Partition-Based Workload Scheduling in Living Data Warehouse Environments

DOLAP 2007
Lisbon, Portugal

Maik Thiele, Ulrike Fischer and Wolfgang Lehner
Dresden University of Technology, Germany
Database Technology Group
Real-Time Warehousing (I): Push/Pull Principle

- **Pull Principle**
  - Process control by data warehouse
  - Pull the data from the data sources

- **Push Principle**
  - Process control by operative data sources
  - Automatically load data changes into the data warehouse
• **Why Scheduling?**
  - Continuous flow of write-only updates and read-only queries
    \(\rightarrow\) compete for system resources
  - No concurrent read/write transactions through push principle
  - Users expect
    - short response times and
    - high freshness of data
Partition-based Workload Scheduling in Living Data Warehouse Environments

First-level scheduling:

\[ \sum_{u_i} QoS \leq \sum_{QoD} \]

Second-level scheduling:

\[ QoS < QoD \quad QoS = QoD \quad QoS > QoD \]

WINE - Workload Balancing Unit

DWH

Staging Area

queries

updates

q_i

u_i

...
**Workload Model**
- Workload $W$ consists of
  - Read-only queries $q \in W_q$
  - Write-only updates $u \in W_u$

\[
W_q \cup W_u = W
\]
- Each query and update is extended by
  - A position in the query/update queue $pos_{q/u}$
  - A timestamp $t_{q/u}$
- Each query is extended by a vote pair $(qos_q, qod_q)$ where
  - $qos_q / qod_q \in [0,1]$, $qos_q + qod_q = 1$
• **Issue**
  - Find correlations between queries and updates to measure the freshness of a query q (number of unapplied updates)

• **Naïve Approach**
  - Treat whole DWH as one large data item \( \rightarrow \) each query affected by each update

• **Our Approach**
  - Divide DWH into a set of disjoint partitions \( P \)
    \[ \text{DWH} = \{p_i|1 \leq i \leq n\}, \ n \text{ is the number of partitions} \]
  - Each query q touches one or more partitions, \( |P_q| \geq 1 \)
  - Each update u modifies data only within one partition at the same time, \( |P_u| = 1 \)
  - Number of unapplied updates at partition p is \( uu(p) \)
• **QoS Metric**
  - Average retention time, i.e. average time between the arrival \( t_q \) and the execution of each \( q \in W_q \)

\[
QoS(W) = \sum_{q_i \in W_q} \frac{rt_{q_i}}{|W_q|}
\]

• **QoD Metric**
  - Lag-based approach
  - QoD metric for a query \( q \)

\[
QoD(q) = \min_{p_i \in P_q} \left( \frac{1}{1 + uu(p_i)} \right)
\]

/* most out-dated partition */

- QoD metric for the whole workload \( W_q \)

\[
QoD(W) = \frac{1}{|W_q|} \sum_{q_i \in W_q} QoD(q_i)
\]
Optimization Goal

- **Optimization Goal**
  - Minimize retention time (QoS metric) for each query $q \in W_{qos}$,
    
    where $W_{qos} = \{ q \in W_q \mid qos_q \geq 0.5 \}$

  - Maximize freshness (QoD metric) for each query $q \in W_{qod}$,
    
    where $W_{qod} = \{ q \in W_q \mid qod_q > 0.5 \}$

- **Note**
  - QoS and QoD metric are not comparable
**Query Priorization**
- Sort queries in descending order of their QoS values
- Queries with a high QoS value are favored by the system
- Queries with a low QoS value are delayed in execution
  → avoid starvation by increasing the QoD value by

\[
d = \frac{1}{|Q| \cdot T_{\text{max}}}
\]

maximum retention time

at each clock tick

![Query Priorization Diagram]

**Legend**
- **q_i:** Query
- **QoS:** Quality of Service
- **QoD:** Quality of Delay
• **Update Priorization**
  - Prioritize updates whose corresponding queries will be executed soon
  - Weight \( w(u) \) for an update \( u \)
    \[
    w(u) = \sum_{\forall q_i, |P_{q_i} \cap P_u| = 1} \frac{qod_{q_i}}{1 + pos_{q_i}}
    \]
  - Sort update queue in decreasing order of the resulting weights \( w_u \) and in increasing order of the timestamps \( t_u \)
  - Timestamp preserves original update order (consistency)
    → two updates \( u_i \) and \( u_j \)
      - refer to the same partition
      - same weight \( (w(u_i) = w(u_j)) \)
      - \( t_{ui} < t_{uj} \) → \( u_i \) executed first
Second-Level Scheduling: Example

Query queue:

- q₀ : D  
  - 0.2 0.8
- q₃ : B  
  - 0.4 0.6
- q₅ : B  
  - 0.6 0.4
- q₁ : A  
  - 0.7 0.3
- q₄ : C  
  - 0.9 0.1
- q₂ : A  
  - 0.9 0.1

Update queue:

- u₄ : C  
  - 0.05
- u₂ : A  
  - 0.20
- u₃ : B  
  - 0.22
- u₁ : B  
  - 0.22
- u₀ : B  
  - 0.22

\[ \frac{1}{1 + 2} \]
\[ \frac{1}{1 + 2} \]

A
B
C
Experimental Setup

Load
- **Low**: every 2s 10 queries, 10 updates with variance of 5
- **Medium**: every 1.5s 10 queries, 10 updates with variance of 5
- **High**: every 1s 10 queries, 10 updates with variance of 5

Middleware (Java 1.5)
- Wine
- FIFO
- FIFO-QH
- FIFO-UH

Database (IBM DB2 V9.1)
- TPC-DS (web returns, 72,176)
- manufact_id

**System Specifications**
- Intel Pentium D 3.0 GHz
  - 2 GB RAM
  - Windows XP
- Xeon 64 Bit 2.8 GHz
  - 4 GB RAM
  - Linux
WINE and FIFO-QH exhibit no unapplied updates.
FIFO-QH has the most unapplied updates, which increase with rising load.
FIFO has some unapplied updates, number remains stable with increasing load= average case.
Evaluation (II) – Adaptability to User Requirements

LOW workload, alternating user preference

\( <\text{QoS}, \text{QoD}> = <0.1, 0.9> \rightarrow <0.9, 0.1> \rightarrow \ldots \) each 25 sec
Evaluation (III) – Adaptability to User Requirements

LOW workload, alternating user preference

\(<QoS, QoD> = <0.4, 0.6> \rightarrow <0.6, 0.4> \rightarrow \ldots \) each 25 sec
Evaluation (IV) – Adaptability to User Requirements

HIGH workload, alternating user preference

\[ <QoS, QoD> = <0.1, 0.9> \rightarrow <0.9, 0.1> \rightarrow \ldots \text{each 20 sec} \]
Summary and Conclusion

- **Summary**
  - Living DWHs manage continuous flows of updates and queries
  - Extended user model required due to new real-time aspects
  - Conflicting demands
    - short response times $\leftrightarrow$ data freshness
  - WINE: two-level-scheduling algorithm
  - Comparison to three baseline approaches (FIFO-*)
  - WINE outperforms all other approaches under different workloads and changing trends in user requirements

Questions?
Partition-based Workload Scheduling in Living Data Warehouse Environments

DOLAP 2007
Lisbon, Portugal

Maik Thiele, Ulrike Fischer and Wolfgang Lehner
Dresden University of Technology, Germany
Database Technology Group
Evaluation (V) – Adaptability to User Requirements

![Graph showing adaptation of workload scheduling to user requirements over time.](image-url)