Automating Workload Management for Enterprise-Scale Business Intelligence Systems: *Old Problems Just Won’t Go Away*

Umeshwar Dayal
[umeshwar.dayal@hp.com](mailto:umeshwar.dayal@hp.com)

Harumi Kuno, Janet Weiner, Kevin Wilkinson, Abhay Mehta, Chetan Gupta
Intelligent Information Management Lab
Hewlett-Packard Labs
Palo Alto, CA, USA

Stefan Krompass, Alfons Kemper (TU Munich), Archana Ganapathi (UC-Berkeley)
Outline

- Context: Next Generation Business Intelligence
- Workload Management Challenges
- Predicting Query Performance
- Managing Batch Workloads
- Managing Interactive Workloads
- Experimental Framework for Studying Mixed Workload Management
Business Intelligence Today

Technologies, tools, and practices for collecting, integrating, analyzing, and presenting large volumes of information to enable better decision making

- Strategic decision making: analytics by experts
- Off-line, serial pipeline: batch ETL (Extract-Transform-Load)
- Workloads run in isolation on different platforms

- BI is the #1 technology priority for CIO’s 2006-2008
- Market for BI is huge and growing (2006 numbers)
  - Business analytics market: $19B, 10.3% growth
  - Data warehouse platform software market: $5.7B, 12.5% growth
  - BI services market (data integration, DW design, and vertical solutions): $31B
Next-Generation BI: Operational BI

- Bringing BI into the “mainstream”

- A change in concurrency, reliability, workload profile
  - Yesterday: Small number of back office power users
  - Tomorrow: 10,000 users perform business actions... dozens of times a day

- Near real-time: no end-to-end latency
- Mixed workloads
- Robust, predictable performance
- Integrate structured & less-structured data
- Gartner: 90% of Global 2000 will have mission-critical DW applications in place in the next 5 years
Operational BI: Research Challenges

Online Operational Systems

Online Analytic Applications

Operational BI Platform

OLTP
External Feeds
Streams/Events
Operational data store

Extraction & Integration Analytics

Parallel Data Warehouse

Relational Query Processor

Transactional Storage Manager

Delivery and Interaction

Visual interaction and collaboration interfaces

Massively scalable analytics

Self-managing: automated design, tuning, and maintenance

Robust, stable performance
Highly scalable & available
Mixed Workloads
Continuous, on-line updates
Complex batch and ad hoc queries

Highly parallel computer architectures, multi-core processors, storage hierarchies

“Low latency” data integration pipeline
End-to-end optimization
Integration of structured and less structured data
Outline

- Context: Next Generation Business Intelligence
- Workload Management Challenges
- Predicting Query Performance
- Managing Batch Workloads
- Managing Interactive Workloads
- Experimental Framework for Studying Mixed Workload Management
Mixed Workload Performance Management

Managing data warehouse performance is like juggling feathers, golf balls, bowling balls, ...

• Enterprise Data Warehouses are very complex and expensive to manage.
  • Hundreds of ‘knobs’

• Workloads are becoming increasingly complex and dynamic (especially for operational BI)
  • Some queries run in seconds, some take hours
  • Mixed workloads
  • Batch (rollups, reports), interactive (ad hoc queries)
  • Different performance objectives
  • Some are urgent, some have lower priority

• Automation is needed

"An Enterprise Data Warehouse must service a wide spectrum of queries: from simple, sub-second lookups, to complex multi-hour rollups. An EDW is expected to provide the best response time for on-demand queries while simultaneously meeting deadlines for batch queries. All of this must be done without overloading (or under-loading) the system, and with effective controls for rogue queries. This is a difficult problem. Managing the query workload effectively is one of the major challenges in running an EDW."

Through 2010, mixed workload performance will remain the single most important performance issue in data warehousing [Gartner].
Workload Management Problems

• How long should this query take?
  • With other queries running concurrently?
  • What resources will it consume?
• Should we allow this query into the system?
• When should we start the query?
• Will this query ever finish? When?
• What should we do if it is taking too long?
  • Wait?
  • Kill it?
  • Kill it and restart it? When?
  • Kill some other query? Which one?
Workload Management Approach

- Machine learning models for predicting performance of queries and workloads
  - Predict query run times under load
  - Simultaneously predict multiple performance metrics: cpu time, memory usage, message bytes, disk I/O, ...
- Experimental Framework
  - Create a taxonomy of problem queries and operational BI workloads
  - Simulate system configurations, workload execution & workload management policies
  - Systematically study the ability of workload management policies to deal with problem workloads
  - Embed lessons learnt into policy-driven self-management tools
Workload Management Architecture

1. Prediction of Query Runtime
- Workload + Query Optimizer Cost Estimates
- Workload + Dynamic Query Cost Estimates

2. Query Workload Management
- SLA Manager
- SLOs
- Queries
- Admission Control
- Queue 1
- Queue 2
- Queue k
- Scheduler
- Execution Manager
- Execution Engine
- Execution Cost Control

3. Query Progress Indicator
- Statement Name: Query_101
- Query Progress Indicator
- Elapsed Time: 53.22
- Current Time (t*): 25th August 2006 16:24:40
- Estimated Cost: 352 dop
- Query Execution Speed (at t*): 1.2 dop/min
- Estimated Query Time Left: 4:09:20

4. Early Warning System
- System Load
- Load Monitor
Outline

- Context: Next Generation Business Intelligence
- Workload Management Challenges
- Predicting Query Performance
- Managing Batch Workloads
- Managing Interactive Workloads
- Experimental Framework for Studying Mixed Workload Management
Currently, no good way to estimate execution time*

Cost as a predictor of Execution time (559 HPIT Canned reports and Yotta generated queries)

\[ y = 0.2206x + 13.067 \]

\[ R^2 = 0.0766 \]

*Even on a ‘quiet’ system with no load and no query concurrency.
Prediction of Query Run Time (PQR)

- Predict the execution time of a query, allowing the user to trade off accuracy and precision.
- Learn from execution histories
- Understand the problem as that of classification with unknown classes.
- At the heart of this approach is a machine learning model called PQR Tree (Prediction Of Query Runtime).

Approach:
- From historic data extract features that describe the query plan and the system load.
- Train a PQR Tree on these features.
- For a new query extract its plan and load features.
- Apply the PQR Tree on these features to predict the time range of the new query.

**PQ R Tree**

- A **PQ R Tree**, denoted by $T_s$, is a binary tree such that:
  1. For every node $u$ of $T_s$, there is an associated **2-class classifier** $f_u$.
  2. The node $u$ contains examples $E_u$, on which the classifier $f_u$ is trained.
  3. $f_u$ is a classifier that decides for each new query $q$ with execution time $t_q$ in $[t_{ua}, t_{ub}]$, if $q$ should go to $[t_{ua}, t_{ua}^\star]$ or $[t_{ua}^\star, t_{ub}]$, where $\star$ lies in $(0, t_{ub} - t_{ua})$.
  4. For every node $u$ of $T_s$, there is an associated **accuracy**, where accuracy is measured as the percentage of correct predictions made by $f_u$ on the example set $E_u$.
  5. Every node and leaf of the tree corresponds to a time range.

![Diagram of PQ R Tree](image-url)
Feature Vectors

• Query Plan Vector
  • Select features through enumeration, construction and domain expertise.
  • Bushiness, Disk Partition Parallelism, Process Parallelism, Number of IOs, IO Cardinality, IO Cost, Non-IO Cost, Number of Joins, Join Cardinality, Number of Probes, Table Count, Number of Sorts, Sort Cardinality, Total Cost, Total Estimated Cardinality, etc.

• Load Vector
  • The CPU characteristics tend to be erratic and unpredictable.
  • “Stretch” of a query is a good indicator of the load the query experiences while running.
  • MPL is a determining factor for stretch and a good variable for prediction. Hence the new load vectors:
    • Query Stretched Cost, Query Stretched Process Parallelism, Query Stretched Operator Cost, Process stretched cost, Process stretched Process parallelism, Process stretched Operator Cost

```
SELECT n.name
FROM tbNation n, tbRegion r
WHERE n.region = r.region
AND r.name = 'EUROPE'
ORDER BY n.name;
```
Creating the PQR Tree

- Recursively create nodes till one of the termination conditions is met:
  - Time range is too small.
  - Number of training examples is too small in a node.
  - Accuracy of prediction falls below a threshold.
- To create a node:
  - Take all the queries in the training set and arrange them in ascending order of running time. Each running time is a potential point at which the time range for the node can be split.
  - For every classifier in a fixed set of classifiers compute accuracy for each potential split point.
  - Choose the combination of classifier and the split that has the highest accuracy on the test set.
- Enhancements:
  - Top K: Compute the percentage change between successive values in this list. Choose some fixed number of largest changes. They are potential split points.
  - Skip Split Points: Skip \( n_{\text{skip}} \) percent points from both ends of an interval as candidate split points.
Applying PQR Tree

- **Predict** the time interval for the execution time of a new query.
  - Decompose the query and the system load into the feature vector described earlier.
  - Apply the classifier at each node to this feature vector, to determine whether this new query belongs to the left or the right child.
  - Do this recursively at each node until we reach a leaf of the PQR tree.
- The leaf has a time range associated with it and this becomes the predicted execution time of the new query.
- As the range of execution times becomes narrower corresponding to a greater depth in the tree, the prediction accuracy will be lower. This approach allows the user to choose a time range they are comfortable with.
Resulting Prediction of Query Runtime (PQR) Model – Loaded system

Test set: 170 actual customer queries run concurrently on a loaded system
## PQ R Results - TPC-H

<table>
<thead>
<tr>
<th>Test</th>
<th>MPL = 10</th>
<th>MPL = 8</th>
<th>MPL = 6</th>
<th>MPL = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Buckets</td>
<td>Accuracy</td>
<td>Buckets</td>
</tr>
<tr>
<td>1</td>
<td>85.58</td>
<td>6</td>
<td>89.34</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>9</td>
<td>73.83</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>84</td>
<td>5</td>
<td>84.21</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>85.19</td>
<td>9</td>
<td>87.5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>75.49</td>
<td>11</td>
<td>88.04</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>88.78</td>
<td>7</td>
<td>84.09</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>81.63</td>
<td>6</td>
<td>92.66</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>75.25</td>
<td>6</td>
<td>67.56</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>85.45</td>
<td>8</td>
<td>81.13</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>75.23</td>
<td>8</td>
<td>87.63</td>
<td>7</td>
</tr>
<tr>
<td>Avg</td>
<td>81.66</td>
<td>7.5</td>
<td>83.60</td>
<td>7.5</td>
</tr>
</tbody>
</table>
Results on a real customer workload

![Graphs showing accuracy vs time ranges for two different workloads.](image-url)
Beyond Run-time Prediction: Predicting Multiple Performance Metrics

• Goal:
  • For a specific machine configuration ...
  • Predict performance of a query run at MPL=1
    • Using only the query and compiler output
    • Predict execution time, cpu time, message bytes, disk i/o,...
  • Predict performance of a workload run at given MPL > 1

• Applications:
  • Initial sizing
    • What system configuration for a given workload?
  • Capacity planning
    • What happens to performance when workload or configuration changes?
  • Workload management
    • What to expect from queries, workload?
    • What is ideal MPL?
SELECT n.name
FROM tbNation n, tbRegion r
WHERE n.region = r.region
    AND r.name = 'EUROPE'
ORDER BY n.name;

Create Query Vector from Compile-Time features for input to prediction

<table>
<thead>
<tr>
<th>Operator</th>
<th>Number of Instances</th>
<th>Sum of Cardinalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>exchange</td>
<td>1</td>
<td>5.00</td>
</tr>
<tr>
<td>file_scan</td>
<td>2</td>
<td>7.02</td>
</tr>
<tr>
<td>nested_join</td>
<td>1</td>
<td>5.00</td>
</tr>
<tr>
<td>partitioning</td>
<td>2</td>
<td>4.52</td>
</tr>
<tr>
<td>root</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>sort</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>split</td>
<td>2</td>
<td>4.52</td>
</tr>
</tbody>
</table>
Training (through Kernel Canonical Correlation Analysis - KCCA)

Compile-time features

Query Plan

Query Plan Feature Matrix

<table>
<thead>
<tr>
<th>Op1</th>
<th>Op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4E4</td>
<td>80E2</td>
</tr>
<tr>
<td>23E5</td>
<td>2E4</td>
</tr>
</tbody>
</table>

Statistics

Elapsed Time
Execution Time
I/Os

Run-time features

Performance

Performance Feature Matrix

<table>
<thead>
<tr>
<th>Time</th>
<th>I/O</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3E7</td>
<td>27E2</td>
<td></td>
</tr>
<tr>
<td>23839</td>
<td>7.3E3</td>
<td></td>
</tr>
</tbody>
</table>

Query Plan Projection

Matrices showing similarity of query pairs using a Gaussian kernel function

L X A

Co-locate queries based on plan AND performance features

L = \begin{bmatrix} 0 & LP \\ PL & 0 \end{bmatrix}

A = \begin{bmatrix} L & 0 \\ 0 & P \end{bmatrix}

P X B
Prediction (through KCCA)
Results: Predicted vs. Actual Elapsed Time

- KCCA predicted elapsed time vs. actual elapsed time
- Under-estimated records accessed
- Disk I/O estimate too high
- Perfect prediction
Optimizer Cost Estimates

Optimizer Cost Estimates

1 hour

1 minute

1 second

Actual Elapsed Time

Cost estimate 100x away from best fit

Cost estimate 10x away from best fit

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 10x away from best fit

Line of best fit

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit

Optimizer Cost Estimates

Cost estimate 100x away from best fit
**Predicting Other Metrics**

- **Under-estimated records accessed**
- **Perfect prediction**

[Graph showing KCCA predicted records used vs. actual records used with a linear relationship and a perfect prediction line]

[Graph showing Actual Message Count vs. KCCA Predicted Message Count with a perfect prediction line]
Performance Prediction Summary

- Predict performance for queries AND workloads
  - Train and predict for a given configuration (hardware / software / MPL)
  - Predict using only query plan features of new queries.
- Test results show good prediction accuracy
  - Predict multiple aspects of performance: cpu time, memory usage, message bytes, disk I/O, ...
  - Multiple metrics useful: explain inaccuracies; Suggest which queries to run together (complementary use of memory, disk)
- Training is time-consuming
  - Many hours to run queries and workloads; Few hours to train models
- But prediction is fast: only 10-15 seconds per query
- Continuous retraining will be needed

Outline

• Context: Next Generation Business Intelligence
• Workload Management Challenges
• Predicting Query Performance
• Managing Batch Workloads
• Managing Interactive Workloads
• Experimental Framework for Studying Mixed Workload Management
Batch Workload Management

- Maximize the throughput of BI batch workloads (while protecting against underload and overload)
- Intuition:
  - Find a good ‘manipulated variable’.
  - Make the system stable over a wide range of this ‘manipulated variable’.
  - Target somewhere in the middle of the range
- MPL has been used but:
  - The optimal mpl is a lower number for resource-intensive queries, whereas the optimal mpl is higher for less intensive queries.
  - A typical BI workload can fluctuate rapidly between long, resource intensive queries and short, less intensive queries.

**Throughput**

Underload

Optimal range

Target

Overload (thrashing)

MPL

Large Workload

Medium Workload

Target

MPL = Number of concurrent streams

*Labs*
Peak Memory Consumption – A Better Manipulated Variable

PG M (Priority Gradient Multiprogramming)

Equal Priority Multiprogramming

- 'm' concurrent streams (M PL=m)
- Each running at an equal priority (eg 148)

Priority Gradient Multiprogramming

- 'm' concurrent streams (M PL = m)
- Each running at a DIFFERENT priority (eg 148, 146, 144, 142, ..., 148-2(m-1))

EPM: Default Execution Control
PG M: Proposed Execution Control
PGM Protects Against Overload.

- PGM is stable and exhibits saw-tooth pattern of acquisition, consumption and release.
- EPM shows the concurrent buildup of memory resulting in overflow.

*SF200, MPL=15, Query 4 data from MEASURE
Hence, stabilizes execution across memory prediction errors

Reasons for stability:
- PG M is unfair
- Quick release
Outline

• Context: Next Generation Business Intelligence
• Workload Management Challenges
• Predicting Query Performance
• Managing Batch Workloads
• Managing Interactive Workloads
• Experimental Framework for Studying Mixed Workload Management
Interactive Workloads: Scheduling

Design a strategy for the online scheduling of queries on an Enterprise Data Warehouse.

OLTP

- a)
  - Query Size: 51 seconds
  - ART: 74.50
  - AS: 1.48

- b)
  - Query Size: 49 seconds
  - ART: 75.50
  - AS: 1.52

BI

- c)
  - Query Size: 99 seconds
  - ART: 50.50
  - AS: 1.00

- d)
  - Query Size: 1 second
  - ART: 99.50
  - AS: 50.50

Stretch = (p_i + w_i)/ p_i
Criteria for a ‘good’ Online Scheduler

FEED

• **Fair**
  - Avoid starvation of queries
  - Lower the ‘max’, or the ‘variance’ of query times

• **Effective**
  - The majority of queries should do well
  - Lower the ‘average’

• **Efficient**
  - The strategy should be lightweight
  - Computational complexity of implementation should be sub-linear

• **Differentiated**
  - Different service levels should give differentiated performance without unduly penalizing the overall avg and max.
  - The ‘CEO’ query
How our scheduler works

- Maintain a queue of queries ordered by their ‘rank’.
- When a new query is submitted, compute its ‘rank’, and insert it into the queue in the order of its ‘rank’.
- Run the queries in the order of their ‘rank’, with ‘Highest Rank First’

 rank? DBMS
Ordered by decreasing ‘rank’

How is rank computed?...

Abhay Mehta, Chetan Gupta, Song Wang and Umesh Dayal, rFEED: A Mixed Workload Scheduler for Enterprise Data Warehouses. ICDE 2009
Design the ‘rank’ function using the FEED criteria

1. Start with a proven ‘effective’ algorithm (SJF):
   \[ r = \frac{1}{p}, \] where ‘p’ is the estimated cost of the query

2. Add a wait component that achieves ‘fairness’:
   \[ r = \frac{1}{p} + K w, \] where K is a constant, and w is the wait time of the query

3. Include a component that allows differentiation between service levels:
   \[ r = \cdot \left( \frac{1}{p} + K w \right), \] where \( \cdot \) indicates the service level

4. Implementing this function as a queue has a sub-linear computational complexity
   - Computational complexity is \( O(\log n) \)
Computing the Parameters

- Based on the value K, rank function \( r = \frac{\cdot}{p} + Kw \) generalizes known scheduling functions:
  - SJF: \( K=0 \)
  - FIFO: \( K > 1 - \frac{1}{\cdot} \) where \( \cdot \) is the execution time of the longest query
- Tradeoff between fairness and effectiveness
  - \( 0 < K < 1 - \frac{1}{\cdot} \)
  - Derive a value for K

\[ K = \frac{1}{\psi^2} \]

- With this value, provably does not cause starvation
- Mapping Service Levels: The ratio \( r \) of stretches of a query of size \( p \) with service level \( \cdot \) and 1 can be computed as:

\[ r = \frac{T_{1,p} + T_{\Delta,p} + p}{T_{\Delta,p} + T_{\Delta,p/\Delta} + p} \]

- For Pareto Distribution:

\[ \Delta = \frac{(r-1)(N_1 \ln p + N_\Delta \ln p + p)}{N_1 + N_\Delta} \]
Experimental Setup

- Model workloads
  - Pareto Distributions
  - Optimizer Cost Estimate
  - Actual Processing Time
- Study $L_2$ norm for Stretch $L_2 = (p_1^2 + p_2^2 + \ldots + p_n^2)^{1/2}$
- Study differentiation: Two models for rFEED: $rFEED_1$ & $rFEED_{10}$
- Compare with an optimal resource slicing system $RS_{OPT}$
- Study Peak Load and Steady State Load
Execution Time With Pareto Distribution (Steady State)
Execution Time with Optimizer Estimate (Peak Load)
Outline

- Context: Next Generation Business Intelligence
- Workload Management Challenges
- Predicting Query Performance
- Managing Batch Workloads
- Managing Interactive Workloads
- Experimental Framework for Studying Mixed Workload Management
Experimental Framework for Workload Management

- It’s non-trivial to gauge the impact of system configuration and workload on a BI system’s performance...

Gives you knobs to control:
- Workload characteristics: problem queries, variance
- Workload Management policies: admission control, scheduling, execution controls
- Multi-programming levels
- Resource models and behavior

So you can see:
- Impact of admission control, scheduling, and execution control policies
- Impact of problem queries
- Impact on different objectives (e.g., complete 100% workload, minimize makespan, maximize useful work
Taxonomy of long-running queries

<table>
<thead>
<tr>
<th></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-Heavy</td>
<td>Yes</td>
<td>Yes</td>
<td>Equal share</td>
</tr>
<tr>
<td>Expected-Hog</td>
<td>Yes</td>
<td>Yes</td>
<td>&gt;Equal share</td>
</tr>
<tr>
<td>Surprise-Heavy</td>
<td>No</td>
<td>Yes</td>
<td>Equal share</td>
</tr>
<tr>
<td>Surprise-Hog</td>
<td>No</td>
<td>Yes</td>
<td>&gt; Equal share</td>
</tr>
<tr>
<td>Overload</td>
<td>No</td>
<td>No</td>
<td>Equal share</td>
</tr>
<tr>
<td>Starving</td>
<td>No</td>
<td>No</td>
<td>&lt; Equal share</td>
</tr>
</tbody>
</table>
## Workload Management Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warn</td>
<td>Print a message to a log</td>
</tr>
<tr>
<td>Hold</td>
<td>Notify DBA; run query only if DBA releases it</td>
</tr>
<tr>
<td>Delay</td>
<td>Delay execution of query; put query in queue</td>
</tr>
<tr>
<td>Reject</td>
<td>Do not run query; return error</td>
</tr>
</tbody>
</table>

### Admission Control Policies

<table>
<thead>
<tr>
<th>Queues</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>All queries start as soon as admitted (default for most database systems)</td>
</tr>
<tr>
<td>One</td>
<td>One FIFO queue for all queries waiting to run</td>
</tr>
<tr>
<td>Priority</td>
<td>Separate queues for different query priorities</td>
</tr>
<tr>
<td>Size</td>
<td>Separate queues for different expected runtimes</td>
</tr>
</tbody>
</table>

### Execution Control Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Let the query run to completion</td>
</tr>
<tr>
<td>Warn</td>
<td>Print a message to a log; query continues</td>
</tr>
<tr>
<td>Reprioritize</td>
<td>Change the priority of the query</td>
</tr>
<tr>
<td>Stop</td>
<td>Stop processing query; return results so far</td>
</tr>
<tr>
<td>Kill</td>
<td>Abort the query and return an error</td>
</tr>
<tr>
<td>Kill &amp; Requeue*</td>
<td>Abort the query; put it in a scheduling queue to start over</td>
</tr>
<tr>
<td>Suspend &amp; Resume*</td>
<td>Stop processing query; put saved state in scheduling queue</td>
</tr>
</tbody>
</table>
### Workload Management Policies Implemented by Commercial Products

<table>
<thead>
<tr>
<th>Admission policy</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limit queries running concurrently</td>
<td>Reject; Hold</td>
</tr>
<tr>
<td>Limit queries in queue</td>
<td>Reject; Hold</td>
</tr>
<tr>
<td>Limit logon sessions</td>
<td>Reject</td>
</tr>
<tr>
<td>Limit expected costs</td>
<td>Hold</td>
</tr>
<tr>
<td>Limit resource usage</td>
<td>Reject; Hold</td>
</tr>
<tr>
<td>Check access permissions</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scheduling</th>
<th>Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue types</td>
<td>One; Priority</td>
</tr>
<tr>
<td>Query starts when under threshold</td>
<td>MPL; Usage</td>
</tr>
<tr>
<td></td>
<td>MPL; Costs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Execution condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed time &gt; threshold</td>
<td>Kill; <em>Reprioritize</em></td>
</tr>
<tr>
<td>Actual cardinality &gt; threshold</td>
<td>Kill, Warn, <em>Reprioritize</em></td>
</tr>
<tr>
<td>Actual / estimated cardinality &gt; threshold</td>
<td>Warn</td>
</tr>
<tr>
<td>Actual CPU / estimated CPU &gt; threshold</td>
<td>Kill; Warn, <em>Reprioritize</em></td>
</tr>
</tbody>
</table>

*Note: *Reprioritize action depends on specific implementation details.
Execution Controller

IF relExecTime IS high AND (Progress IS low OR Progress IS medium) THEN cancel IS applicable

IF relExecTime IS high AND (Progress IS high Then reprioritize IS applicable
Preliminary Results from Experimental Framework

• Huge variance in query sizes and resource demands
  • Short queries: wait time can outweigh execution time
  • Long queries: even a few can significantly slow system
• Admission control is effective when execution cost estimates are accurate
• When execution costs are underestimated, threshold-based execution control actions have to be taken
  • But need guidelines for setting thresholds
• Corrective actions against long-running queries can help, but
  • Overly aggressive actions can hurt performance
  • Simple actions (kill, resubmit) about as effective as newly proposed actions (suspend/resume)

Example of Experimental Results

Interactive jobs

Batch jobs

Impact of problem queries

Impact of control actions
Thank You!