Microsoft Research

Microsoft

Optimizing Optimistic Concurrency Control for Tree-Structured, Log-Structured Databases

Philip A. Bernstein (Microsoft Research)
Sudipto Das (Microsoft Research)
Bailu Ding (Cornell University)
Markus Pilman (ETH Zurich)

Hyder: Scale-out OLTP w/o partitioning

Scale-out OLTP usually requires partitioning the DB

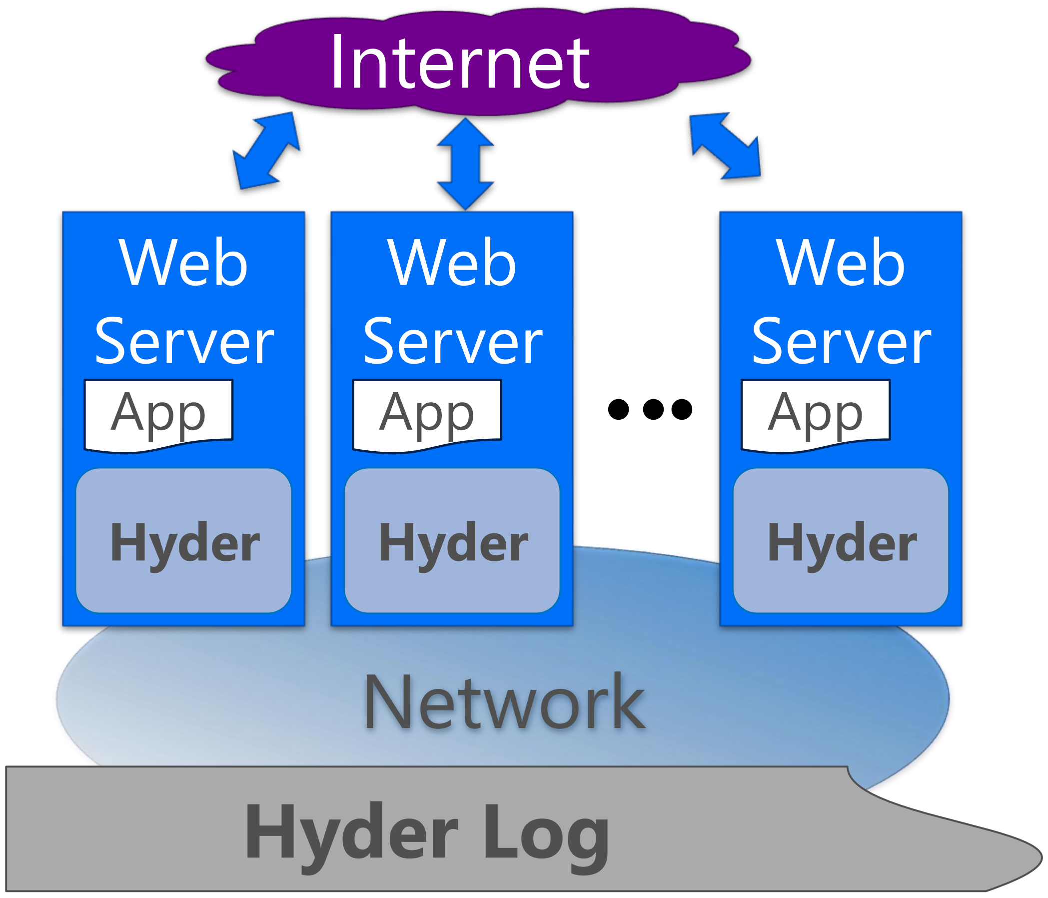
- Partitioning the DB is hard

Data sharing architectures

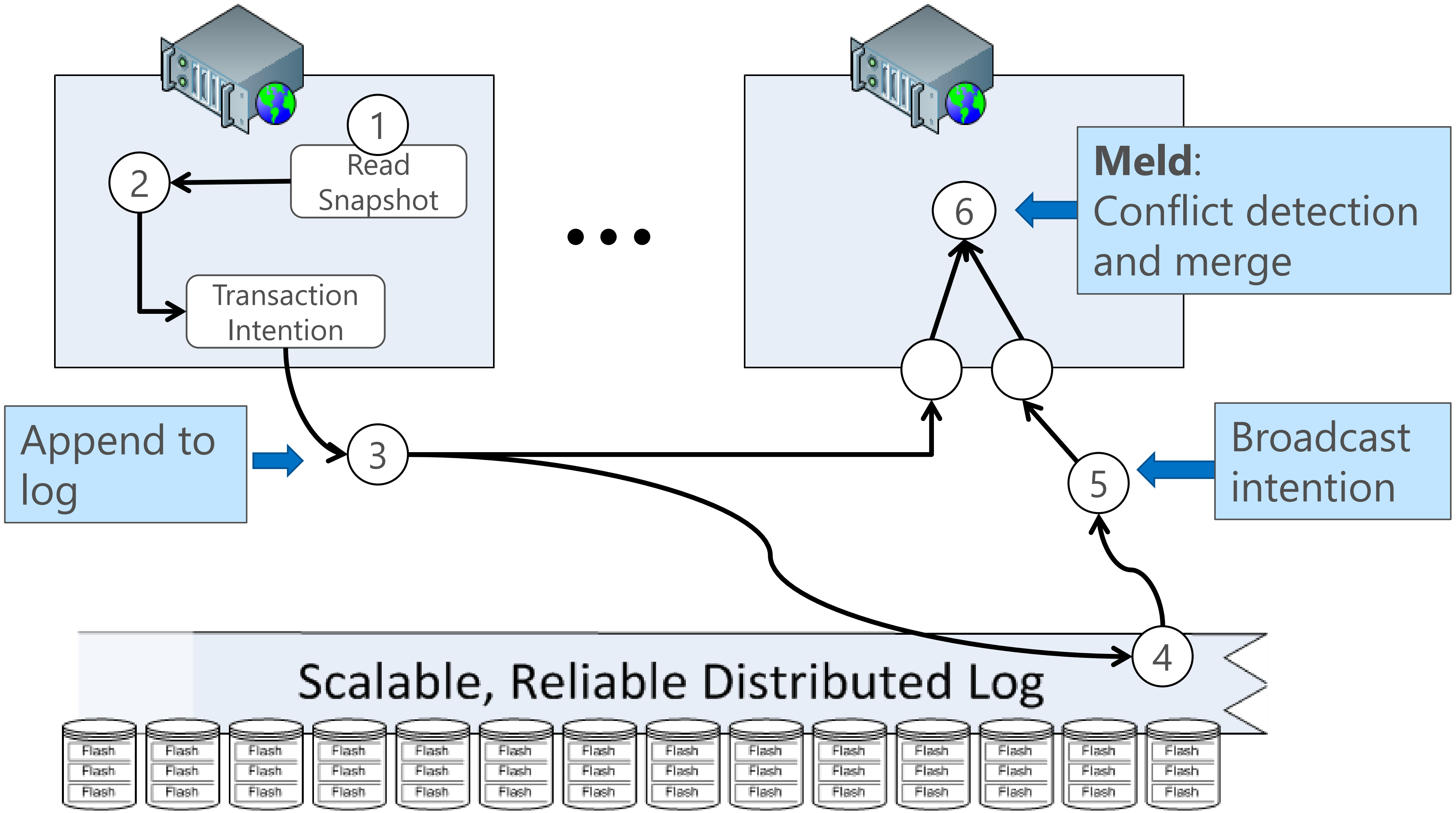
- The entire database is accessible by all servers that can run transactions
- Scales-out without partitioning

Hyder

- The log is the database
- All servers roll-forward the log



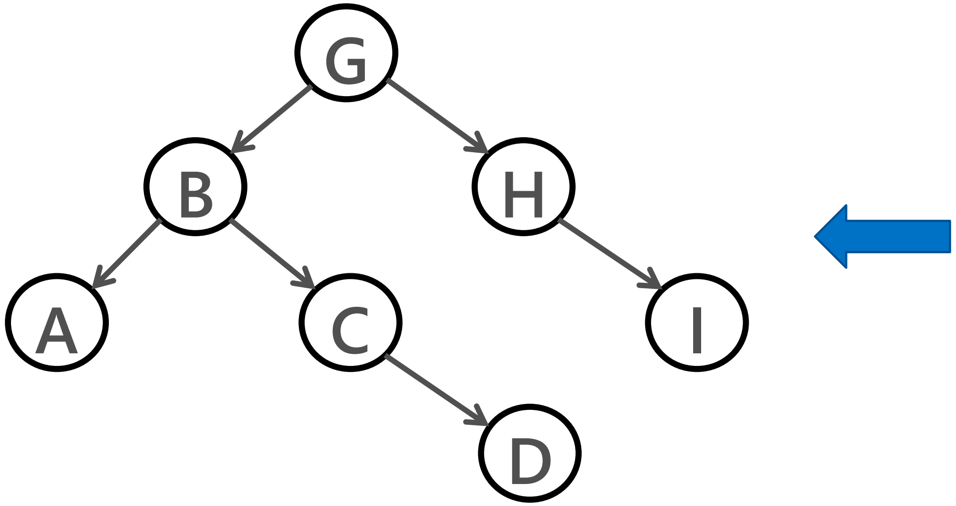
Life of a transaction in Hyder



Database is a search tree

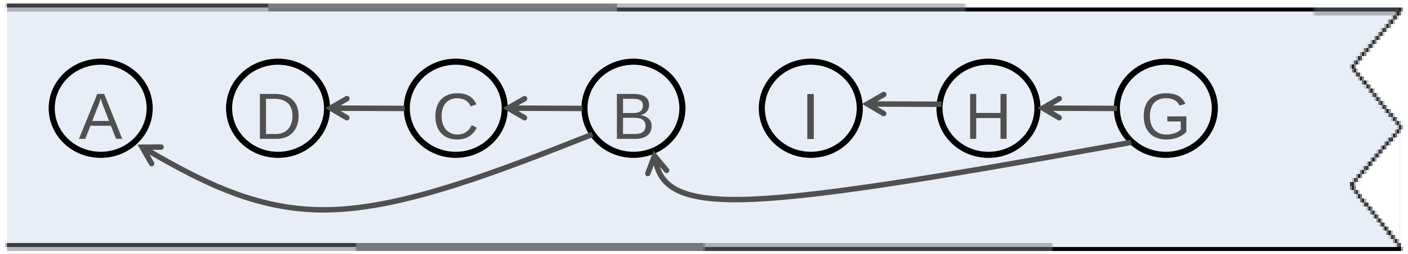
In this paper, it's a balanced binary search tree (AVL or Red-Black)

Tree is serialized into a network-attached *shared log* stored on SSDs



Binary Search Tree

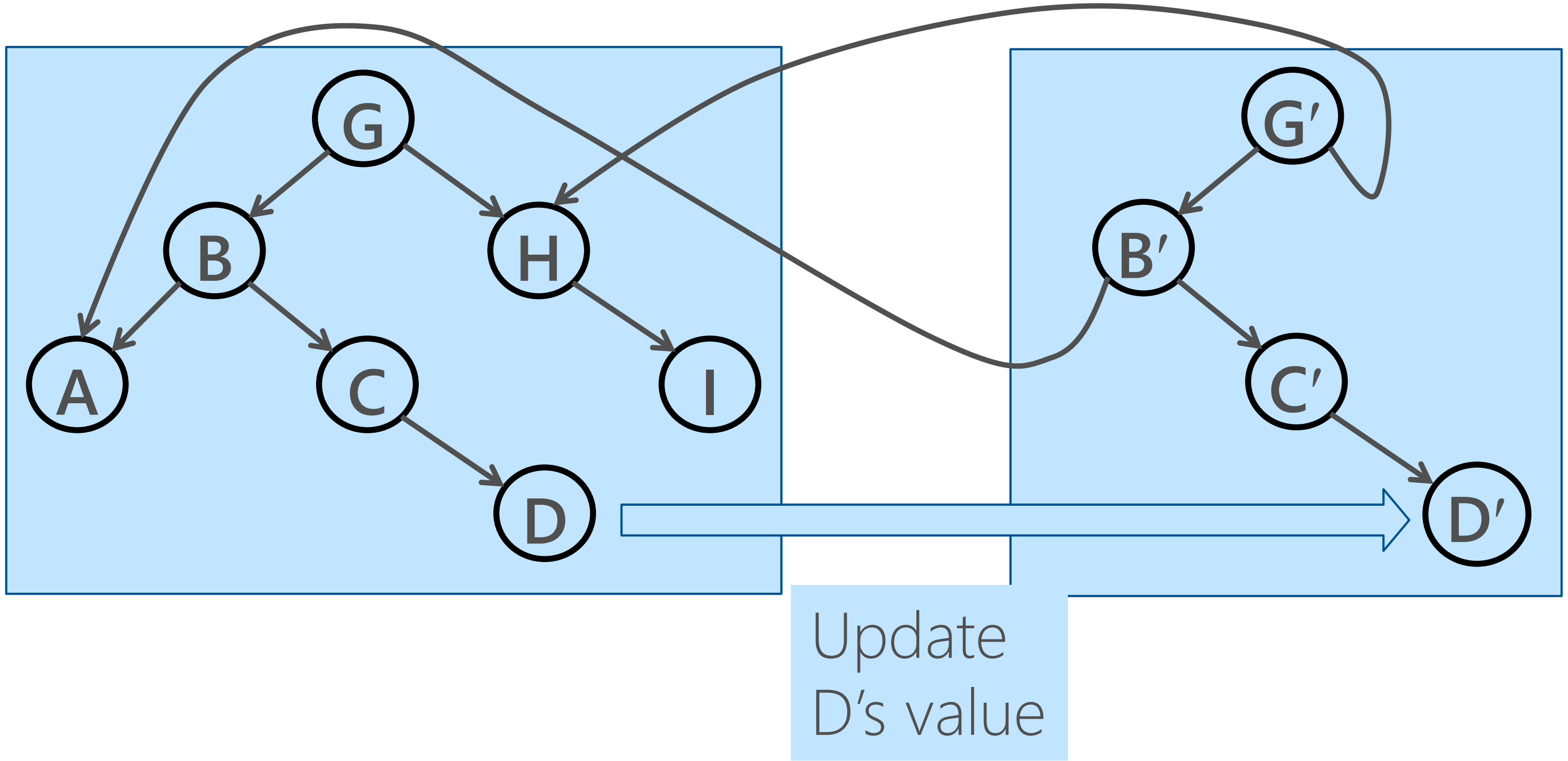
Tree is marshaled into the log



Database tree is multi-versioned

To update a node, copy its ancestors up to the root

Copy on write

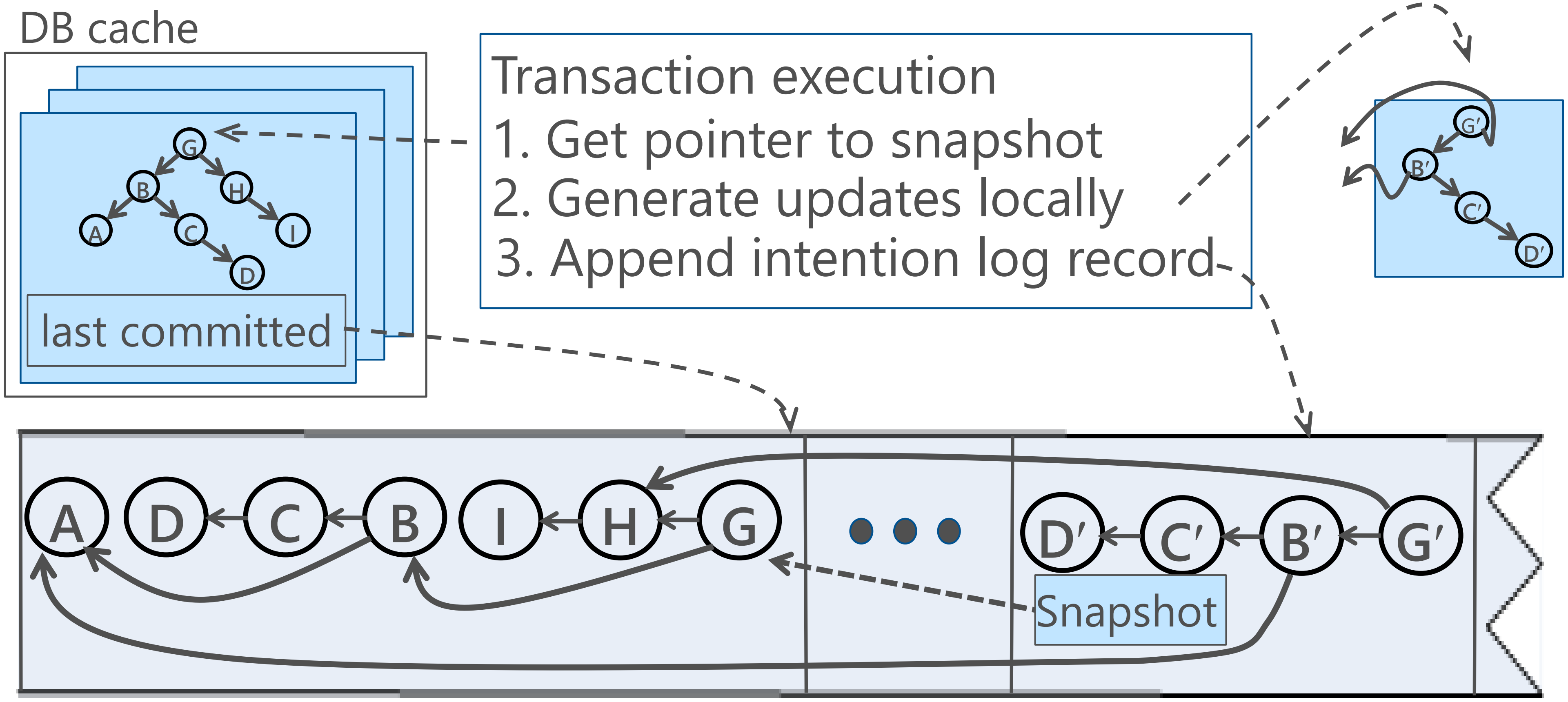


Transactions execute optimistically

Each server has a *cache* of the last committed database state

A transaction executes *optimistically*

Reads a snapshot, creates an *intention*, and *appends* it to the log



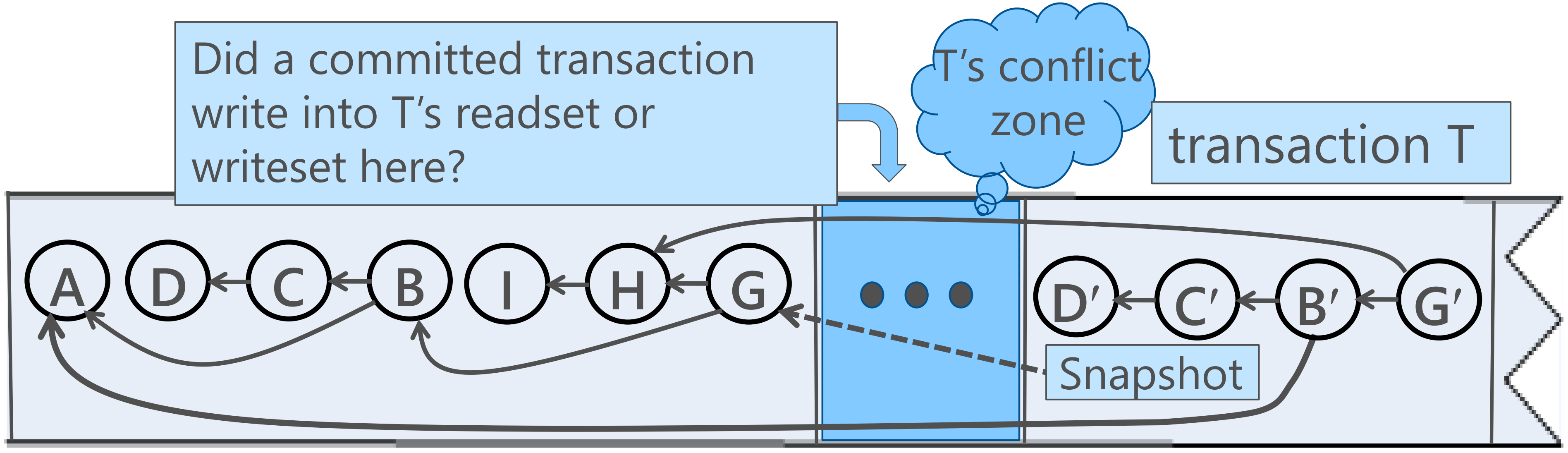
Meld validates and merges intentions

Meld - *sequentially* roll-forward transactions in log order

For each intention log record ***I*** *for transaction* ***T***,

- check whether ***T*** experienced a conflict
- if not, ***T*** committed, so merge ***I*** into server's last committed state
- ***efficient*** conflict detection using metadata in ***I*** and last committed state

Determinism – All servers make the same commit/abort decisions



Hyder's evolution

Simulation model presented at CIDR 2011

C++ main-memory implementation presented at VLDB 2011

Single-machine transactional file system within Microsoft

All prior implementations were on *a single server*

Hyder II, *distributed* implementation with a *highly-optimized* meld

- Written from scratch in C# in 2013-2014
- CORFU as the distributed shared log [Balakrishnan et al. NSDI 2012]

Hyder II

Major Learnings

Solved many issues that limited performance

Studied performance for a variety of workloads

Bottlenecks in Hyder

- Log append throughput
- Network bandwidth
- Meld throughput
- Data contention and optimistic concurrency control

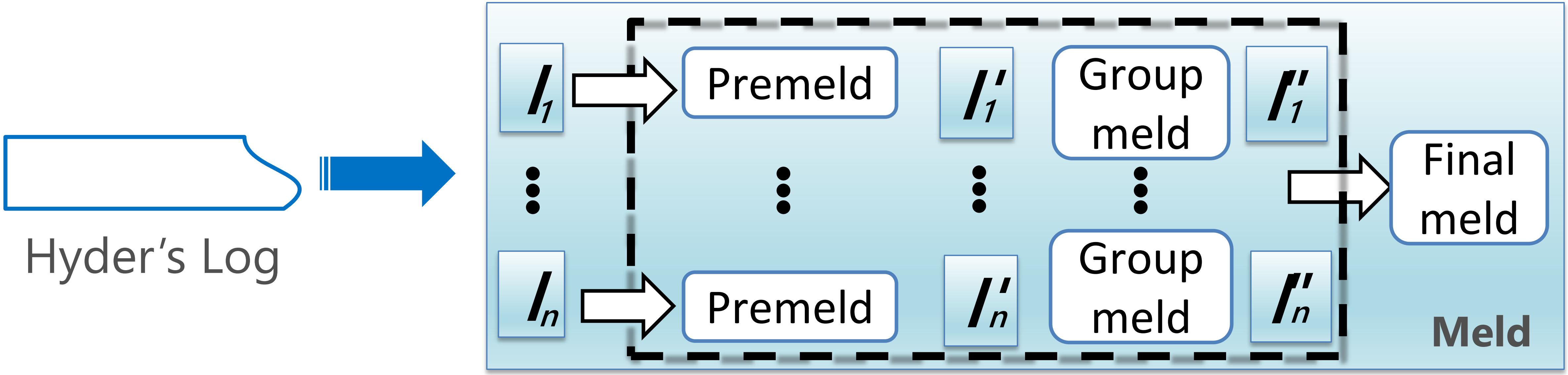
Optimized meld to ***increase peak update transaction throughput by 3X*** across a variety of workloads

Hyder II: Optimizing Meld

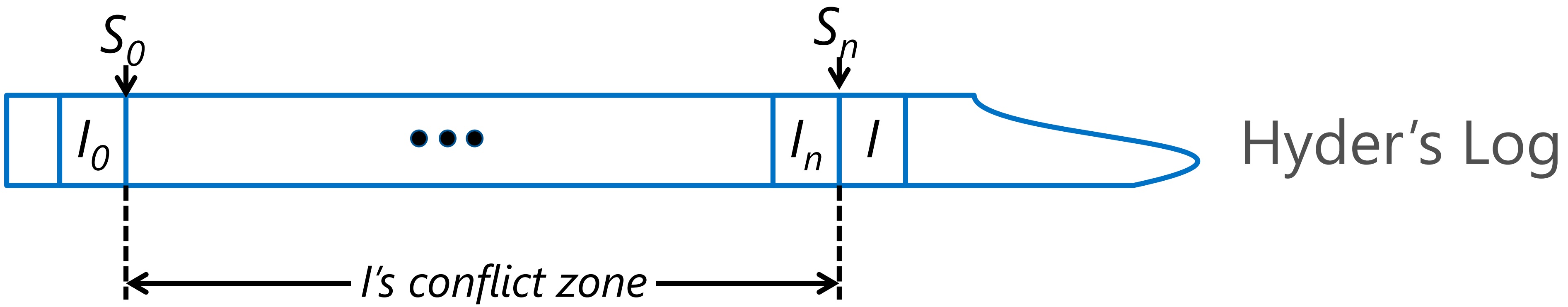
Stagnant CPU speeds limit meld's throughput

Parallelize meld via **pipelining** by adding two **deterministic** preliminary stages when melding in intention

- **Premeld**: Merge intention with a *recent snapshot* leaving very little for the sequential final meld
- **Group meld**: Merge two intentions into one

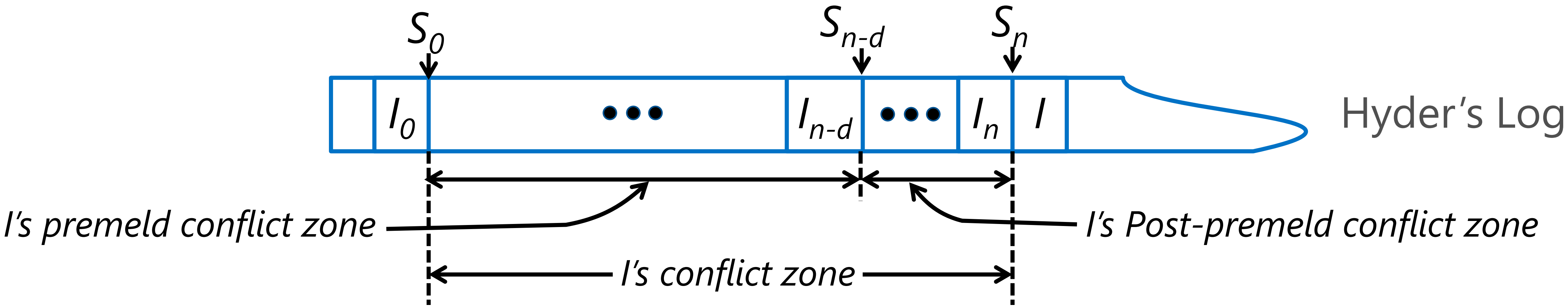


Meld as an operator



- Meld merges I to S_n
- At high transaction loads, **tens of thousands** of transactions appear in I's conflict zone
- Can we model meld as an operator on trees and apply it repeatedly in parallel?
  - *Meld two trees and output a tree*

Premeld to a later snapshot



- Premeld *I* to a later snapshot S_{n-d} when *I* arrives at a server ***after*** serialization, append, and broadcast
- Premeld conflict zone is *~2 orders of magnitude smaller* than post-premeld conflict zone
- Significantly reduces the work to do a final meld of *I* to S_n
- Challenges: determinism, metadata for final meld

Experimental Results

Experimental Setup

Cluster of twenty commodity servers on 10Gbps network, and server-grade Intel SSDs

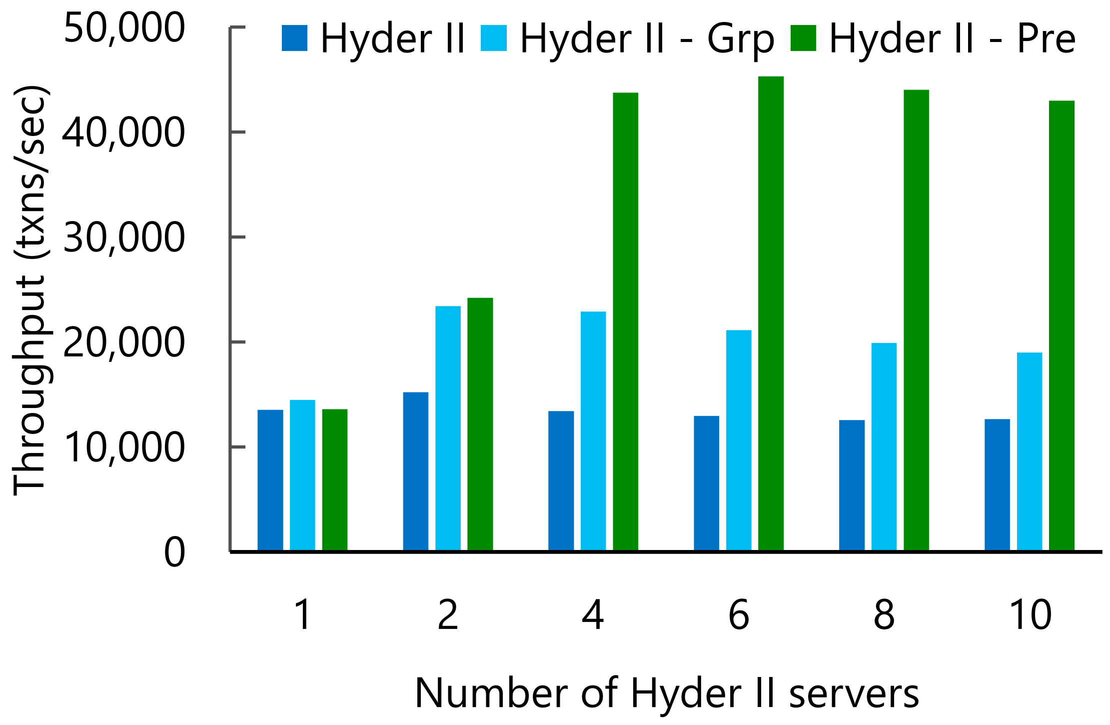
Workload generator derived from YCSB

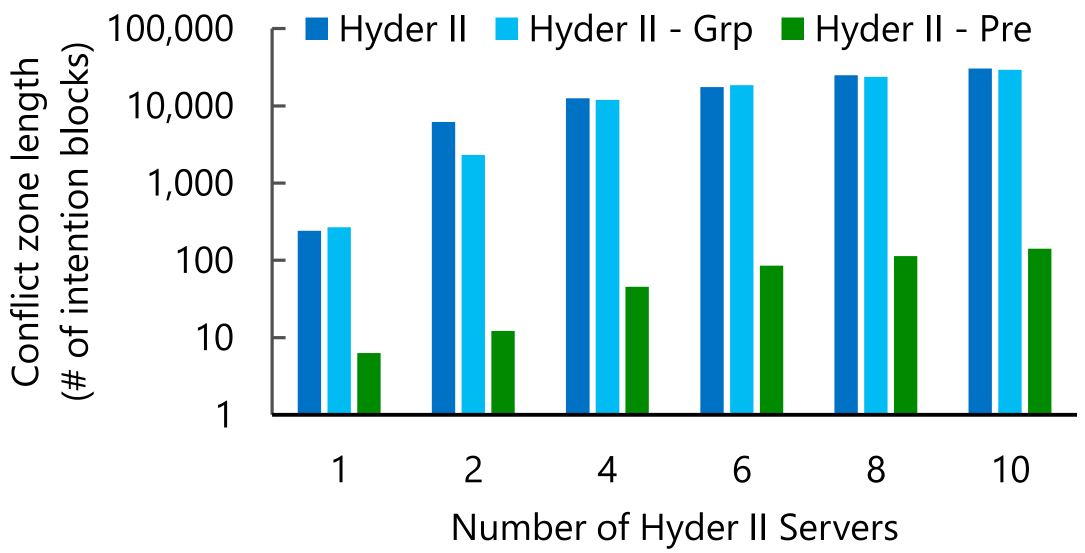
- Multi-operation transactions
- Database of 10M items, each item about 1K

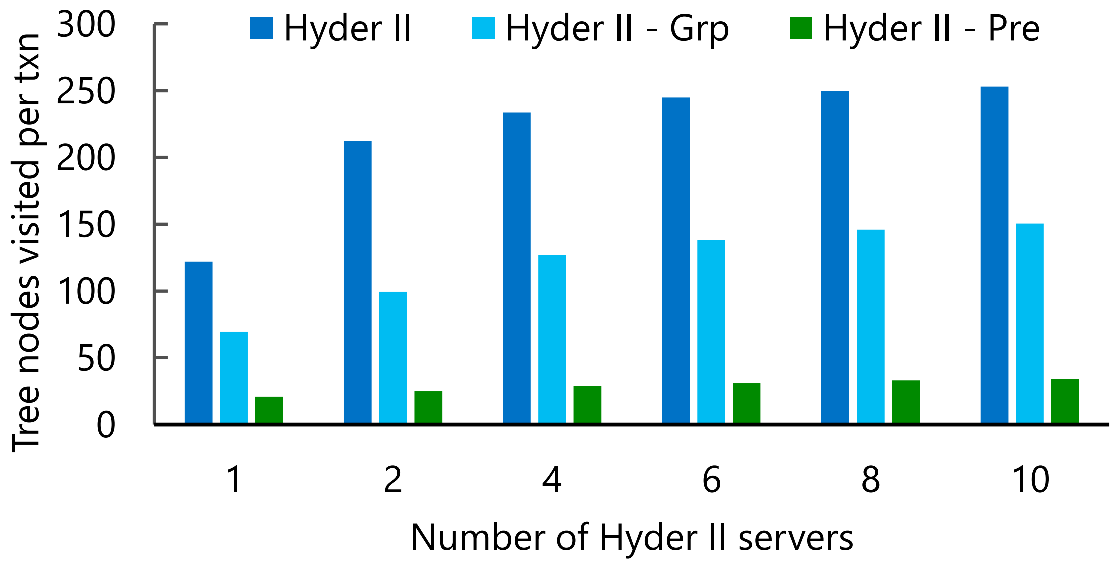
Various workload parameters varied

- No. of operations per transaction, default 10
- No. of write operations per transaction, default 2
- Isolation level, default Serializable
- Data distribution, default in Uniform
- Database size …

Workload with all Write Transactions

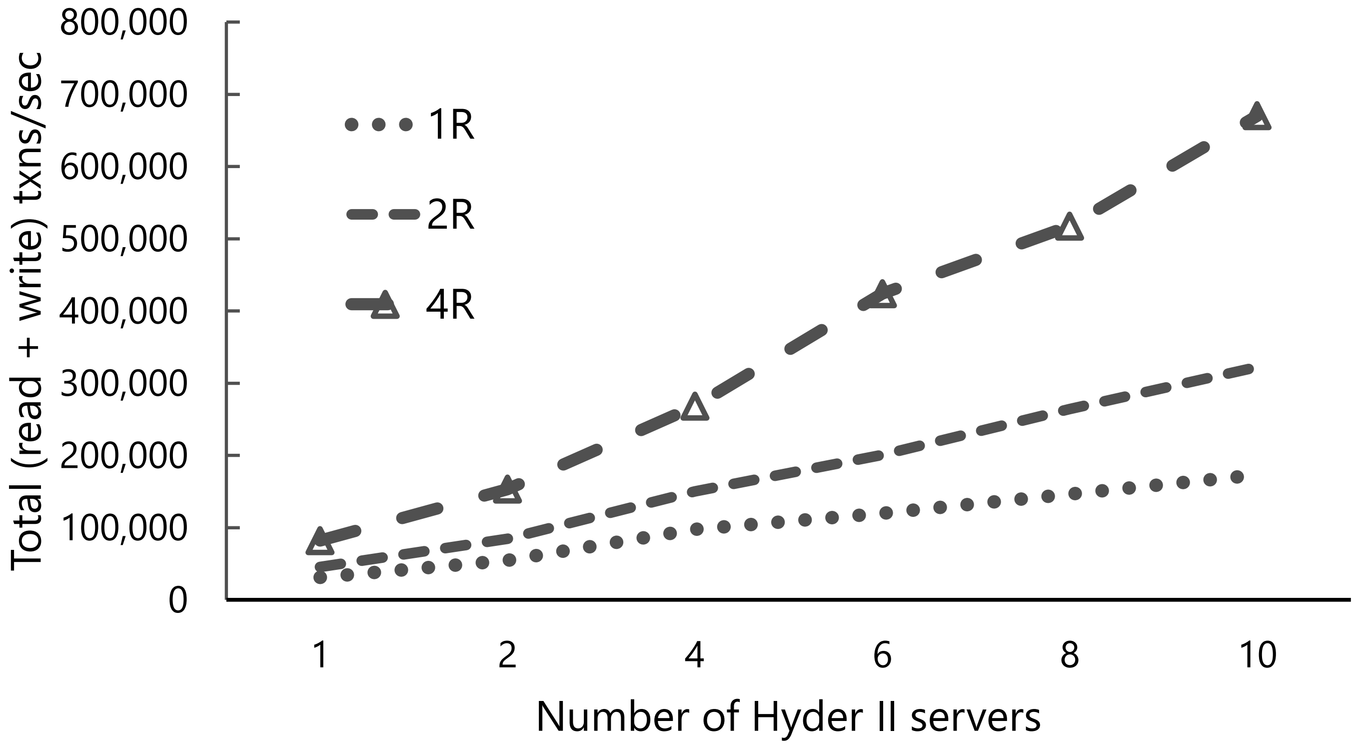






Group meld: ~1.5X improvement,
Premeld: ~3X improvement

Read-Write Transaction Mix



Conclusion

Hyder: a novel architecture for scaling-out transactions without partitioning the database

An end-to-end implementation of Hyder

Hyder II: Pipelined parallelism to optimize meld

Analyzed behavior under a variety of workloads

- Read-only transactions scale almost linearly
- Premeld improves throughput by 3X

Partitioned Hyder in IEEE Data Eng. Bulletin, March 2015

More information: http://aka.ms/hyder

Thank You!

Questions?