HicampDB: In-memory Column Database with Hardware Snapshot Isolation and Deduplication

Bo Wang, David Cheriton
Stanford University

Efficient OLTP and OLAP in the Same Database

• Real-time analytics require serving OLTP and OLAP efficiently in the same database.
• Traditional approach to business data analytics (row-based database + data warehouse) has significant lag for updates becoming visible to analytics.
• Recent emerging databases (e.g. SAP HANA, Vertica) shorten the lag by process both OLTP and OLAP workloads in the same system.
  • Maintain a separate write-optimized delta partition and a read-optimized full partition
  • Changes are cached in the delta partition and merged into the full partition regularly
  • Merging has non-neglectable overhead and requires high extra memory space

HICAMP Memory Architecture

• HICAMP (Hierarchical Immutable Content Addressable Memory Processor)
  • Content-unique lines: 64 bytes by default, indexed by Physical Line ID (PLID)
  • Memory segments: a Directed Acyclic Graph (DAG) of memory lines
  • Virtual segment map: a mapping from Virtual Segment IDs to the root PLID of DAG

Data Deduplication on HICAMP Hardware

• HICAMP deduplication
  • Save the same content only once
  • Compress encoding (e.g. run-length encoding (RLE), delta encoding)
  • Reduce the number of bits used to present the information

  Run Length Encoding
  • RLE tuple: (int value, byte length): 5 bytes

  Delta Encoding
  • Delta sequence: (int base, byte delta1, byte delta2, ...)

Hardware Support for Snapshot Isolation

• Snapshot Isolation
  • All reads made in a transaction see a consistent snapshot of database.
  • Requires no locks and allows for higher performance than serializability.
  • Typically implemented with Multi-Version Concurrency Control (MVCC).
    • Store new versions of data with timestamps
    • Periodically sweep through and delete the old, obsolete data objects

• HICAMP Hardware Support for Snapshot Isolation
  • A write to HICAMP memory creates a new leaf node.
  • At commit, new leaf node will recursively change the PLID in its parent line, which finally generates a new root PLID (i.e. a new root of DAG).
  • Transactions with old root PLID are undisrupted and checked whether to abort at commit.