Automatically Indexing Millions of Databases in Microsoft Azure SQL Database

Sudipto Das, Miroslav Grbic, Igor Ilic, Isidora Jovandic, Andrija Jovanovic, Vivek R. Narasayya, Miodrag Radulovic, Maja Stikic, Gaoxiang Xu, Surajit Chaudhuri
Motivation

- Indexes can bring orders of magnitude better performance and lower resource consumption
  - A challenging task
  - Human still drives the tuning process despite the help of tools
- Significant burden on users lacking DBA skills
- Doesn’t scale for Software-as-a-Service vendors (SaaS) and Cloud Software Vendors (CSV)
  - SnelStart, AIMS360
Challenges

- **Scale**
  - Millions Databases, Upgrades, Failures, Compliances

- Automatically identify the workload to tune and other tuning constraints

- State-of-the-art index recommenders rely on the query optimizer’s cost estimates

- Minimal interference to the application
  - Low resource footprint
  - Not blocking user operations
Outline

- Auto-indexing Offering
- Architecture
- Deeper-dive
- Experiments
- Statistics and Customer Feedback
- Operational Challenges
Configuration

Configure the automatic tuning options

<table>
<thead>
<tr>
<th>OPTION</th>
<th>DESIRED STATE</th>
<th>CURRENT STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE INDEX</td>
<td>ON</td>
<td>INHERIT</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>DROP INDEX</td>
<td>ON</td>
<td>OFF</td>
</tr>
<tr>
<td></td>
<td>INHERIT</td>
<td></td>
</tr>
</tbody>
</table>

ON
Inherited from server

OFF
Inherited from server
# Index Recommendations

<table>
<thead>
<tr>
<th>ACTION</th>
<th>RECOMMENDATION DESCRIPTION</th>
<th>IMPACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create index</td>
<td>Table: lineitem</td>
<td>⚠ High</td>
</tr>
<tr>
<td></td>
<td>Indexed columns: [L_ShipDate], [L_SuppKey]</td>
<td></td>
</tr>
<tr>
<td>Create index</td>
<td>Table: orders</td>
<td>⚠ Medium</td>
</tr>
<tr>
<td></td>
<td>Indexed columns: [O_OrderKey]</td>
<td></td>
</tr>
<tr>
<td>Create index</td>
<td>Table: lineitem</td>
<td>⚠ Medium</td>
</tr>
<tr>
<td></td>
<td>Indexed columns: [L_ReceiptDate], [L_OrderKey], [L_CommitDate],</td>
<td></td>
</tr>
<tr>
<td>Create index</td>
<td>Table: orders</td>
<td>⚠ Low</td>
</tr>
<tr>
<td></td>
<td>Indexed columns: [O_OrderDate], [O_OrderKey], [O_CustKey], [O_</td>
<td></td>
</tr>
</tbody>
</table>

---

6
**Recommendation Details**

Create index `dbo.orders`

<table>
<thead>
<tr>
<th>Recommended action</th>
<th>Status</th>
<th>Last update</th>
<th>Initiated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create index</td>
<td>Active</td>
<td>10/23/2018</td>
<td>N/A</td>
</tr>
<tr>
<td>Learn more</td>
<td></td>
<td>8:14:47 AM</td>
<td></td>
</tr>
</tbody>
</table>

**Estimated impact**

- **Impact**: Medium
- **Disk space needed**: 10.00 MB

**Details**

- **Index name**: `_dta_index_orders_5_1285579618_K1_5_6`
- **Index type**: NONCLUSTERED
- **Schema**: `dbo`
- **Table**: `orders`
- **Index key columns**: `[O_OrderKey]`
- **Included columns**: `[O_OrderDate], [O_OrderPriority]`
Architecture
Control Plan

- **Per-region centralized service**
  - Speed of engineering, operationalization, and monitoring
  - A centralized store of history of actions

- **Micro-services**
  - Analysis, implement, validate, detect issues/correct

- **Recommendation states:**
  - Active, expired, implementing, validating, success, reverting, reverted, retry, error
Index Recommendation

▪ Workload Coverage
  ➢ Challenging to identify the representative workload (W) even for DBAs
  ➢ Look for high workload coverage (e.g., >80%): ratio of consumed resource

▪ Recommenders
  ➢ Missing Indexes (MI): simpler
  ➢ Database Tuning Advisor (DTA): more complex
Missing Indexes

- Analyze the best indexes relevant to the predicates during query optimization
  - Using simple heuristics

- Predominantly in the leaf node

- Filter with # executions

- Conservative merging, e.g., prefix key columns

- Classifier to further filter out bad indexes
Database Tuning Advisor

- Methods from AutoAdmin
- Resource budget and minimal production impact
  - Reduce samples/optimizer calls, Lower priority lock, automated tracking
- Identify the workload $W$
  - The most expensive $K$ query templates in the past $N$ hours, issues to retrieve from Query Store
- Running DTA as a service
  - Debugging the rec quality is challenging
Drop Indexes

▪ Challenges
  ➢ Occasionally used indexes, e.g., reports
  ➢ Hints/forced plans
  ➢ Which to drop among duplicates

▪ Conservative approach
  ➢ Statistics instead of workload-driven
  ➢ Analyze constraints over long time (e.g., 60 days)
  ➢ Offline analysis to reduce storage overhead
Implementation and Validation

- **Implementation**
  - Resource Governing
  - Scheduling at low activity periods

- **Validation**
  - Logical execution metrics
  - Has plan change due to index change
  - Conservative setting: regression on any major statement triggers a revert
Experiments

- An experimentation framework that adds/removes components and databases easily
- On a few thousands production databases
Statistics

- Around 2 years
- Turned-on by about a quarter of the databases
- Per week: 50K creation and 20K drop
- Tens of thousands of databases reduces >50% CPU consumption
- 11% reverted
  - MI does not account for maintenance cost
  - Optimizer error
Customer Feedback

▪ Earning customer’s trust
  ➢ Business continuity
  ➢ Meaningful performance gains
  ➢ Transparency
  ➢ Robustness

▪ Many seek more control
  ➢ How/when indexes are implemented
  ➢ How to share resource
  ➢ Naming
Operational Lesson

▪ Fill up transaction log
  ➢ Resumable index create

▪ Metadata contention
  ➢ Schema lock when dropping indexes

▪ Not block application process, e.g. schema changes