Managing Long-Running Queries

Stefan Krompass\textsuperscript{TUM}, Harumi Kuno\textsuperscript{HPL}, Janet Wiener\textsuperscript{HPL}, Kevin Wilkinson\textsuperscript{HPL}, Umeshwar Dayal\textsuperscript{HPL}, and Alfons Kemper\textsuperscript{TUM}

\textsuperscript{TUM}Technische Universität München
Munich, Germany

\textsuperscript{HPL}Hewlett-Packard Laboratories
Palo Alto, CA, USA
Motivation

- Business analysis
- Order entry
- Customer relations
- Sales
- Maintenance
- Administrator

DBMS
Motivation

Business analysis

Order entry

Maintenance

Customer relations

Sales

DBMS

Administrator
Goals of this work

• Develop technology to study policies for mixed workloads
• Initial study of managing mixed workloads, in particular: impact of long-running queries on a workload
  – Unreliable cost estimates
    \textit{under-informed admission control and scheduling decisions}
  – Unobserved resource contention
    \textit{monitored resource not the source of contention}
  – System overload
Outline

• Workload management components & workload management policies
• Experiments
• Conclusions
Workload management overview
Experimental approach

• Create workloads that inject “problem” queries (our workloads are derived from actual mixed workload queries)
• Develop a workload management software that implements admission control, scheduling, and execution control policies
• Workload manager feeds queries into database engine simulator
  – Investigate workloads that run for hours
  – Obtain reproducible results
  – Experiment with comprehensive set of workload management policies
  – Inject problem queries
### Experimental input: queries

<table>
<thead>
<tr>
<th>query type</th>
<th>size of query pool</th>
<th>queries per workload</th>
<th>average elapsed time</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>2807</td>
<td>400</td>
<td>30 sec</td>
</tr>
<tr>
<td>medium</td>
<td>247</td>
<td>23</td>
<td>10 min</td>
</tr>
<tr>
<td>long</td>
<td>48</td>
<td>3</td>
<td>1 hr</td>
</tr>
</tbody>
</table>
## Taxonomy of long-running queries

<table>
<thead>
<tr>
<th>Query type</th>
<th>Query expected to be long</th>
<th>Query progress reasonable</th>
<th>Uses equal share of resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>expected-long</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>expected-hog</td>
<td>yes</td>
<td>yes</td>
<td>no (&gt; equal)</td>
</tr>
<tr>
<td>surprise-long</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>surprise-hog</td>
<td>no</td>
<td>yes</td>
<td>no (&gt; equal)</td>
</tr>
<tr>
<td>overload</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>starving</td>
<td>no</td>
<td>no</td>
<td>no (&lt; equal)</td>
</tr>
</tbody>
</table>
Experimental inputs

- **Workload types**: expected-long, surprise-long, surprise-hog

- **Admission control**
  - Policies: none, limit expected costs of a query
  - Thresholds: 0.2m, 0.5m, and 1.0m

- **Scheduling**
  - Queue: FIFO
  - Multiprogramming level (MPL)

- **Execution control**
  - Policies: none, kill, kill&requeue, suspend&resume
  - Thresholds: absolute 5000 (time units), absolute 12000, absolute 5000 & progress < 30%, relative 1.2x
How did we choose the thresholds?
How did we choose the thresholds?

Expected = actual CPU costs
How did we choose the thresholds?

- **Actual CPU cost (in simulator time units)**
- **Estimated CPU cost (in simulator time units)**

- **Short**
- **Medium**
- **Expected long**
- **Surprise-long**

**Surprise-***
How did we choose the thresholds?
How did we choose the thresholds?
How did we choose the thresholds?
Experimental measure - weighted makespan

Elapsed time of queries
Experimental measure - weighted makespan

Q4
Q3
Q2
Q1

Elapsed time of queries
Experimental measure - weighted makespan

Makespan (Q₄ filtered, as expected ➔ no penalty)
Experimental measure - weighted makespan

Makespan (Q₂ and Q₄ filtered)
Experimental measure - weighted makespan

- $Q_4$: 13
- $Q_3$: 4 + 9
- $Q_1$: 5
- Penalty for treating $Q_2$ as false positive: 4.5
Experimental measure - weighted makespan

\[\text{Weighted makespan} = 31\]

\[\text{Weighted makespan} = 4.5 + 9\]
Experimental measure - weighted makespan

1A1C

“one long query admitted” (1A)
“one long query completed” (1C)
Can WM handle unreliable cost estimates?
Admission control thresholds

Workload type

- expected-long
- surprise-long

Weighted makespan (in simulator time units)

- penalty
- 0.2m
- 0.5m
- none
- 1.0m
Can WM handle unreliable cost estimates?

Admission control thresholds

![Bar chart showing weighted makespan in simulator time units for different workload types and admission control thresholds.](chart)

- **Workload type**:
  - expected-long
  - surprise-long

- **Admission control thresholds**:
  - penalty
  - none
  - 0.2m
  - 0.5m
  - 1.0m

- **Weighted makespan** values for different combinations of workload type and admission control thresholds.
Can WM handle unreliable cost estimates?

Adm ctl + exec ctl with different kill thresholds
Can WM handle unreliable cost estimates?

Adm ctl + exec ctl with different kill thresholds

Can WM handle unreliable cost estimates?

Adm ctl + exec ctl with different kill thresholds

Can WM handle unreliable cost estimates?

Adm ctl + exec ctl with different kill thresholds

Can WM handle unreliable cost estimates?

Adm ctl + exec ctl with different kill thresholds

Can WM handle unreliable cost estimates?
Can WM handle unreliable cost estimates?

Adm ctl + exec ctl with different kill thresholds

- penalty
- none
- absolute 5000
- absolute 12000
- progress <30%
- relative 1.2x

Weighted makespan (in simulator time units)

Admission control threshold

0
1.0m

none

3A3C
3A0C
3A2C
3A3C
Can WM handle unreliable cost estimates?

Execution control actions (w/ admission control 1.0m)

- none
- kill
- kill & requeue
- suspend & resume

- penalty
- absolute 5000
- relative 1.2x
Can WM handle unreliable cost estimates?

Execution control actions (w/ admission control 1.0m)

- **penalty**
- **absolute 5000**
- **relative 1.2x**

The diagram shows the weighted makespan (in simulator time units) for different execution control actions:
- **None**
- **Kill**
- **Kill & Requeue**
- **Suspend & Resume**

The x-axis represents the different actions, and the y-axis shows the weighted makespan values.
Can WM handle unreliable cost estimates?

Execution control actions (w/ admission control 1.0m)

Weighted makespan (in simulator time units)

- None
- Kill
- Kill & Requeue
- Suspend & Resume

- Penalty
- Absolute 5000
- Relative 1.2x

Can WM handle unreliable cost estimates?
Can workload management handle system overload?

<table>
<thead>
<tr>
<th>Workload Management</th>
<th>Weighted Makespan (in simulator time units)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>60</td>
</tr>
<tr>
<td>Absolute 5000</td>
<td>80</td>
</tr>
<tr>
<td>Absolute 12000</td>
<td>100</td>
</tr>
<tr>
<td>Relative 1.2x</td>
<td>120</td>
</tr>
</tbody>
</table>

MPL 4

- 2A2C
- 2A0C
- 2A0C
- 2A0C

MPL 10 (overload)

- 2A2C
- 2A0C
- 2A0C
- 2A0C

**Questions**: Can workload management handle system overload?
Can workload management handle system overload?

- Orange: absolute 5000
- Yellow: absolute 5000, progress <30%
- Maroon: relative 1.2x
- Blue: none
- Black: penalty

Weighted makespan (in simulator time units)

MPL 4

MPL 10 (overload)
Conclusion

• Systematic study of workload management policies to mitigate the impact of long-running queries

• Can workload management handle…
  – unreliable cost estimates ✓
  – unobserved resource contention ✓
  – system overload ✗

• Value of this work: experimental framework for studying more challenging workload management problems
Related work (excerpt)

D. G. Benoit. *Automated Diagnosis and Control of DBMS Resources*. EDBT PhD. Workshop, 2000

S. Chaudhuri, R. Kaushik, and R. Ramamurthy. *When Can We Trust Progress Estimators for SQL Queries?* SIGMOD 2005


Workload management tools: HP Neoview, IBM Workload Manager for DB2, Microsoft SQL Server, Oracle Database Resource Manager, Teradata Dynamic Workload Manager