POWER DB: A Cluster-based Database Replication System
talk adjusted from Sumedha Ahuja

Papers


Client Transactions

- **Read-only (OLAP) Transactions.**
  - Online Analytical Processing
  - Read-only operations
  - Quick Answers to complex database queries
  - Coined by Ted Codd

- **Update Transactions**
  - At least one insert, delete or update statement

Power DB Architecture

- **Cluster of Databases**
  - Commodity computers, each running an off-the-shelf RDBMS
  - All cluster nodes are homogenous

- **Coordinator/Coordination Middleware**
  - On top of the Cluster
  - Queries and Updates submitted to middleware
  - Middleware schedules and routes queries to cluster nodes (OLAP nodes).
  - Updates routed to a special node: OLTP node
  - Lazy Primary-copy replication
Architecture

Detailed System Architecture

*CDBMS: Component DBMS
Query Execution

- Query submitted to PowerDB.
- Parse and validate SQL statement.
- Generate simple execution plan.
- Insert action into scheduling queue.
- Scheduler determines set of candidate components.
- Router selects actual components from the candidates.
- SQL statements evaluated locally on OLAP nodes.
- Coordinator combines local results.
- Final results sent directly to client.

Physical Design

- **Partitioning**
  - Partition (horizontal) relocation across many nodes.
  - Results in intra-query parallelism.
  - Query processing at coordinator may be required.
  - Goal: reduce response time

- **Replication**
  - Entire data replicated over all the nodes.
  - Results in inter-query parallelism.
  - Query processing at coordinator not necessary.
  - Goal: Increase throughput
Current Study

- **Full Replication**
  - Each component contains a copy of all data.
  - Coordinator doesn’t process queries submitted by clients.
  - Query routed to a component by the coordinator and the component returns result to the client directly.

Full Replication of TPC-R
Hybrid Design

- Partition biggest relation over all \( n \) nodes
- Each node holds a copy of all the other relations.
- Query Evaluation:
  - Query referring to the partitioned relation
    - All components process query and coordinator computes overall result.
    - Intra-query execution for queries that run on large data sets
  - Other Queries
    - Query executed on one replica
    - Inter-query parallelism for queries on small data sets
- Update Heuristic:
  - Largest relation receives most updates
  - No replication \( \rightarrow \) no update propagation needed
  - Partitioning leads to equal update load

Hybrid Design of TPC-R

- Part: 20,000
- PartSupp: 80,000
- LinItem: 600,000
- Orders: 150,000
- Supplier: 1,000
- Customer: 15,000
- LinItem: 600,000
- LinItem: 600,000
- LinItem: 600,000
- Region
Routing Schemes

- Balancing-Query-Number
- Affinity-based
- Short Queries ASAP

Balancing-Query-Number

- Same number of active queries at each node
- New query routed to the component with lowest number of active queries

```
function BalanceQNr.Routing ( NodeList nodes, Action new_query ) : Node
begin
  // sort nodes by ascending load; if two nodes have the same number of queries,
  // the stable sort algorithm keeps them in the given order
  stable_sort( nodes, NumberOfQueriesComparison );
  return nodes[0]; // choose node with least load
end
```
Affinity-based

- Assign queries which access the same data to the same component
- Yields an increased buffer-hit ratio => reduces I/O costs.
- Core of algorithm: Affinity Matrix

Affinity Matrix

- N x N matrix; N = Number of queries
- Each cell contains the affinity value of a pair of queries
- Not symmetric
- Affinity value represents improvement (in %) of Mean Response Time (MRT) by concurrent execution of 2 queries.

\[
\text{affinity}(Q_x, Q_y) = 100 - \frac{\text{MRT}(Q_x || Q_y) \times 100}{\text{MRT}(Q_x; Q_y)}
\]

\(\text{MRT}(Q_x || Q_y)\): MRT of \(Q_x\) and \(Q_y\) executed concurrently
\(\text{MRT}(Q_x; Q_y)\): MRT of \(Q_x\) and \(Q_y\) executed sequentially
### Distributed Data Management: Short Queries ASAP

#### Mixture of queries:
- **Complex Analysis Queries**
- **Short Queries**

#### Minimize waiting period of short queries.

- **Complex Query** has to wait until a node is free (not running any query)

- **Short queries** are assigned to a free node or node that has one complex query but no short query running

---

#### Affinity Matrix of TPC-R Queries.

| Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 | Q11 | Q12 | Q13 | Q14 | Q15 | Q16 | Q17 | Q18 | Q19 | Q20 | Q21 | Q22 |
|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|    | 10% |    |    |    |    |    |    |    |     |     |     |     |     |     |     |     |     |     |     |     |     |
| Q3 | 32% | 15% | 6% | 4% | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q4 | 3% | 2% | 38% | 11% | 26% | -  | -  | -  | -  | 9% | -  | -  | 3% | -  | -  | 4% | -  | -  | 21% | -  | -  | -  |
| Q5 | -  | -  | 29% | -  | -  | -  | 9% | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q6 | -  | -  | 7% | 32% | -  | -  | -  | 26% | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q7 | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q8 | 6% | 15% | 28% | 30% | -  | -  | -  | -  | -  | -  | 10% | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q9 | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q10 | 8% | 3% | 10% | 19% | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q11 | -  | -  | -  | -  | 6% | 18% | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q12 | -  | -  | -  | -  | 31% | 37% | 36% | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q13 | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q14 | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q15 | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q16 | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q17 | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q18 | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q19 | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q20 | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q21 | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
| Q22 | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   | -   |
Analysis of Routing Strategies

(FULL REPLICATION)

Analysis of Physical Designs

- More efficient execution plans with hybrid
**Freshness-Aware Scheduling (FAS)**

- **Freshness Index**
  - $f(d) \in [0,1]$: measure of freshness of some data $d$.
  - Reflects how much data has deviated from up-to-date version.
  - $f(d) = 1 \Rightarrow$ Up-to-date data
  - Reflects how stale a certain cluster node is compared to OLTP node.
  - Computed at level of entire database.
  - Based on last propagated update and most recent update on the up-to-date node (OLTP).

\[
    f(C) = \frac{\tau(C)}{\tau(C_0)}
\]

- $\tau(C)$: commit time of the last propagated update transaction on an OLAP node
- $\tau(C_0)$: commit time of most recent update transaction on the OLTP node

**Features**

- **Quality-of-Service parameter.**
  - Trade freshness of data for query performance.
- **Ensures read consistency.**
- **Recall:** Updates propagated using lazy primary-copy replication
  - Updates executed first on OLTP node.
  - Decoupled refresh transactions propagated to replica nodes.
  - Local serializable executions guaranteed at each node.
  - Read transactions prohibited on a node while a refresh transaction executes.
- **Freshness limit additional constraint for query routing.**
Example of FAS

Groundwork (Notations)

- Cluster of Databases, $C = \{c_0, c_1, \ldots, c_n\}$
  - $c_0$ is the OLTP node.
- $f_t$: freshness limit of read-only transaction $t$
- $\mathcal{T}(t)$: Unique timestamp assigned to transaction $t$
- $\mathcal{T}(c_i)$: timestamp of cluster node $c_i$
  - Timestamp of OLTP node = commit time of latest update transaction
  - Timestamp of OLAP node set by each bulk refresh transaction
- $ff_t$: freshness of node accessed by the first query operation of a transaction. Assigned to each transaction.
foreach $a_i$ in queue loop
    // initial set of candidate nodes?
    if $a_i$ is the first of $t_i$ then
    candidates := \{c | c ∈ C, c ≠ c_o : f(c) >= f_{t_i}\};
    // if candidates = ∅ ⇒ activate refresh
    else
    candidates := \{c | c ∈ C, c ≠ c_o : f(c) = f_{t_i}\};
    if candidates = ∅ then Abort($t_i$) end if
end if

// nodes not currently refreshed?
choices := \{c | c ∈ candidates
    \quad \text{ node } c \text{ not refreshing } \};

if ( choices ≠ ∅ ) then
    // router chooses target node
    $c_{target} := \text{Router.} \text{Choose}(choices, a_i);$;
    // remember accessed freshness
    $f'_{t_i} := f( c_{target} )$
    // start subtransaction
    StartSubtransaction($a_i, c_{target});
    RemoveEntry(queue, $a_i$);
else if waiting_time($a_i$) > limit then
    // activate refresh of oldest node
    // and start $a_i$ on refreshed node
end if

// an action remains in the input queue
// until a suitable target is available

end loop

// \[ c_i : c_i \text{ is idle ⇒ activate refresh } \]
Refreshment Strategies

- **ASAP**
  - Propagate changes as soon as possible.
  - Cluster nodes refreshed between queries.
  - Freshness index not used at all.

- **On-Demand**
  - Defers refreshment until demand for a freshness limit that no node can meet.
  - Refreshment starts on node with lowest freshness index.

- **m-idle**
  - Defers refreshment until $m$ cluster nodes have become 'too' old.
  - 1-idle performs best in most cases.

Analysis of Refreshment Strategies

- 10 concurrent update transactions
- 64 concurrent queries
- 32 nodes

(a) Query Throughput

(b) Query Mean Response Time
Influence of Freshness Parameter
(_query performance_)

- \( n \) concurrent update transactions
- \( 2n \) concurrent queries
- 1-idle

Comparison of FAS and Synchronous Updates

- 1 update transaction, \( n \) concurrent queries
Summary of PowerDB

- Design, implementation and evaluation of a Coordination middleware.
- 2 alternatives to data organization:
  - Full replication.
  - Hybrid design.
- Routing schemes:
  - Balance-Query-Number routing.
  - Affinity-Based Query routing.
  - Short Queries ASAP.
- Freshness limit as a Quality-of-Service parameter:
  - Requested freshness limit always met.
  - Data accessed within a transaction is consistent, independent of its freshness.
- Refreshment strategies:
  - ASAP.
  - On-Demand.
  - M-idle.

614: Distributed Data Management