Carnegie Mellon University

Metric Collection Infrastructure

Qidu He, Dongsheng Yang, Wenxuan Qiu
Objective

- **75%**: Implement a minimum set of metrics
  - Database level metrics: number of committed/aborted transactions
- **100%**:
  - Transaction level metrics: latency, number of tuple accesses/modifications
  - Persist to storage
  - Write more tests (current code coverage: 89%)
  - Push data to aggregator when a thread leaves the worker pool
- **112.5%/125%**:
  - Interact with Catalog
  - Interact with Settings Manager
Testing

Six test cases in total:

1. Single thread, single statistic aggregation
2. Single thread, database level statistics aggregation
3. Single thread, database level statistics storage in SQL tables
4. Single thread, transaction level statistics aggregation
5. Single thread, transaction level statistics storage in SQL tables
6. Multiple thread, transaction level statistics storage in SQL tables
Benchmark

Two benchmarks:

Collection speed:
  ● single thread
  ● only collection, no operation
  ● throughput: ~ 1,000,000 event/s

Aggregation speed:
  ● single aggregator
  ● throughout: ~ 500 aggregation/s
Code Quality

Strengths:
- Flexible selection of collectable metrics
- Minimal disruption to execution paths
- Extensible to additional events, metrics, and output streams

Weaknesses:
- Stats aggregator requires a separate thread, which can potentially become a performance bottleneck
- Thread level stats collectors need to be created and destroyed with caution,
Future Work

- Memory usage
  Support memory usage collection, such as memory used per table
- CPU time
  Besides total latency, provide fine-grained timing such as CPU time per transaction
- Flexible storage
  Support more storage methods besides SQL table
- Optimization of tuple-level data
  Approximately compress tuple level metric data using randomized data structures to reduce storage space
Extra Slides
## Implementation

<table>
<thead>
<tr>
<th>Role</th>
<th>Class</th>
<th>Inheritance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Holder</td>
<td>AbstractRawData</td>
<td>DatabaseMetricRawData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TransactionMetricRawData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Latch Wrapper</td>
<td>RawDataWrapper</td>
<td>DatabaseMetric</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TransactionMetric</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Data Handler</td>
<td>AbstractMetric</td>
<td></td>
</tr>
<tr>
<td>Worker</td>
<td>ThreadLevelStatsCollector</td>
<td></td>
</tr>
<tr>
<td>Aggregator</td>
<td>StatsAggregator</td>
<td></td>
</tr>
</tbody>
</table>
Data Flow

Aggregator Thread
- triggers aggregation from other threads
- swap
- persist

Worker Thread
- allocate a metric:
  - register events
  - txn counter type
- callback
- txn commit

txn begin
txn commit
txn abort

txn counter
txn latency

latch

update

Aggregate metrics

register

events

txn counter handler

from other threads

Carnegie Mellon University