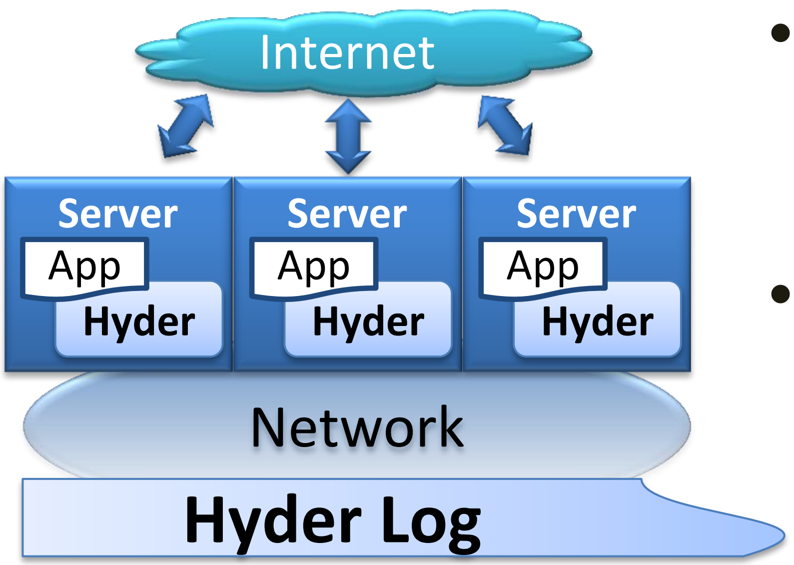
Hyder – A Transactional Record Manager for Shared Flash

Philip A. Bernstein, Microsoft Corporation
Colin Reid, Microsoft Corporation
Sudipto Das, UC Santa Barbara

CIDR 2011
January 10, 2011

Hyder: The Big Picture

- Goal: Enable scale-out without partitioning DB or app



- Store the whole DB in flash
  - which is accessible to all servers
  - via a fast data center network
- Main architectural features
  - Uses a log-structured DB in flash
  - Broadcast log to all servers
  - Roll forward log on all servers
  - Optimistic concurrency control

- There's no cross-talk between servers
  - Hence, Hyder scales-out without partitioning

What is Hyder?

An incubation, i.e. research project.

A software stack for transactional record management

- Stores [key, value] pairs, which are accessed within transactions

Functionality

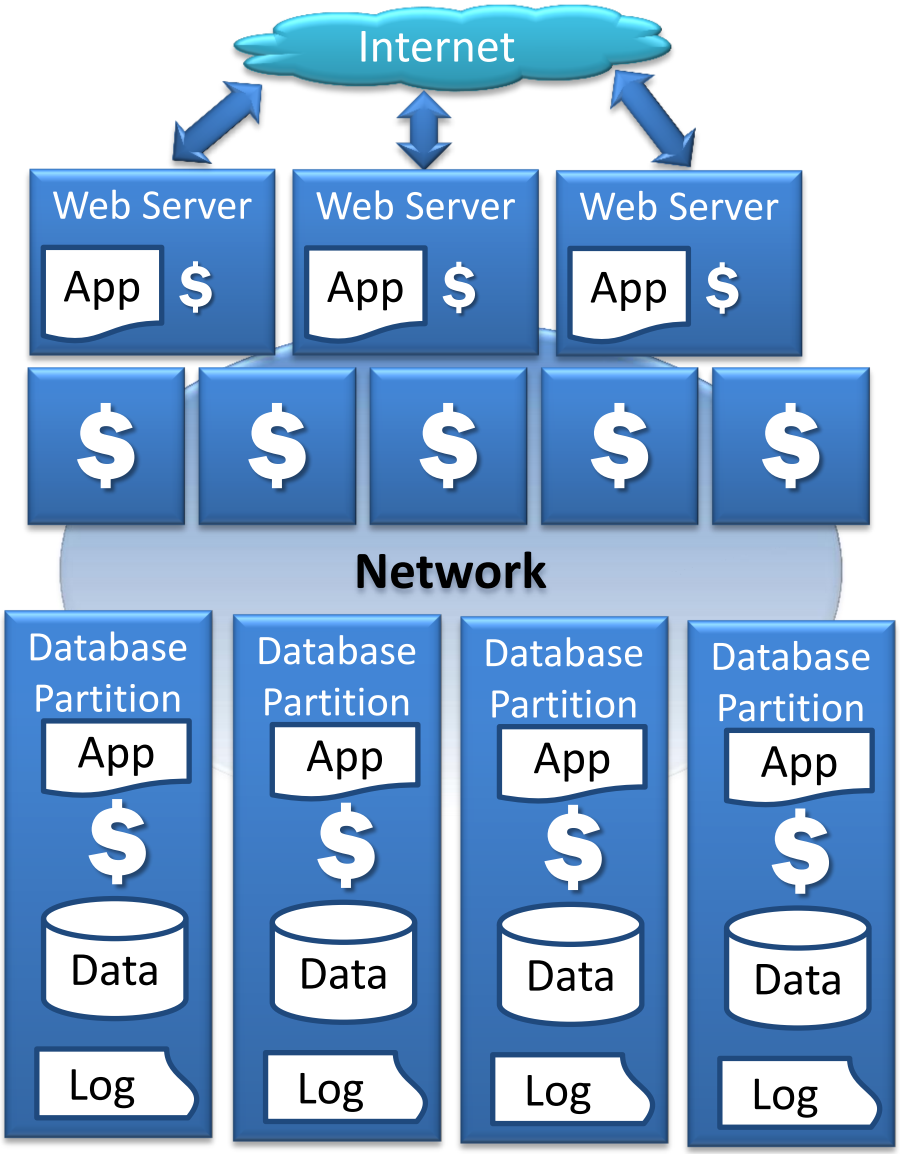
- Record operations:
  - Insert, Delete, Update, Get where field = X; Get next
- Transactions: Start, Commit, Abort

Why build another one?

- Exploit flash memory and high-speed networks

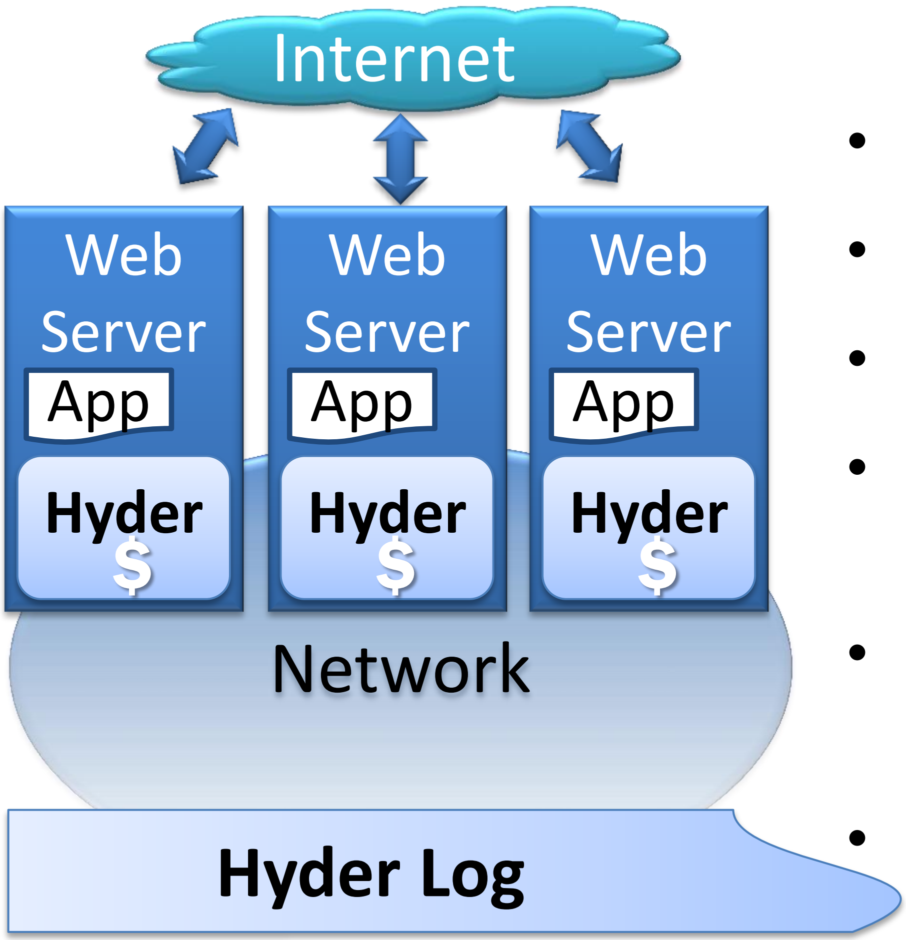
  to simplify scaling out large-scale web services

Scaling Out with Partitioning



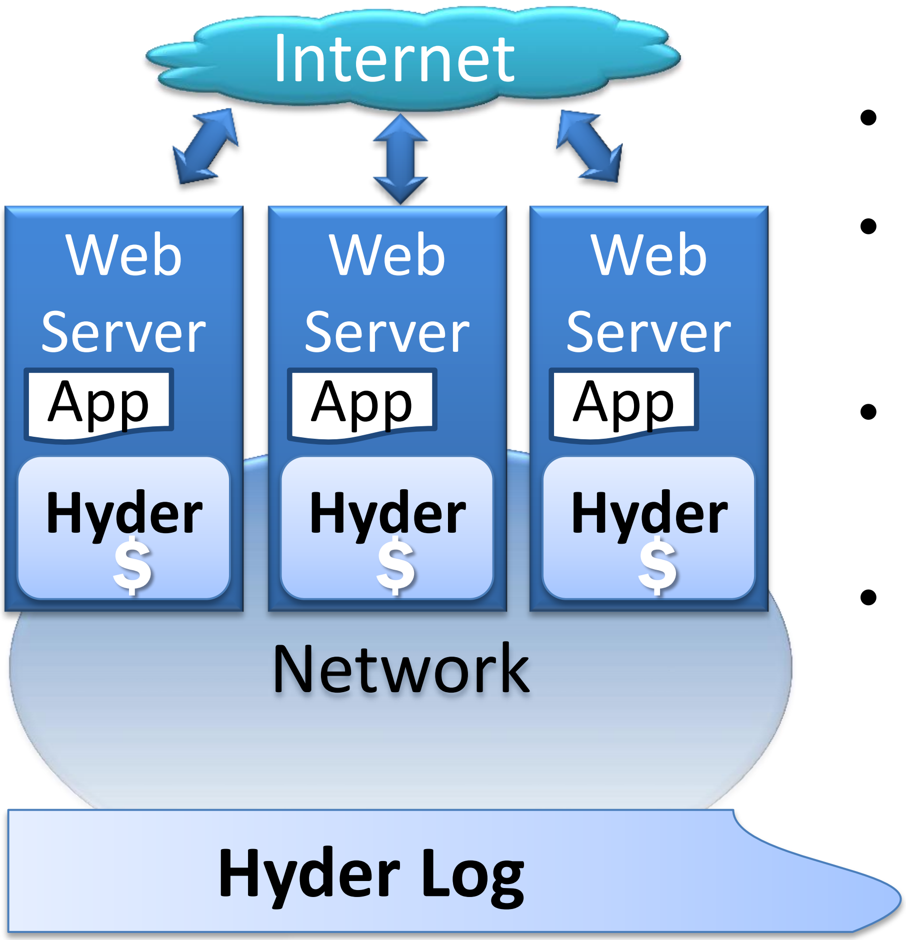
- Database is partitioned across multiple servers
- Each query is sent to the appropriate partition(s)
- For scalability, avoid distributed transactions
- Cross partition consistency is enforced in the application
- Hard to provision servers and distribute load evenly

Hyder Scales Out Without Partitioning



- In Hyder, the log is the database
- All servers can access the log
- No partitioning is required
- Database is multi-versioned, so server caches are trivially coherent
- Hence, can parallelize a query with consistency across servers
- And servers can fetch pages from the log or from neighboring servers’ caches

Hyder Runs in the Application Process

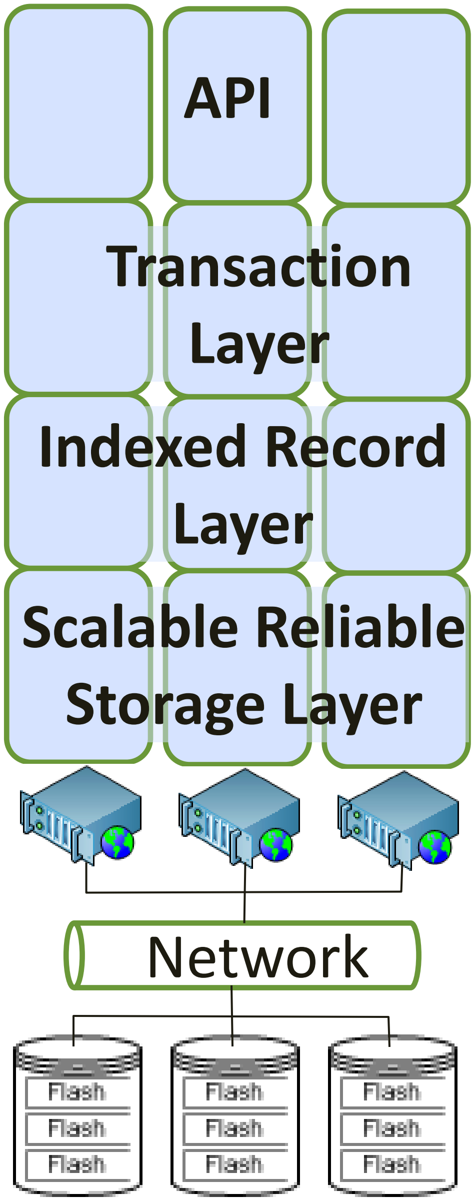


- No distributed programming
- No distributed caches for the app to keep consistent
- Avoids the expense of RPC's to a database server
- Simple high performance programming model

Enabling Hardware Assumptions

- Flash offers cheap and abundant I/O operations
  - $\Rightarrow$ Can spread the DB across a log, with less physical contiguity
- Cheap high-performance data center networks
  - $\Rightarrow$ Many servers can share storage, with high performance
- Large, cheap, 64-bit addressable memories
  - $\Rightarrow$ Reduces the rate that Hyder needs to access the log
- Many-core web servers
  - $\Rightarrow$ Hyder can afford to roll forward the log on all servers

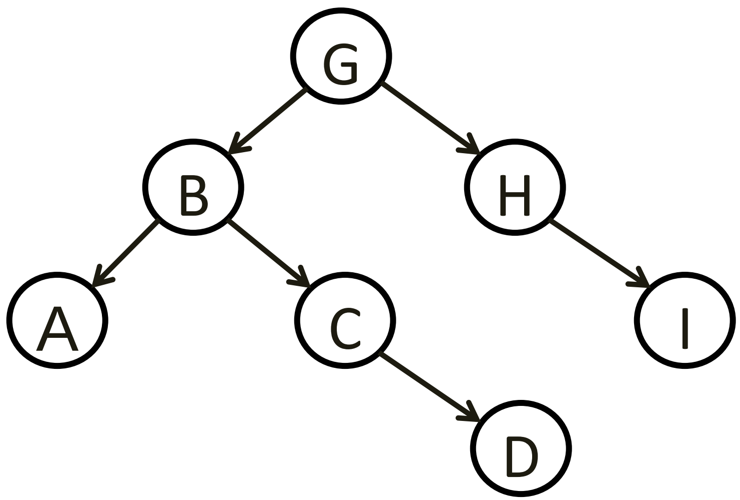
The Hyder Stack



- ISAM, SQL, LINQ, etc.
- Optimistic transaction protocol
- Multi-versioned search tree
- Segments, stripes and streams
- Append-only custom controller interface

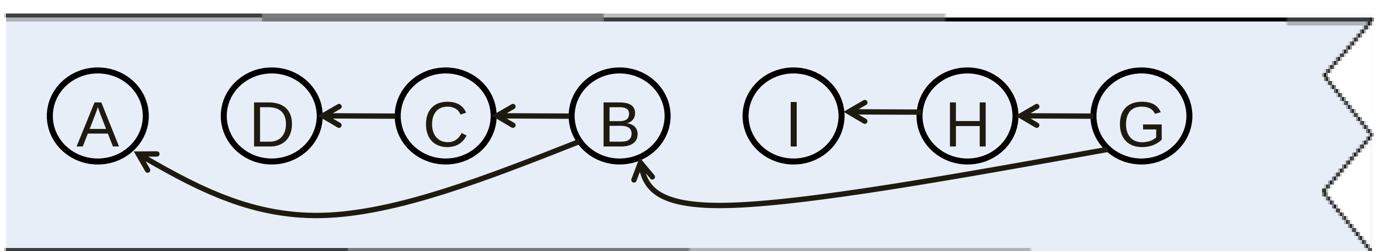
Database is a Search Tree

In this paper, it's a binary search tree.



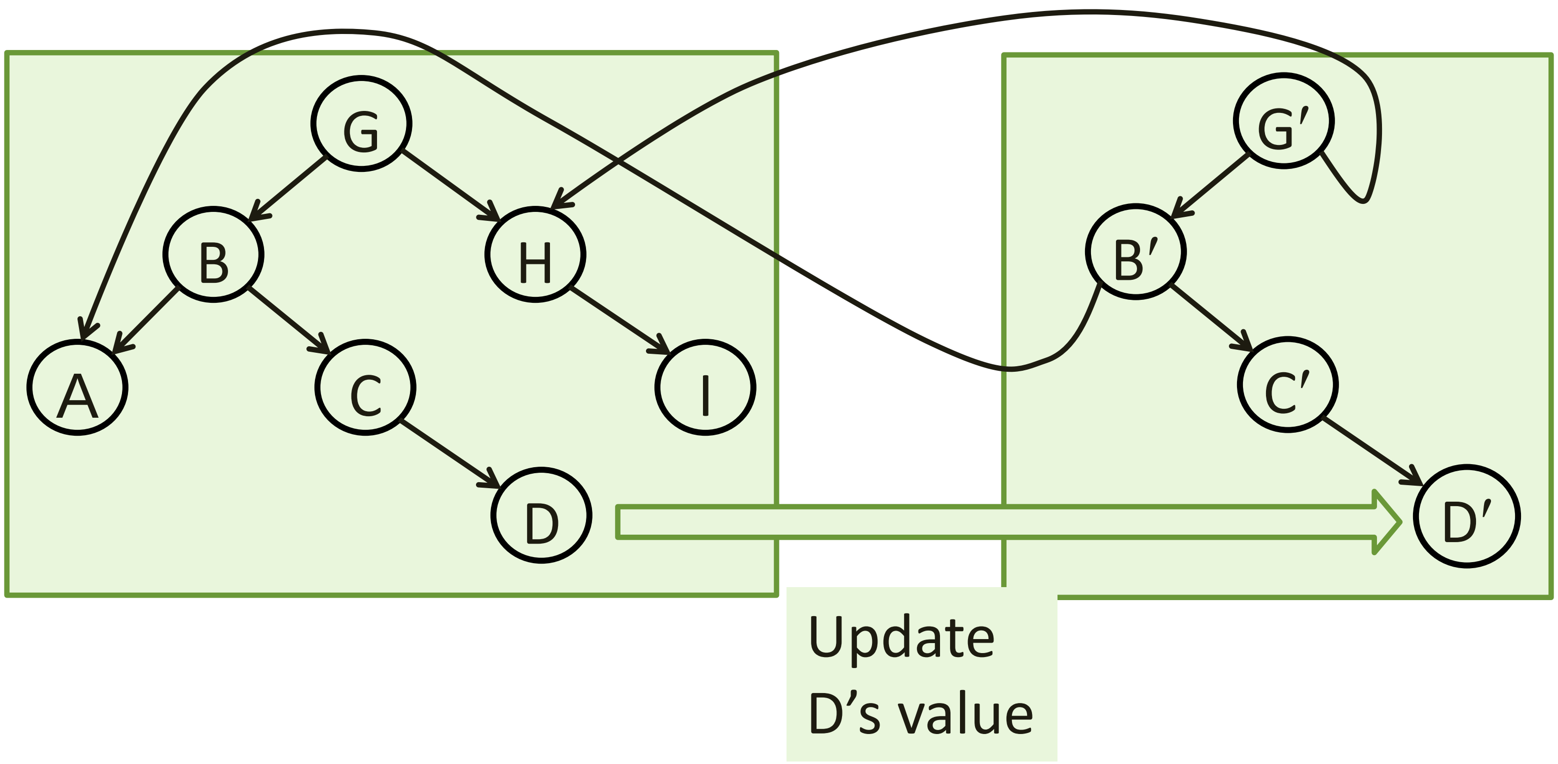
Binary
Search
Tree

Tree is marshaled into the log



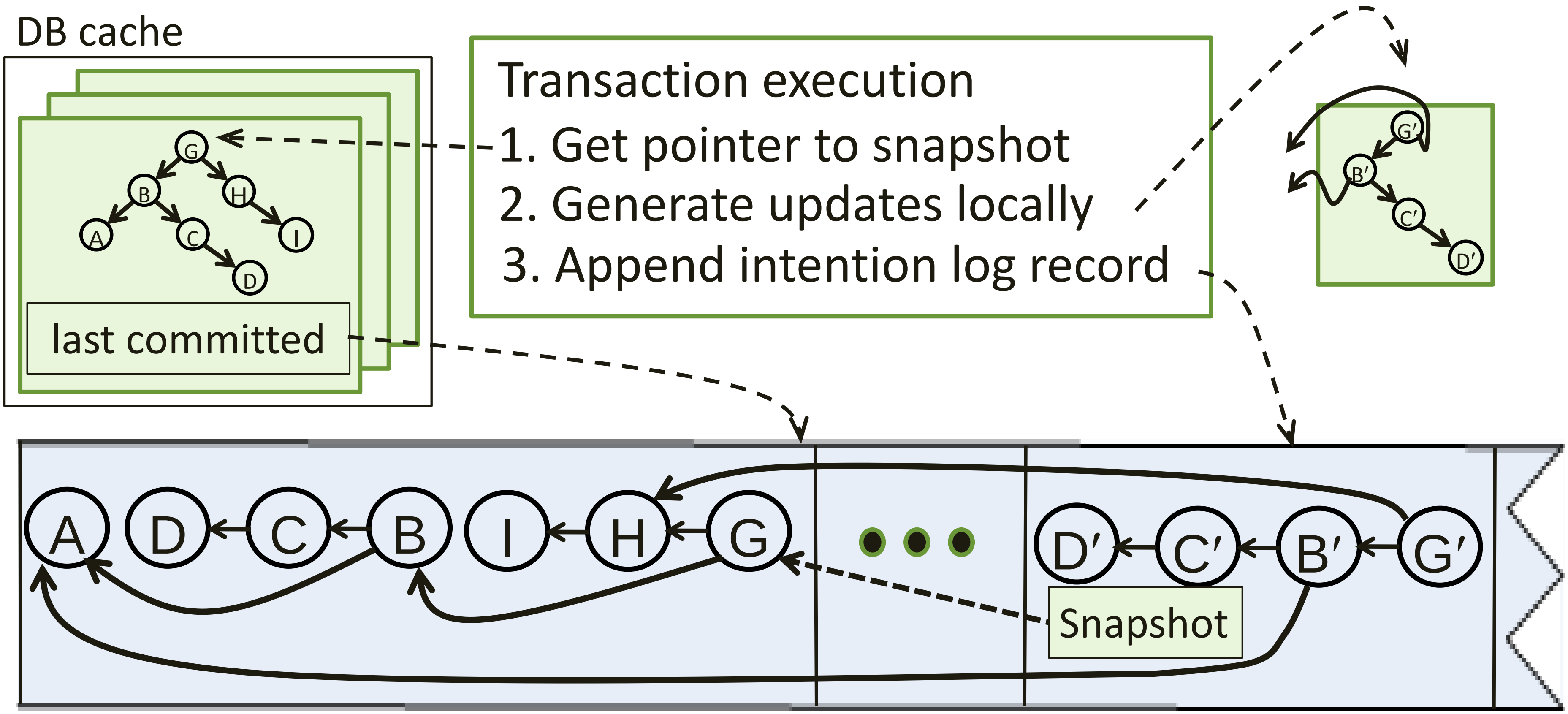
Binary Tree is Multi-versioned

- Copy on write
- To update a node, replace nodes up to the root

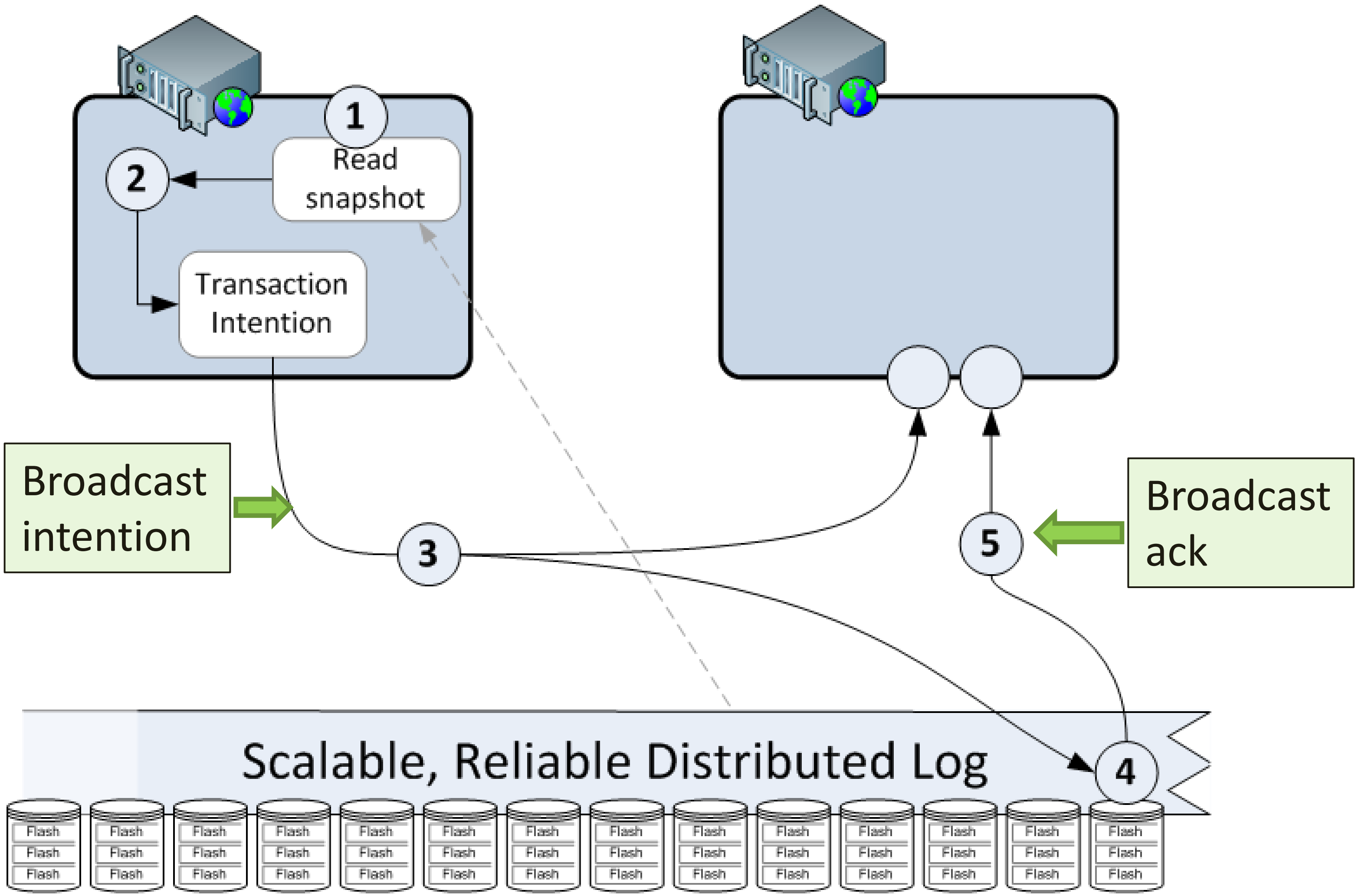


Transaction Execution

- Each server has a cache of the last committed database state
- A transaction reads a snapshot and writes an **intention log record**

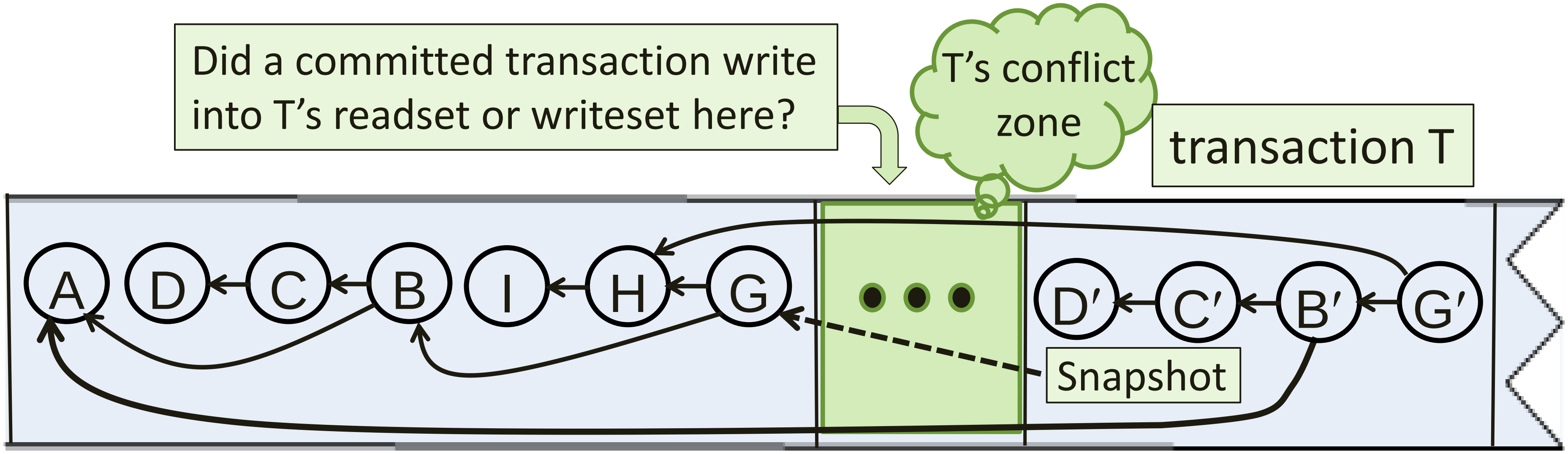


Log Updates are Broadcast



Transaction Commit

- Each server rolls forward transactions in log sequence
- When it processes an intention log record,
  - it checks whether the transaction experienced a conflict
  - if not, the transaction committed and the server merges the intention into its last committed state
- All servers make the same commit/abort decisions



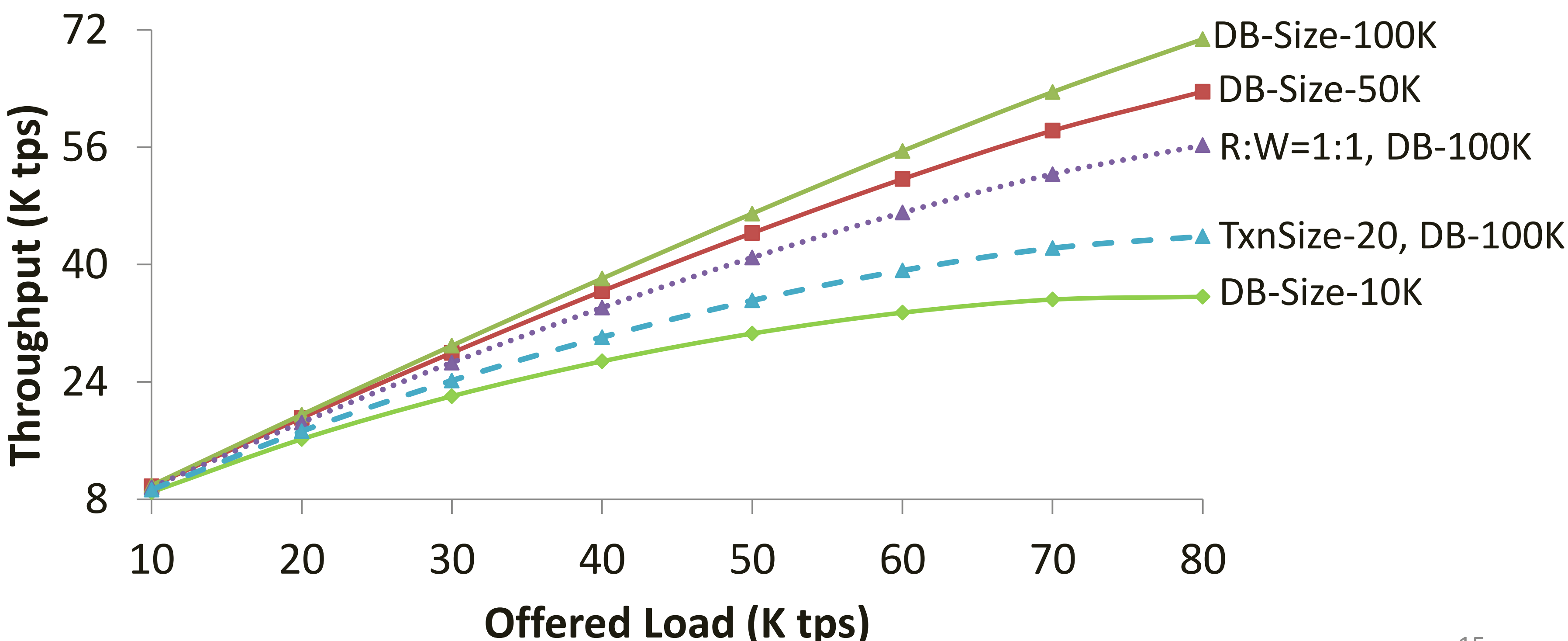
Performance
Bottleneck Analysis

- There are 4 bottlenecks in the update pipeline

1. 100K log-appends/second, assuming 20-way parallel flash storage
2. Broadcast 67K update transactions/second over 10 Gb Ethernet
3. Meld can do up to 400K update transactions/second
4. Opt CC: Abort rate depends on conflict probability and txn latency
   - Suppose transaction latency is 200 µs
   - If all txns conflict, best case SR execution is serial ==> 5000 TPS
   - With random arrivals ==> ~ 1600 update TPS

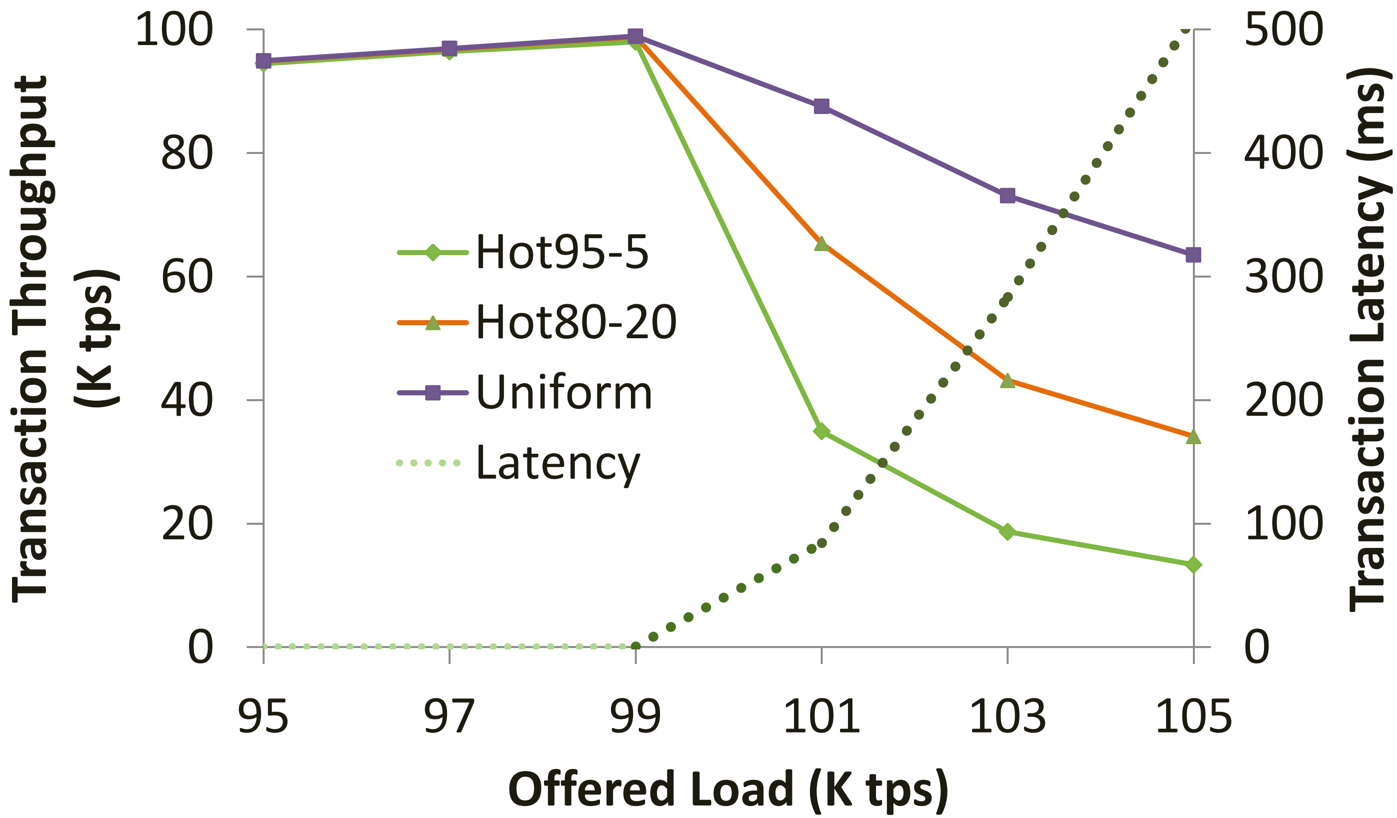
Throughput with High Data Contention

- 8 reads, 2 writes per transaxn
- 99% of ops access 1% of data
- Serializable isolation
- Assume the network, log, and meld can perform 100K intentions/sec



Thrashing due to Resource Contention

- Thrashing occurs when exceeding the maximum resource throughput of 100K/second



Major Technologies

- Flash is append-only. Custom controller has mechanisms for synchronization & fault tolerance
- Storage is striped, with a self-adaptive algorithm for storage allocation and load balancing
- Fault-tolerant protocol for a totally ordered log
- Fast meld algorithm to detect conflicts and merge intention records into last-committed state

Summary of Contributions

- A new data-sharing architecture for scaling out without partitioning.
- A fault-tolerant append-only log that arbitrates concurrent appends by independent servers.
- A log-structured multiversion binary-search-tree index.
- An efficient meld algorithm to detect conflicts & merge committed updates into the last-committed state.
- A simulation analysis of the Hyder architecture under a variety of workloads and system configurations.

Errata for the paper

- In the 5th paragraph of Section 2.3 on sliding window striping, “AppendStripe” should be “AppendPage”.
- Also, the following paper should have been included as related work:
  - Radu Stoica, Manos Athanassoulis, Ryan Johnson, Anastasia Ailamaki: Evaluating and repairing write performance on flash devices. DaMoN 2009: 9-14