
  - Look-Present Approaches
- Hermes
  - Key Observation
  - Key Idea
  - The Prescient Routing Algorithm
- Experiment Results
- Conclusion

Outline

- Background
  - When a deterministic DBMS meets dynamic workloads
- Related Work
  - Look-Back Approaches
  - Look-Present Approaches
- Hermes
  - Key Observation
  - Key Idea
  - The Prescient Routing Algorithm
- Experiment Results
- Conclusion

Deterministic DBMSs Has Been Proved to Be Highly Scalable and Highly Available



Good Scalability Rely on High Quality of Data Partitions

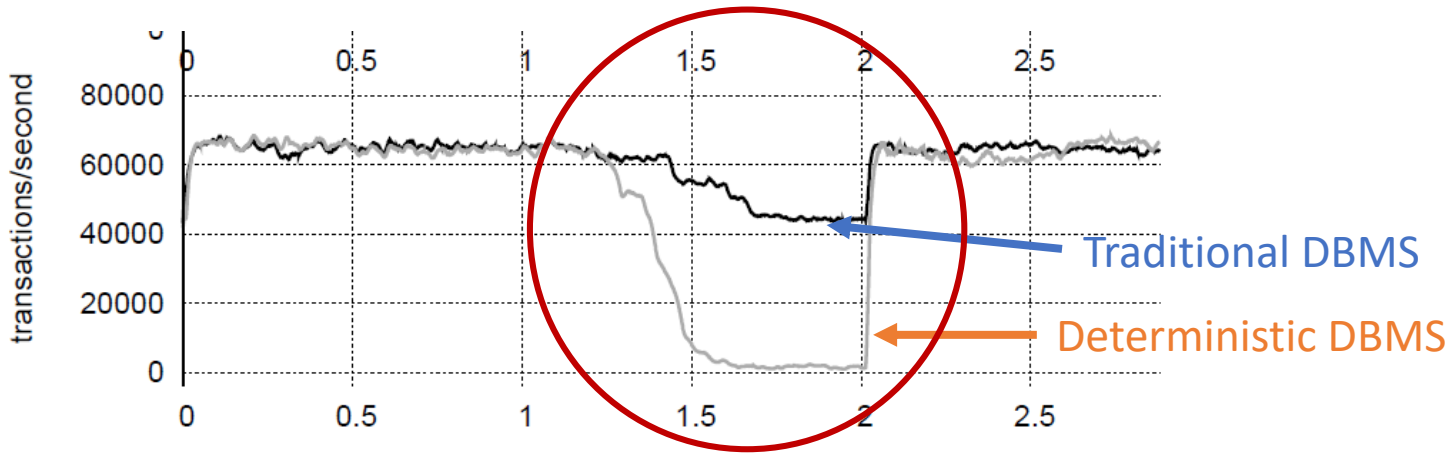
- However, in order to achieve such performance, the database must be **well partitioned**.

- Poor data partitions introduce
  - Many distributed transactions.
  - Unbalanced loads.

Deterministic DBMSs Are Sensitive to Stalled Transactions

- Distributed transactions and overloaded nodes all introduce additional delay for transactions.

- The clogging problem



Thomson at el., "The Case for Determinism in Database Systems", in VLDB'10

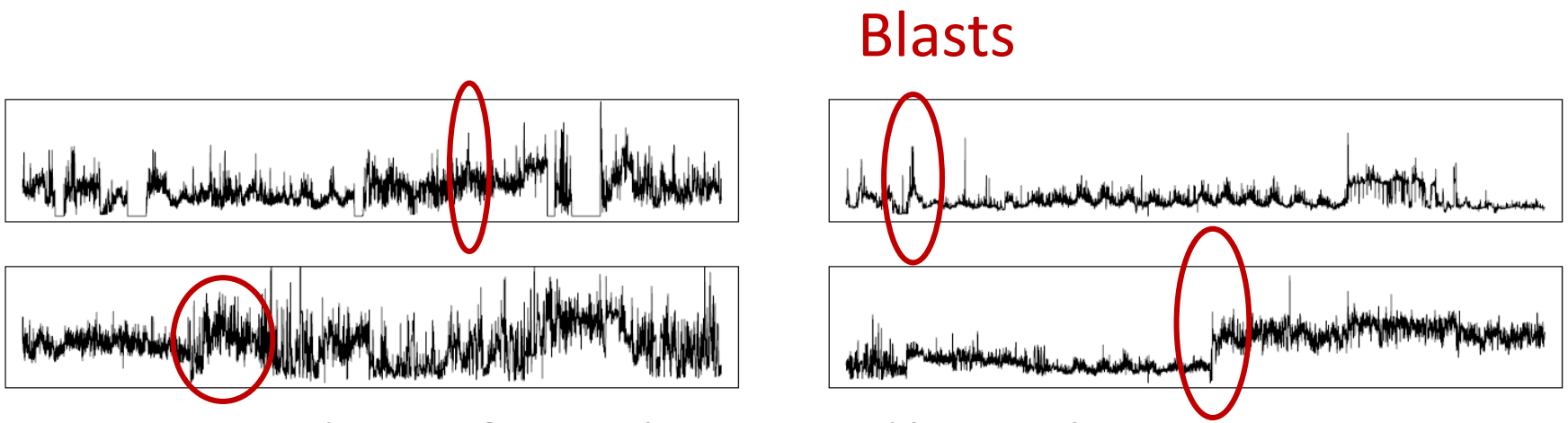
Things Getting Worse on Dynamic Workloads

- **Dynamic workloads** makes the best data partitions constantly change.

- In order to fit the latest workload, a DBMS may have to **keep re-partitioning** its database on-line.
  - This also requires the system to perform a time-consuming data migration.

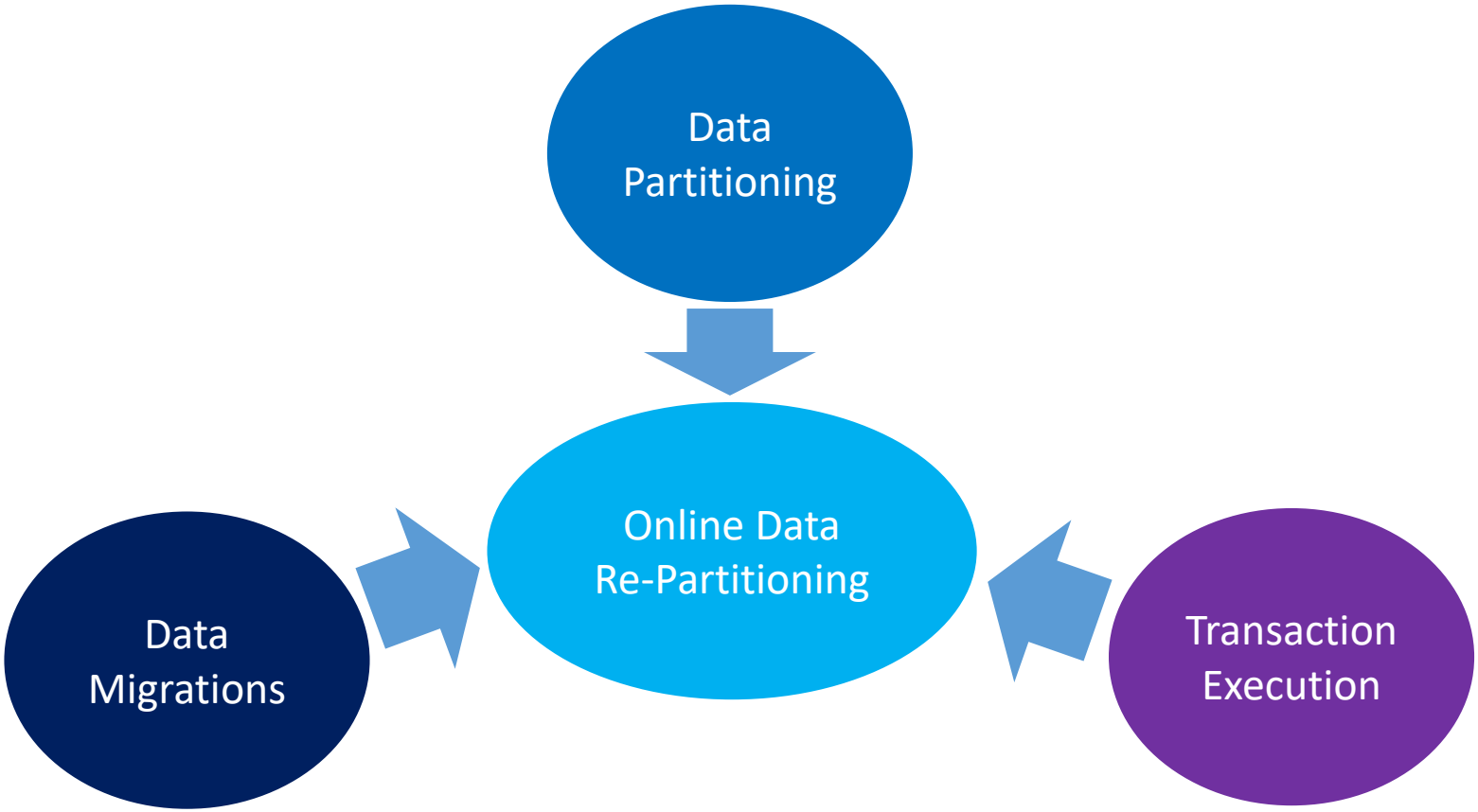
Even Worse: Episodic Workload Changes

- Blast workloads are usually **unpredictable** and show in a small amount of time.
  - It is hard for a DBMS to change its data partitions for these workloads in time.



Load Traces from a Cluster owned by Google in 30 Days

Performing Effective Online Data Re-partitioning Requires Cooperation of Many Different Components



Is it possible to jointly optimize data partitioning, data migration, and transaction execution for dynamic and episodic workloads in a simple way?

Outline

- Background
  - When a deterministic DBMS meets dynamic workloads
- Related Work
  - Look-Back Approaches
  - Look-Present Approaches
- Hermes
  - Key Observation
  - Key Idea
  - The Prescient Routing Algorithm
- Experiment Results
- Conclusion

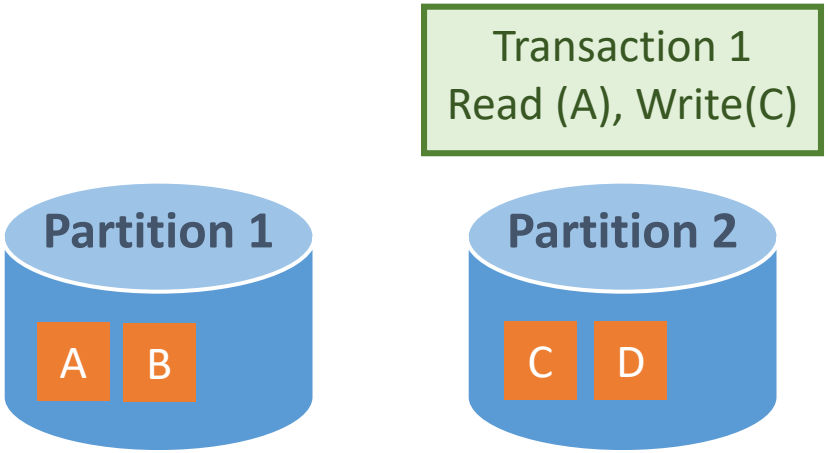
Look-Back Approaches

- Key Idea: plans new data partitions based on **a past workload history**.
  - E.g., E-Store [VLDB'14], Clay [VLDB'16]

- Cons
  - Needs to perform dedicated and slow data migrations.
  - Cannot react against unpredictable and episodic workload changes.

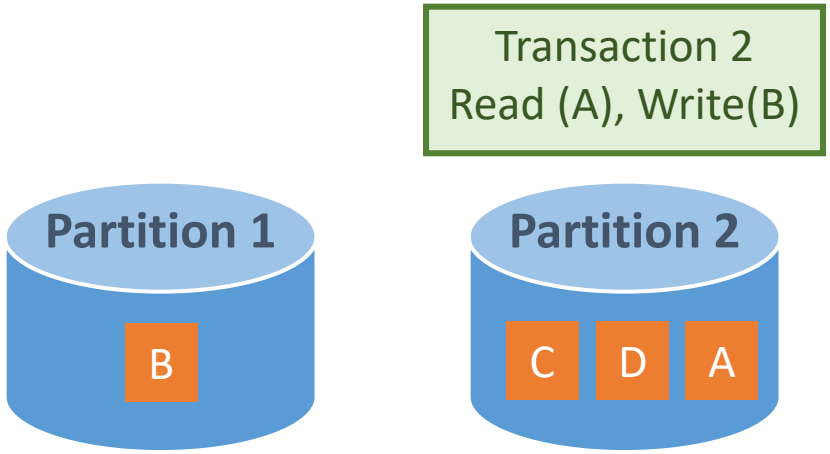
Look-Present Approaches

- Key Idea: uses **data-fusion** to merge/migrate data for the **current** transaction on the fly.
  - E.g., G-Store[SoCC'10], LEAP[SIGMOD'16]
- Benefit from temporal locality.



Problem: Data Accumulation

- If the system always routes transactions based on where the data are, it will be easy for the data to accumulate in a single partition/machine.
- Bad for load balancing.



Moreover,
all these approaches do not consider how to fit data partitions for **future** workloads.

They cannot react to unpredictable workloads in time!

Outline

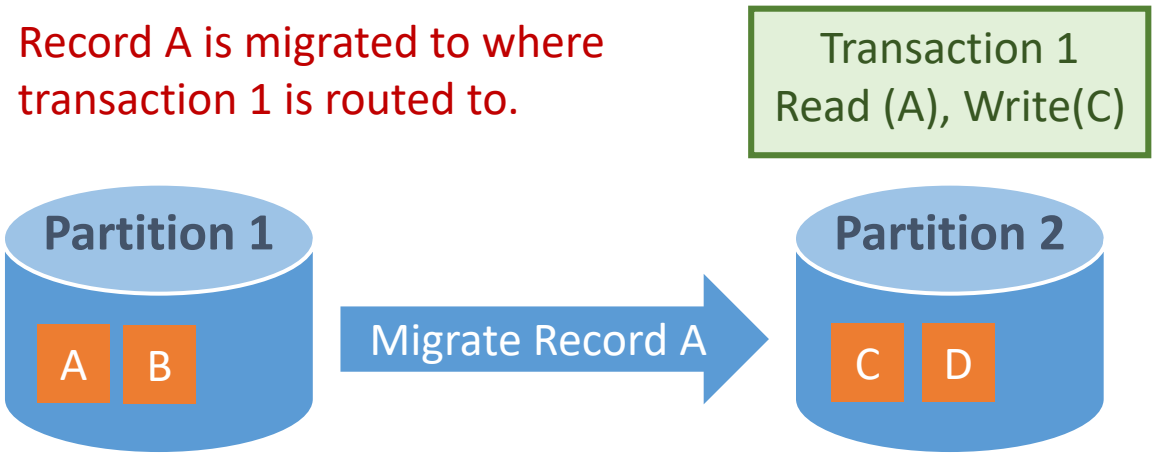
- Background
  - When a deterministic DBMS meets dynamic workloads
- Related Work
  - Look-Back Approaches
  - Look-Present Approaches
- Hermes
  - Key Observation
  - Key Idea
  - The Prescient Routing Algorithm
- Experiment Results
- Conclusion

Hermes

A joint design that address all above issues in a **single** module with a **single** technique.

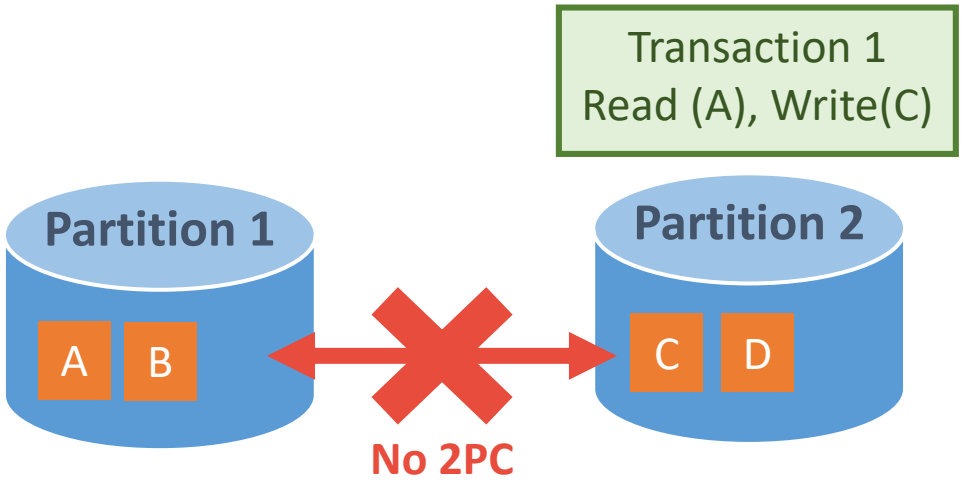
Key Observation: Data Re-partitioning by Transaction Routing

- With data-fusion technique, we can guide data flowing to where we want **by simply routing transactions** to proper locations.



Data Fusion on a Deterministic DBMS = Lightweight Data Migrations

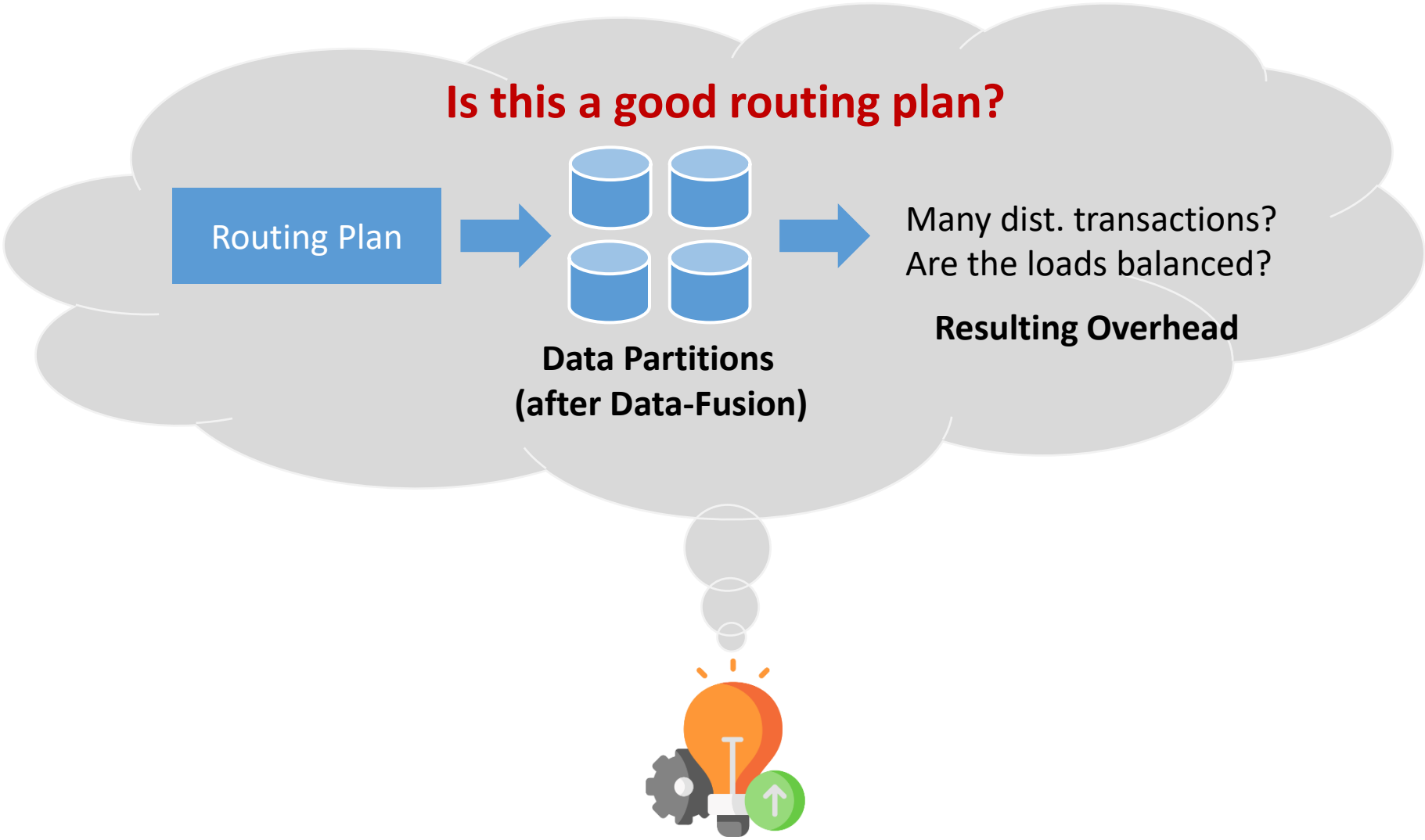
- Because the database state is deterministic, data-fusion is **much more lightweight** in deterministic DBMS than in a traditional DBMS.



How Do We Leverage On Data-Fusion To Help The System React to Dynamic Workloads?

Key Idea: Look into the Future

- Instead of routing a transaction request at a time, **we batch requests and route all of them at once**.
- Why? The router can analyze the dependencies between future transactions to **foresee possible data partitions** resulted from routing plans.
  - So that it can find the best routing plan that leads to the best data partitions.
- We call it **the prescient routing**.



The Prescient Transaction Router

What Exactly is the Routing Algorithm?

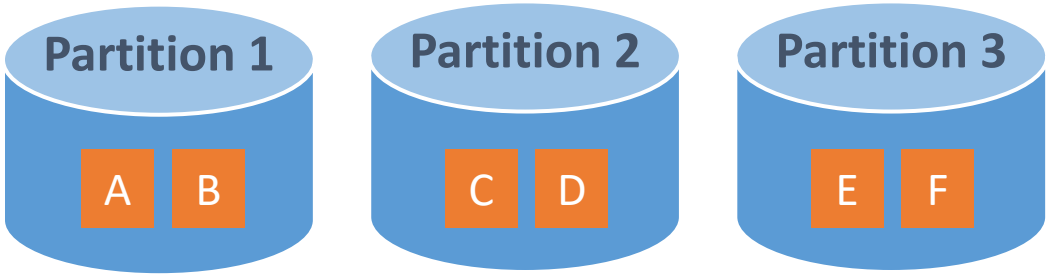
Our objective for good data partitions:
- The loads are balanced among the machines
- The number of distributed transactions are minimized

Step 1: Routes Transactions Such That # of Dist. Transactions Are Minimized

- The router may reorder transaction requests for better performance.

T1: Read {C}, Write {C}
T2: Read {C}, Write {C}
T3: Read {D}, Write {D}
T4: Read {D}, Write {D}
T5: Read {A, B, E}, Write {A, E}
T6: Read {A, B, E}, Write {A, E}

A Batch of Requests

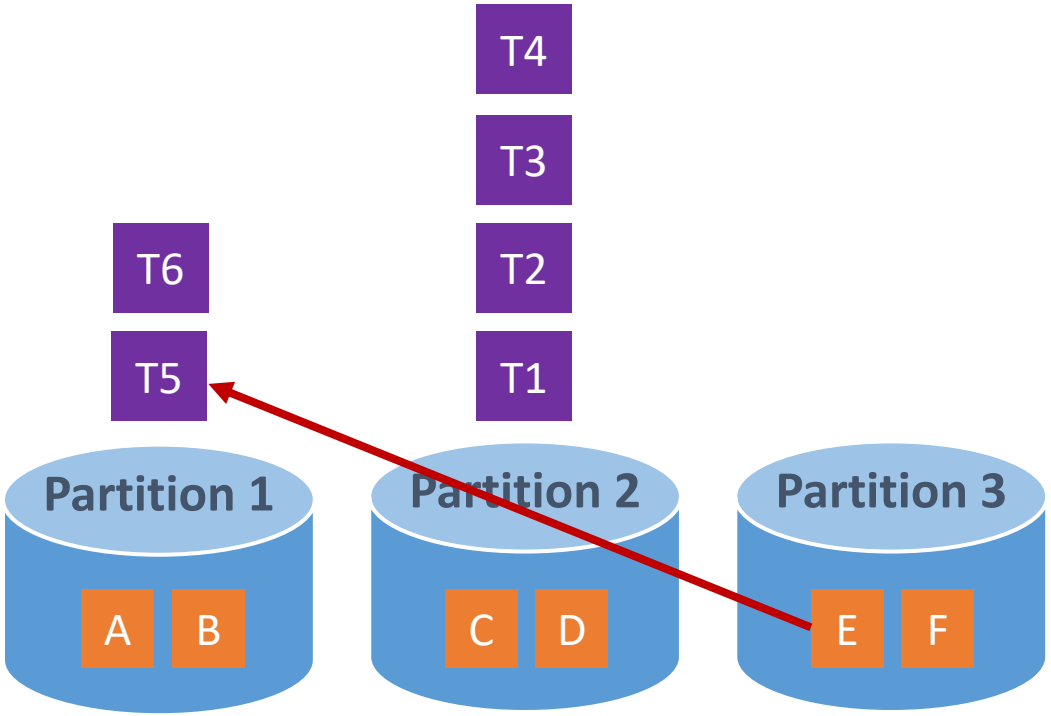


of Dist. Transactions: 1

- Note that T5 will migrate record E to partition 1 because of data fusion.

T1: Read {C}, Write {C}
T2: Read {C}, Write {C}
T3: Read {D}, Write {D}
T4: Read {D}, Write {D}
T5: Read {A, B, E}, Write {A, E}
T6: Read {A, B, E}, Write {A, E}

A Batch of Requests

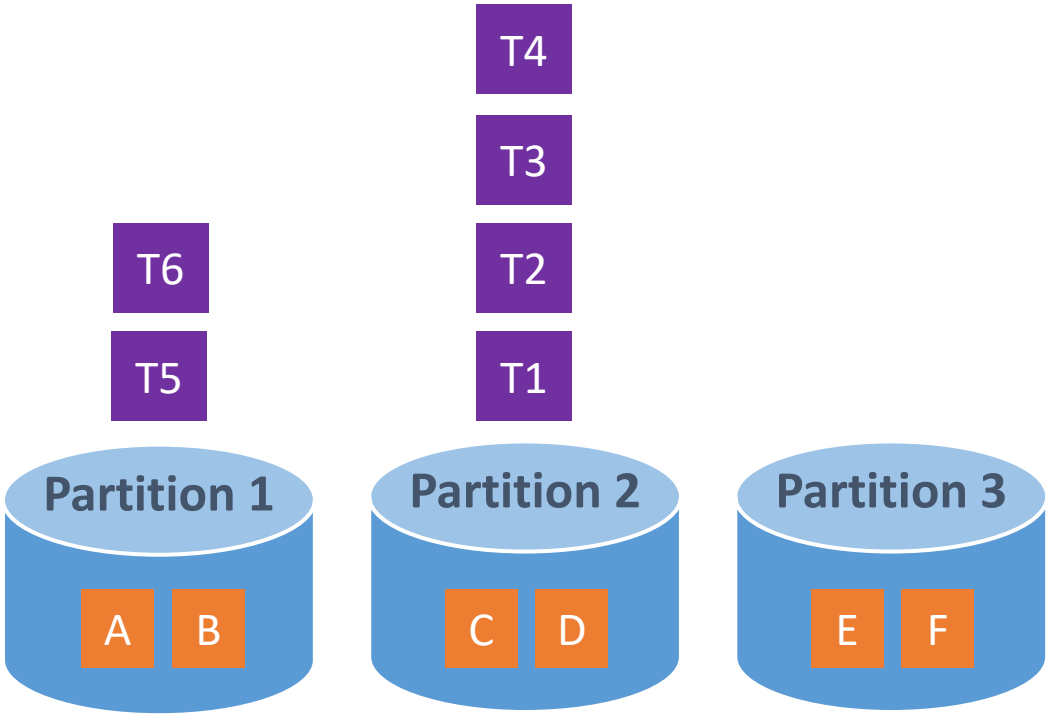


Step 2: Tries to Reroute Transactions to Balance Loads

- With a relaxed condition: allow to create **one more** distributed transaction.

T1: Read {C}, Write {C}
T2: Read {C}, Write {C}
T3: Read {D}, Write {D}
T4: Read {D}, Write {D}
T5: Read {A, B, E}, Write {A, E}
T6: Read {A, B, E}, Write {A, E}

A Batch of Requests

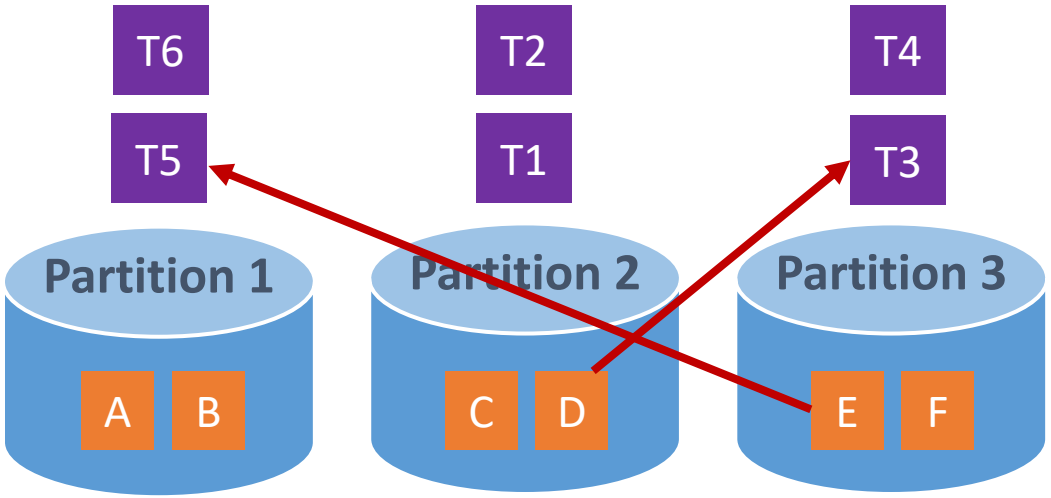


of Dist. Transactions: 2

- The load are balanced. ✓
- Not creates more than one dist. transaction. ✓

T1: Read {C}, Write {C}
T2: Read {C}, Write {C}
T3: Read {D}, Write {D}
T4: Read {D}, Write {D}
T5: Read {A, B, E}, Write {A, E}
T6: Read {A, B, E}, Write {A, E}

A Batch of Requests



What If It Cannot Find a Balanced Schedule Under This Condition?

Then, we **relax** the condition more until it finds one.

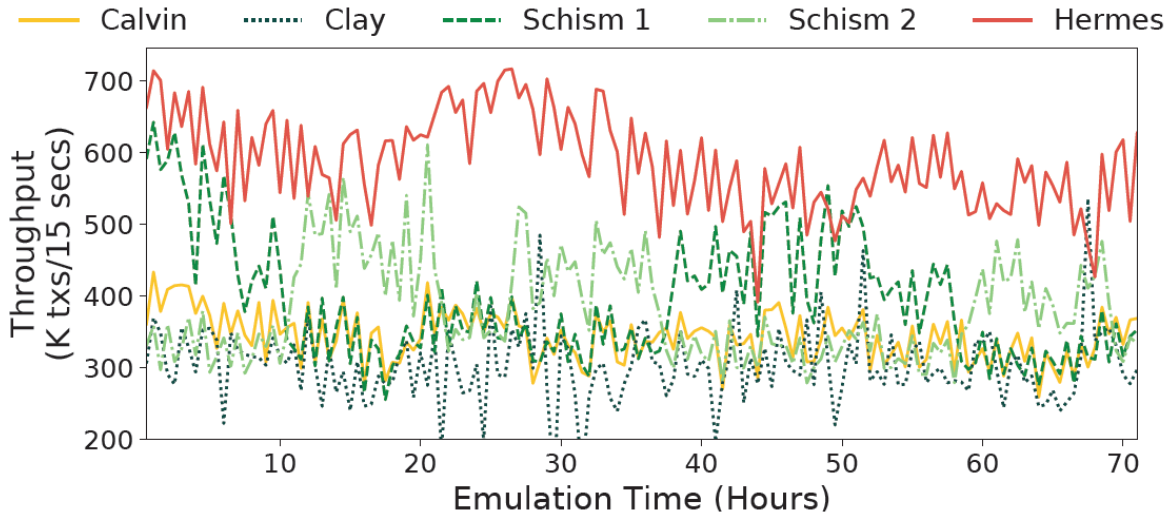
Please Check Our Paper for More Information!

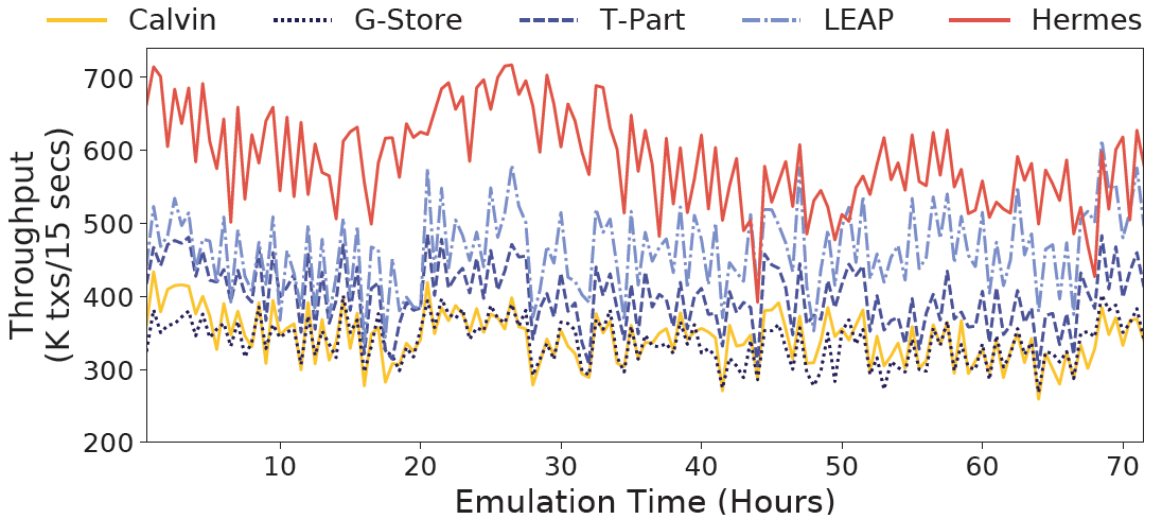
- How to maintain data partitioning across machine without additional cost?
- How to work with dynamic machine provisioning?
- What if a transaction aborts?

Outline

- Background
  - When a deterministic DBMS meets dynamic workloads
- Related Work
  - Look-Back Approaches
  - Look-Present Approaches
- Hermes
  - Key Observation
  - Key Idea
  - The Prescient Routing Algorithm
- Experiment Results
- Conclusion

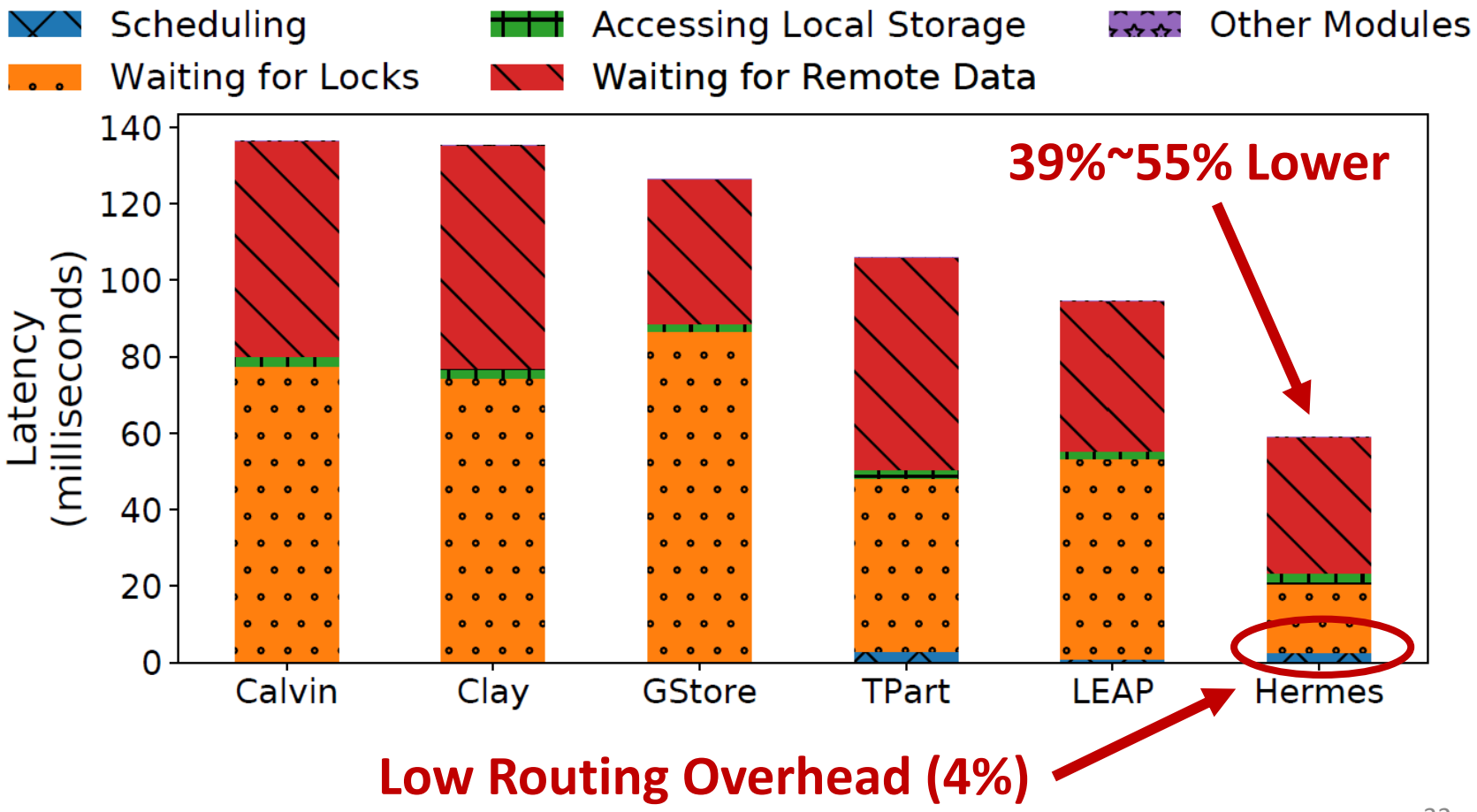
The Overall Performance



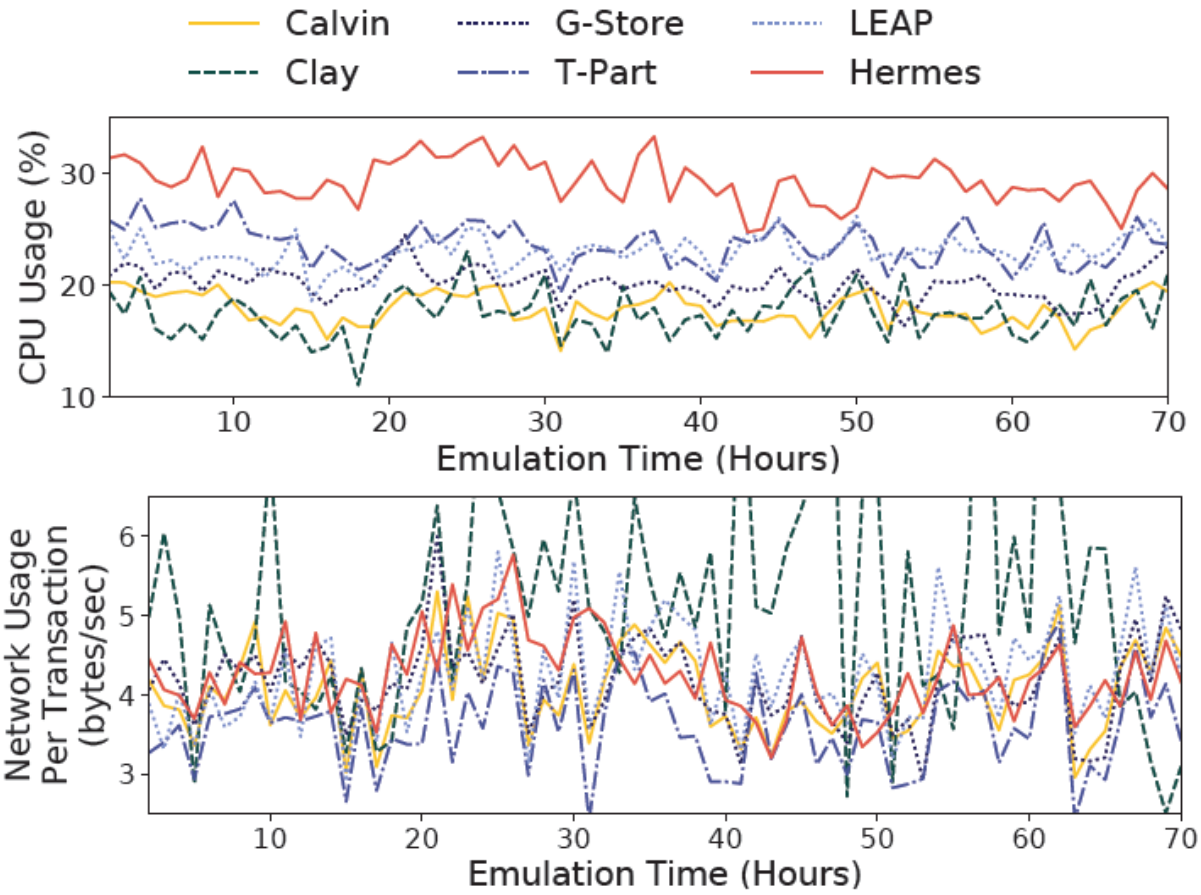


Outperforms all the baselines

Latency Breakdown



Resource Utilization and Consumption



CPU Utilized More

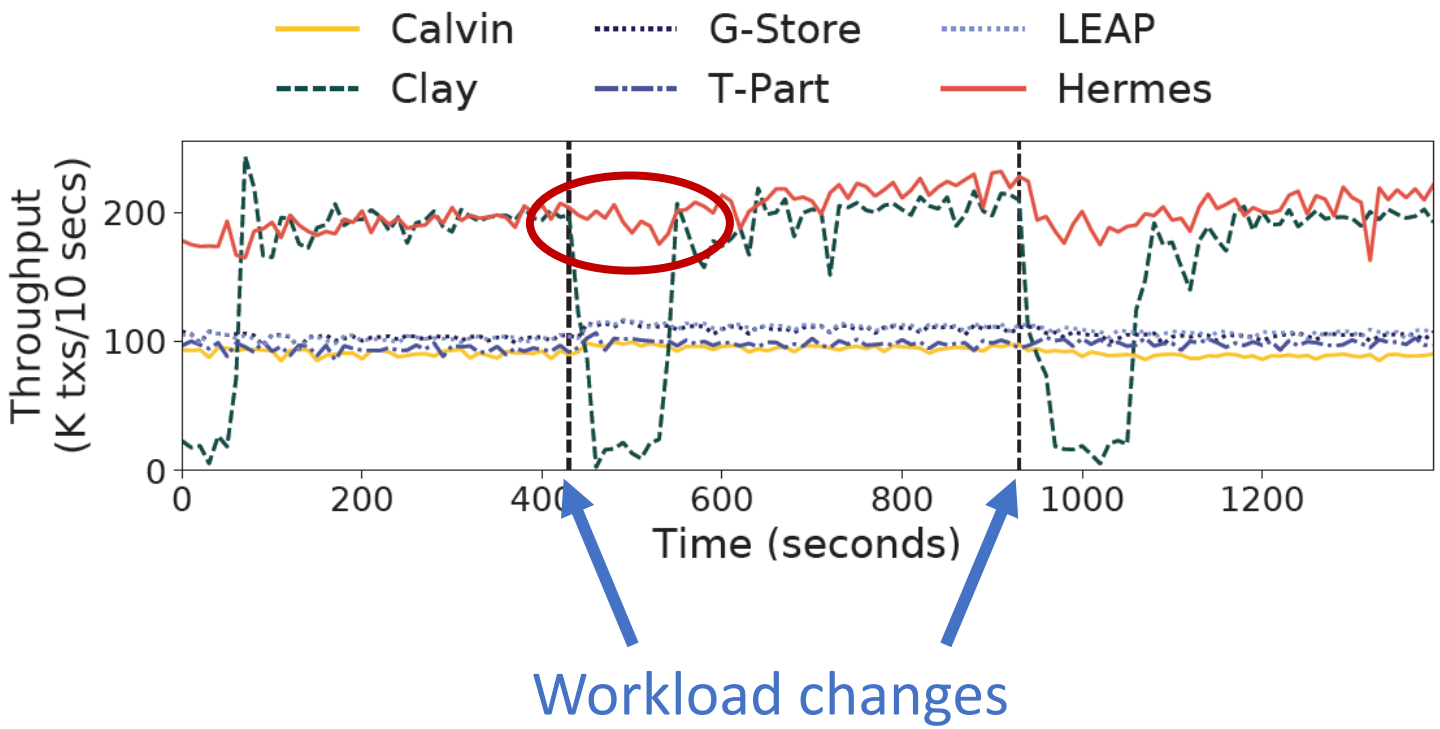
Not Introduce Additional Network Overhead

Simple Hotspot Workloads

A workload consisting of a hotspot that receives 90% of workloads.

The hotspot changes every 450 seconds.

Hermes Achieves Fast Adaptation and Lightweight Migration



Outline

- Background
  - When a deterministic DBMS meets dynamic workloads
- Related Work
  - Look-Back Approaches
  - Look-Present Approaches
- Hermes
  - Key Observation
  - Key Idea
  - The Prescient Routing Algorithm
- Experiment Results
- Conclusion

Conclusion

- We proposed Hermes based on the following key observations:
  - Transaction routing with data fusion controls data partitions.
  - Looking into the future by batching transaction requests.
- We showed the effectiveness of Hermes on both complex and simple dynamic workloads, which shows how fast Hermes adapts to new workloads.

Q&A

- Please contact me if you have more questions
  - yslin@datalab.cs.nthu.edu.tw
- Slides available at