Developing an Architecture to Search for When Different Parallelization Operations are effective: An attempt to apply machine learning to database parallelization

A Ph.D. Dissertation Proposal

By
Jozsef Patvarczki

Dissertation Committee:
Prof. Neil T. Heffernan  Prof. Elke A. Rundensteiner
Prof. Craig E. Wills  Dr. Daniel C. Zilio (IBM)
Outline

- Motivation and Expected contributions
- Related work
- Main part of my thesis:
  - Definitions
  - Fundamental assumptions
  - Accomplished work
  - Research Questions
  - Proposed tasks
- Proposed Evaluation
- Project Milestones
- Questions
Motivation

- There are thousands of web applications (Amazon, eBay, etc.), and these systems need to figure out how to scale-up their performance.
- Scalability challenges: Web server, Application server, backend.
- Common problem when running web-based applications is how to scale-up the database.
- Distributing load across multiple application servers is fairly straightforward.
- Distributing load (read, update, delete, and insert queries) across multiple database servers is more complex.
- Automatic physical database design tools mostly rely on “what-if” that estimates the execution time of the queries.
- Developers of Web-based technologies want to know how to adapt their systems to distribute the load for the best results.
- ASSISTment System (www.assistment.org) Web-based ITS.
Automatic database design tool (IBM DB2)
Motivation

- Why should computer scientists care?
  - Surijat Chaudhuri (Microsoft Research), “Query Optimizers: Time to Rethink the Contract?”, SIGMOD'09, Rhode Island, USA
    - “We suggest revisiting the contract we have with the query optimizer – they should be able to leverage significant additional information from the application developer and usage based analysis such as search directives”
  - Researchers in database technologies are interested in methods for exploiting system characteristics
  - Database Administrators want to know how to adapt their systems to scale-up and achieve the best results
  - Researchers of Cloud computing want to understand how users interact with a system and how to generate database intensive input information
Expected contributions

1) A data placement algorithm that can converge quickly over time as it trains over a set of examples and machine learned rules;

2) Parameterized and machine learned rules to help govern the physical design of the database across an arbitrary number of computer nodes;

3) A shared-nothing data replication middleware for Web-based applications that can be easily built using low-cost existing resources to realize database scaling possibilities without expensive storage area networks.
Related work

- Issues of physical database layout design have a very long history in the database field
- Issues related to the distribution of requests using multiple database servers to decrease server loads have stayed open
- A characteristic of web based applications
  - all the incoming query templates are known beforehand: 
    <INPUT NAME="items" TYPE="text" SIZE="12"> transformed into 
    SELECT * FROM items WHERE item_type = 'jacket'
  - users typically interact with the system through a web interface such as web forms
- Knowing each query template in advance allows us to propose better solutions for balancing load
Related work

- Choosing a good physical database design, using different operators, is an essential task for almost any automatic physical database design tools
  - Operators: denormalization, horizontal partitioning, vertical partitioning, full replication, etc.

- Robust physical database design can have a great impact on the system performance

- Physical database design faces a huge design search space
  - For example, the horizontal and vertical partitioning problems over a set of processors have been showed to be NP hard
  - Different operators could potentially interact with each others

- Most of the automatic physical design tools consider the workload-based approach
  - A query workload is any group of queries that run on the database
  - SQL query workload resembles the future workload in the production environment
Related work

-(Hammer and Niamir, 1979) A Heuristic Approach to Attribute Partitioning
-(Agrawal, Chaudhuri, and Narasayya, 2000) Automated Selection of Materialized Views and Indexes for SQL Databases
-(Rao, Zhang, Megiddo, and Lohman, 2002) Automating physical database design in a parallel Database
-(Plattner, Alonso, and Ozsu, 2006) DBFarm: A Scalable Cluster for Multiple Databases
-(Agrawalm, Chu, and Narasayya, 2006) Automatic Physical Design Tuning: Workload as a Sequence
-(Groothuyse, Sivasubramanian, and Pierre, 2007) GlobeTP: Template-Based Database Replication for Scalable Web Applications
-(Hueske, and Markl, 2007) Detecting Attribute Dependencies from Query Feedback
-(Wei, Dejun, and Pierre, 2008) Service-Oriented Data Denormalization for Scalable Web Applications
Related work

**Industrial work**

-(Lightstone, Lohman, and Zilio, 2002) Toward Autonomic Computing with DB2 Universal Database
-(Zilio, Rao, Lichtstone, et. al., 2004) DB2 Design Advisor: Integrated Automatic Physical Database Design
-(Dageville, Das, Dias, et. al., 2004) Automatic SQL Tuning in Oracle 10g.
-(Agrawal, Chaudhuri, Narasayya, et. al., 2004) Database Tuning Advisor for Microsoft SQL Server 2005

- PostgreSQL (PGPOOL-II), [http://pgpool.projects.postgresql.org](http://pgpool.projects.postgresql.org)
**Operator definitions**

- **Denormalization (DN)**
  - One moves from higher to lower normal forms in the database modeling and adds redundant data;
  - Performance improvement is achieved because some joins are already pre-computed.

<table>
<thead>
<tr>
<th>Table: users</th>
<th>id</th>
<th>login</th>
<th>email</th>
<th>pass</th>
<th>salt</th>
<th>date</th>
<th>timezone</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Table: user_details</th>
<th>id</th>
<th>first name</th>
<th>last name</th>
<th>birthdate</th>
<th><strong>userid</strong></th>
<th>display name</th>
</tr>
</thead>
</table>

Denormalized new table

<table>
<thead>
<tr>
<th>id</th>
<th>login</th>
<th>email</th>
<th>pass</th>
<th>salt</th>
<th>date</th>
<th>timezone</th>
<th>first name</th>
<th>last name</th>
<th>birthdate</th>
<th>display name</th>
</tr>
</thead>
</table>
Operator definitions

- **Partitioning**
  - **Horizontal (HP)**
    It distributes the rows of a table into several separated tables based on common column values;
    New tables contain the same number of columns, but fewer rows.
  - **Vertical (VP)**
    It distributes the columns of a table into several separated tables;
    New tables contain the same number of rows, but fewer columns.

Table: users

<table>
<thead>
<tr>
<th>id</th>
<th>login</th>
<th>email</th>
<th>pass</th>
<th>salt</th>
<th>date</th>
<th>timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>david</td>
<td><a href="mailto:dd@wpi.edu">dd@wpi.edu</a></td>
<td>K12sj</td>
<td>12sa</td>
<td>2010</td>
<td>EST</td>
</tr>
<tr>
<td>13</td>
<td>dann</td>
<td><a href="mailto:da@wpi.edu">da@wpi.edu</a></td>
<td>Da1f3</td>
<td>942s</td>
<td>2010</td>
<td>EST</td>
</tr>
<tr>
<td>14</td>
<td>Peter</td>
<td><a href="mailto:pt@wpi.edu">pt@wpi.edu</a></td>
<td>F3jd3</td>
<td>14k3</td>
<td>2010</td>
<td>EST</td>
</tr>
</tbody>
</table>
Operator definitions

- Data Replication (R)
  - Full Replication (FR)
    Storing the same data in multiple locations;
    It replicates the entire table on a sub-set of database servers (nodes).

  - Partial Replication (PR)
    It assumes that shared data is partitioned into n disjoint databases and we allow replication of an arbitrary subset of the databases.

<table>
<thead>
<tr>
<th>id</th>
<th>login</th>
<th>email</th>
<th>pass</th>
<th>salt</th>
<th>date</th>
<th>timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>david</td>
<td><a href="mailto:dd@wpi.edu">dd@wpi.edu</a></td>
<td>K12sj</td>
<td>12sa</td>
<td>2010</td>
<td>EST</td>
</tr>
<tr>
<td>13</td>
<td>dann</td>
<td><a href="mailto:da@wpi.edu">da@wpi.edu</a></td>
<td>Da1f3</td>
<td>942s</td>
<td>2010</td>
<td>EST</td>
</tr>
<tr>
<td>14</td>
<td>Peter</td>
<td><a href="mailto:pt@wpi.edu">pt@wpi.edu</a></td>
<td>F3jd3</td>
<td>14k3</td>
<td>2010</td>
<td>EST</td>
</tr>
</tbody>
</table>
This thesis makes three fundamental assumptions:

1) We know ahead of time every query template that could come to the system;
2) We demand that each query will be answerable by a single database server;
3) Database has more than one Join-Graph partition.
Fundamental assumptions

1) **We know ahead of time every query templates that could come to the system**

   - Prior knowledge of all the incoming query templates and the query workload give us the ability to select an appropriate table placement;

   - Knowing each query template in advance allows us to propose better solutions for balancing load across multiple servers for web based applications;

   - Queries that are not seen before are not guaranteed to be answerable at all because the query processing logic is not capable to answer them.
This thesis makes three fundamental assumptions:

1) We know ahead of time every query template that could come to the system;
2) We demand that each query will be answerable by a single database server;
3) Database has more than one Join-Graph partition.
Fundamental assumptions

2) Each query will be answerable by a single database server

- It might be a good idea for simplification purposes;
- Query routing logic can be simple and it should be able to be made very fast and effective;
- Cost of communications between two nodes that are needed to answer a query might be high;
- It might allow us to better minimize the response time of a database and eliminate the intercommunication cost.
The data replication middleware

- We **characterize** the problem as an AI search over layout
- Our **hypothesis** is that we can learn rules to capture human-like expertise and use these rules to better partition a given database
- It will employ a **machine learning approach** to induce rules to help govern the physical design of the database across multiple nodes
- By the help of the learned rules, it will be **capable to fit layout characteristics**, and the layout generation can be faster and faster
- It will perform the layout and empirically measure the cost, since we want to know what is effective and under what conditions
- **Optimization goal**: maximize the total system throughput and minimize the query response time by figuring out how to best distribute the data
- The system will incorporate four operators: full replication, horizontal and vertical partitioning, and de-normalization
- No transaction support, retrieval/UDI queries as independent operations
Core parts

- 1) The data placement problem
- 2) AI search
- 3) How to determine valid operators?
- 4) Possible partitioning algorithm
- 5) Machine-learning
- 6) Query Routing Logic
The data placement problem

- Given a query workload, that describes all the query templates for a web-based application, and the percentage of queries of each template that the application typically processes;
- Given this workload and the optimization goal, determine the best possible placement using four operators (FR, HP, VP, and DN) and arbitrary number of database servers answering each query by a single node;
- Our optimization goal: maximize the total system throughput and minimize the queries’ response time.
Core parts

- 1) The data placement problem
- 2) AI search
- 3) How to determine valid operators?
- 4) Possible partitioning algorithm
- 5) Machine-learning
- 6) Query Routing Logic
State-based search algorithm

- AI search algorithm to iteratively search the huge search space of the combined operators
- Beam-search algorithm
- Beam-search can potentially reduce the computation, the memory consumption, and the time of the search
State-based search algorithm

- A state represents the actual complete status of the system
- It generates all successors of the state at the current level, sorts them in increasing cost order and it only stores a pre-determined number of states at each level (beam width)
- If the beam width is small then more states are pruned, if the beam width is infinite then no states are pruned (breadth-first)
- Beam width bounds the memory required to do the search: less memory consumption
- The algorithm terminates if the required goal state is reached or no more states are left to be explored
State-based search algorithm

Beam Length=2
Number of nodes=3
Initial State: Single database
TSTG: Total System Throughput Gain

Each state represents the actual complete status of the system

Sort: D, C, B

Sort: F, E, H, G

- The system creates the layout and runs the workload
Search with back-jumping

- Back-jumping is a technique to increase the efficiency of the search method.
- Back-jumping can avoid the incompleteness of the search.
- Instead of going one level up in the search tree, back-jumping may go up more than one level if the state cannot be further expanded during the search.
Core parts

- 1) The data placement problem and algorithm
- 2) AI search
- 3) How to determine valid operators?
- 4) Possible partitioning algorithm
- 5) Machine-learning
- 6) Query Routing Logic
To be able to consider valid candidates and refine our search we propose to analyze the table schema and query dependencies.

- If state “B” creates a layout with horizontally partitioning table “users” by “id”, then queries that involve “users” by “name” are broken.
- What if there are join queries that involve other tables that are previously partitioned on a different key?
- If we analyze the query dependencies we are able to consider states that will not break the system and we can reduce the search space more effectively.
How to determine valid operators?

- Constraints define different rules regarding the values allowed in the database table or in a specific column
  - Check, Not-NUL, NULL, Unique constraints
  - Primary Key, Foreign Key constraints
  - Foreign Key identifies a relationship between two tables
    - A row in a referencing table cannot contain values that do not exist in the referenced table
How to determine valid operators?

- Table relationships
  - Table relationships can be one-to-one, one-to-many, and many-to-many
  - One-to-one
    - Each row in one table is linked to one and only one other row in another table
    - \( \text{LEN(TABLE A)} = \text{LEN(TABLE B)} \)

<table>
<thead>
<tr>
<th>userid</th>
<th>address</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100 Institute Rd.</td>
<td><a href="mailto:kann@wpi.edu">kann@wpi.edu</a></td>
</tr>
<tr>
<td>20</td>
<td>48 Worcester Rd.</td>
<td><a href="mailto:pl@wpi.edu">pl@wpi.edu</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>userid</th>
<th>firstname</th>
<th>lastname</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Tom</td>
<td>Anderson</td>
</tr>
<tr>
<td>20</td>
<td>Adam</td>
<td>Smith</td>
</tr>
</tbody>
</table>
How to determine valid operators?

- Table relationships
  - One-to-many
    - Each row in the related table can be related to many rows in the relating table
    - LEN(TABLE A) < LEN(TABLE B) (almost always)

<table>
<thead>
<tr>
<th>teacherid</th>
<th>address</th>
<th>email</th>
<th>classid</th>
</tr>
</thead>
<tbody>
<tr>
<td>234</td>
<td>100 Institute Rd.</td>
<td><a href="mailto:kann@wpi.edu">kann@wpi.edu</a></td>
<td>10</td>
</tr>
<tr>
<td>121</td>
<td>48 Worcester Rd.</td>
<td><a href="mailto:pl@wpi.edu">pl@wpi.edu</a></td>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>classid</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>biology</td>
<td>intro</td>
</tr>
<tr>
<td>20</td>
<td>chemistry</td>
<td>intro</td>
</tr>
<tr>
<td>30</td>
<td>history</td>
<td>advanced</td>
</tr>
</tbody>
</table>
How to determine valid operators?

- Table relationships
  - Many-to-many
    - Each row in the related table can be related to many rows in the relating table and vice versa

<table>
<thead>
<tr>
<th>userid</th>
<th>address</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>234</td>
<td>100 Institute Rd.</td>
<td><a href="mailto:kann@wpi.edu">kann@wpi.edu</a></td>
</tr>
<tr>
<td>121</td>
<td>48 Worcester Rd.</td>
<td><a href="mailto:pl@wpi.edu">pl@wpi.edu</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>userid</th>
<th>itemid</th>
</tr>
</thead>
<tbody>
<tr>
<td>234</td>
<td>10</td>
</tr>
<tr>
<td>234</td>
<td>20</td>
</tr>
<tr>
<td>121</td>
<td>20</td>
</tr>
</tbody>
</table>

Table A

Table B

<table>
<thead>
<tr>
<th>itemid</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>car wax</td>
<td>strong</td>
</tr>
<tr>
<td>20</td>
<td>light bulb</td>
<td>small</td>
</tr>
<tr>
<td>30</td>
<td>tire</td>
<td>snow</td>
</tr>
</tbody>
</table>
How to determine valid operators?

- The Constraint-Graph
  - The Constraint-Graph represents the constraints of the database and the table relationships in one;
  - The Constraint-Graph is obtained from the given database schema set;
  - The system will analyze the graph to determine dependencies between tables.
How to determine valid operators?

- The Join-Graph
  - The Join-Graph focuses on queries;
  - JOIN expressions and conditions;
  - It presents the WHERE expression of the queries attached to the graph;
    - SELECT * from users WHERE user_id=x
  - It identifies the different slices of joins as individual join-partitions ‘FARMS’;
  - Each node of the graph is a table and each edge is a join relationship;
The Join-Graph

FARM:A (a partition of the Join-Graph)

For example, user_details table has a JOIN expression with table class_assignments ON user_details.user_id=class_assignments.id WHERE class_assignments.id=X
How to determine valid operators?

- The Complex-Graph
  - The Complex-Graph is for integrating the Join-Graph with all queries into one data structure
  - Processing this structure and the constraint graph, the partitioning algorithm will be able to propose valid operators
  - Reduce the search space, do not consider invalid operators
  - After partitioning, the system has to process the complete workload (system functionality)
The Complex-Graph

Table user_details has two queries:
SELECT (user_id) FROM user_details WHERE user_details.id=X
UPDATE ( birthdate, grade, middle_name, last_name, first_name) WHERE user_details.id=X

The graph has information about the related queries (selects, updates, inserts, and deletes)
Core parts

- 1) The data placement problem and algorithm
- 2) AI search
- 3) How to determine valid operators?
- 4) Possible partitioning algorithm
- 5) Machine-learning
- 6) Query Routing Logic
Important assumptions

- Partition localization for a given query
  - Hash function for partitioning (avoid skewness)
- Correct result set generation
- Answering the query using a single node
  - Table placements
  - Problem with JOIN queries
  - Problem with aggregate functions and grouping columns
  - Update propagation problem
- Algorithm will be able to determine the partitioning key
- Algorithm will use the Constraint-Graph and the Complex-Graph as input parameters
Possible partitioning algorithm

**Constraint Graph:**
- relation between table 'answers' and 'problems' is one-to-many
- Primary Keys: problem.id, answers.answerid
- Foreign Key: answers.id

Is relationship one-to-one/one-to-many?
FARM has no other JOINS?
Select problem.id as partitioning key;
Has the JOIN a WHERE condition?
If yes, is the WHERE condition contains problem.id?
If yes, Check tables’ related queries: Q1;
Check that Q1 has a WHERE condition;
If yes, check that the WHERE condition does not create any problem;
If not, mark (problem.id-answers.id) as a valid HP key.

**Complex Graph:**
- FARM has no other JOINS
- JOIN has ‘WHERE’ condition
- Table answers has one related query:
  - Q1: SELECT * FROM answers WHERE answers.id=24
- Table ‘problems’ has no related query
- Assumed: no grouping and aggregate queries for the FARM

```sql
SELECT * FROM problems JOIN answers ON problem.id=answers.id WHERE problem.id=12
```
Possible partitioning algorithm

- What if we have multiple tables (A,B,C) with the same joining keys?

- What if we have multiple tables (A,B,C) with different joining keys?

- If state “B” creates a layout with horizontally partitioning table “users” by “id”, then queries that involve “users” by “name” are broken
- Column relationships need to be evaluated
- Consider de-normalization or FR(Table C) on all nodes
Update propagation

- Update propagation can be a problem
- Multi-master replication
  - store the data by a group of computers
  - update by any member of the group
  - PostgreSQL supports multi-master replication (Postgres-R)
    - Run on shared-nothing cluster
    - Prepare to distribute/distribute (two phase)
      - Maybe workload retrieval/UDI characteristics can help
- Oracle has a deferred queue which is periodically processed on all nodes (and log file propagation)
Core parts

- 1) The data placement problem and algorithm
- 2) AI search
- 3) Graph structures
- 4) Partitioning algorithm
- 5) Machine-learning
- 6) Query Routing Logic
**Hypothesis**: We can learn rules to capture human-like expertise and use these rules to better partition a given database

- We characterize the problem as an AI search over layout
- The system creates the layout and initiates the test run
- Exact sequence of instantiated queries is given
- Measure the execution time and gain of each query
- Measure the total system throughput gain for the created layout structure
Research Questions

- RQ1: Can we learn rules that are effective at speeding up the whole system?
  - By rules we mean like:
    - “when the number of Update/Delete/Insert queries on a table is small compared to the number of retrieval queries (e.g. selects), then one should fully replicate”;
    - “if there is a wide table but a lot of read queries are focused on a small set of columns of the table, then one should vertically partition”;
    - “when the table size is large, then one should horizontally partition”
  - What is a good cut off for “small”, “large”, or “wide”?
**Research Questions**

- **RQ1**: Can we learn rules that are effective at speeding up the whole system? (Cont.)
  - How can we go and parameterizing these rules for cut-off values?
  - How does CPU load effect these rules?
  - What other factors might count for instances when these rules are false?
  - Is a rule always true in practice? Can we learn more precise rules?
  - Will instances when a rule is not true help us to create more refined rules?
Research Questions

• RQ1: Can we learn rules that are effective at speeding up the whole system? (Cont.)
  – What are the possible set of important features that we need to take under consideration to learn a general rule?
  – We can consider:
    • Features of the workload;
    • Features of the tables;
    • Features of the queries;
    • Features of the created layout.
### Possible features I.

#### Features of the workload

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of queries</td>
<td></td>
</tr>
<tr>
<td>Total number of read queries</td>
<td></td>
</tr>
<tr>
<td>Total number of UDI queries</td>
<td></td>
</tr>
<tr>
<td>Total number of distinct read queries</td>
<td></td>
</tr>
<tr>
<td>Total number of distinct UDI queries</td>
<td></td>
</tr>
<tr>
<td>Most frequently accessed table</td>
<td></td>
</tr>
<tr>
<td>Least frequently accessed table</td>
<td></td>
</tr>
<tr>
<td>Most frequently accessed column</td>
<td></td>
</tr>
<tr>
<td>Least frequently accessed column</td>
<td></td>
</tr>
<tr>
<td>Read vs. UDI ratio</td>
<td></td>
</tr>
<tr>
<td>Percentage of queries for each query template</td>
<td></td>
</tr>
<tr>
<td>Time involved between retrieval and UDI queries</td>
<td></td>
</tr>
<tr>
<td>Frequency of occurrence of each query</td>
<td></td>
</tr>
<tr>
<td>Sequence of queries/ordering of statements (Web based application)</td>
<td></td>
</tr>
</tbody>
</table>

#### Features of table: schools

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of read queries accessing the table</td>
<td></td>
</tr>
<tr>
<td>Total number of UDI queries accessing the table</td>
<td></td>
</tr>
<tr>
<td>Total number of distinct read queries accessing the table</td>
<td></td>
</tr>
<tr>
<td>Total number of distinct UDI queries accessing the table</td>
<td></td>
</tr>
<tr>
<td>Total number of queries accessing the table and other tables</td>
<td></td>
</tr>
<tr>
<td>Most frequently accessed column</td>
<td></td>
</tr>
<tr>
<td>Least frequently accessed column</td>
<td></td>
</tr>
<tr>
<td>Column relationships</td>
<td></td>
</tr>
<tr>
<td>Distinct table values (cardinality of the table)</td>
<td></td>
</tr>
<tr>
<td>Distinct values of each column (cardinality of a column)</td>
<td></td>
</tr>
<tr>
<td>Table size</td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td></td>
</tr>
<tr>
<td>Primary key</td>
<td></td>
</tr>
<tr>
<td>Foreign key</td>
<td></td>
</tr>
<tr>
<td>Other key</td>
<td></td>
</tr>
<tr>
<td>Read vs. UDI ratio</td>
<td></td>
</tr>
</tbody>
</table>
### Possible features II.

**Features of query: Q1**

<table>
<thead>
<tr>
<th>Requested table/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table relationships</td>
</tr>
<tr>
<td>Requested column/s</td>
</tr>
<tr>
<td>Query condition/s</td>
</tr>
<tr>
<td>Join</td>
</tr>
<tr>
<td>Number of joins</td>
</tr>
<tr>
<td>Join condition/s</td>
</tr>
<tr>
<td>Query plan</td>
</tr>
</tbody>
</table>

**Features of the created layout**

| CPU loads of each node |
| Most heavily loaded node |
| Most lightly loaded node |
| Operator |
Research Questions

- RQ2: If these rules are effective in general, then does this system get more efficient with laying out databases over time?

  - Can we make the search for layout for a new system more efficient over time?
  - If the rules we learned are good, then we could use the rules themselves to bias a search for layout for a new database
Core parts

- 1) The data placement problem and algorithm
- 2) AI search
- 3) Graph structures
- 4) Partitioning algorithm
- 5) Machine-learning
- 6) Query Routing Logic
Query Routing Logic

- The Query Router (QR) routes the queries to the appropriate database
- QR needs to know about the created layout structure
- It should handle multiple database connections
- It should handle locking
- Application should be able to connect to QR without any modification
We propose different types of evaluation

- Based on the training workload, we can generate the layouts and empirically measure:
  - Query execution time and gain
  - Overall system throughput and gain
- Train to induce rules and use the test set for evaluation
- Learn not-obvious general rules, theory may not be enough
- Explore multiple ways to represent this knowledge (maybe decision-tree)
- Cross-validation
  - To prevent overfitting our rules to training data
Proposed evaluation

- We plan to use several different benchmarks
- Transaction Processing Performance Council (TPC) designed different models
  - TPC-H is a decision support benchmark
  - TPC-W is a benchmark for E-commerce
- Can we achieve better performance w/wo learned rules?
- IBM DB2 advisor
  - DB2 advises layout
  - Check our the system layout for the same workload
  - Compare the created layout structures
  - Search for evidence for our method
Implemented system components

- Simplified version of the placement algorithm
  - Query templates and workload
  - FR, HP, and VP operators
- AI search
  - Temporary Data structures
- Layout handling
  - Automatic physical layout generator
- Simplified Query Routing Logic
- Tester
## Implemented system components

<table>
<thead>
<tr>
<th>Start State</th>
<th>End State</th>
<th>Operator</th>
<th>Table</th>
<th>Query</th>
<th>QET Before [s]</th>
<th>QET After [s]</th>
<th>QET Gain [s]</th>
<th>OST Gain [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>HP(_{\text{user_id}})</td>
<td>user details</td>
<td>Q36</td>
<td>0.36</td>
<td>0.33</td>
<td>0.03</td>
<td>-0.14</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>HP(_{\text{login}})</td>
<td>users</td>
<td>Q39</td>
<td>0.328</td>
<td>0.28</td>
<td>0.005</td>
<td>+1.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Q41</td>
<td></td>
<td>0.359</td>
<td>0.358</td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>D</td>
<td>FR</td>
<td>users</td>
<td>Q39</td>
<td>0.328</td>
<td>0.5</td>
<td>-0.172</td>
<td>+2.72</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Q41</td>
<td></td>
<td>0.359</td>
<td>0.48</td>
<td>-0.121</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>E</td>
<td>HP(_{\text{user_id}})</td>
<td>user roles</td>
<td>Q37</td>
<td>0.203</td>
<td>0.146</td>
<td>+0.057</td>
<td>-2.41</td>
</tr>
<tr>
<td>D</td>
<td>F</td>
<td>FR</td>
<td>user roles</td>
<td>Q37</td>
<td>0.203</td>
<td>0.813</td>
<td>-0.61</td>
<td>-3.2</td>
</tr>
<tr>
<td>C</td>
<td>G</td>
<td>FR</td>
<td>roles</td>
<td>Q18</td>
<td>0.375</td>
<td>0.155</td>
<td>+0.22</td>
<td>+2.39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Q19</td>
<td></td>
<td>0.344</td>
<td>0.28</td>
<td>+0.064</td>
<td></td>
</tr>
</tbody>
</table>

QET: Query Execution Time, OST: Overall System Throughput

Workload: 88 queries, Number of database servers: 3
Not implemented system components

- Machine learning
- Knowledge bank
- Partitioning logic
- Denormalization operator
- Automatized Search logic
- Complex query handling
- Locking and queuing
- Automatic testing
Expected contributions

1) A data placement algorithm that can converge quickly over time as it trains over a set of examples and machine learned rules;

2) Parameterized and machine learned rules to help govern the physical design of the database across an arbitrary number of computer nodes;

3) A shared-nothing data replication middleware for Web-based applications that can be easily built using low-cost existing resources to realize database scaling possibilities without expensive storage area networks.
Project milestones

- (February-April) Creating a fully implemented working system;
- (March) Submit paper to VLDB 2010;
- (May-August) Generating training data set for Machine Learning and determine general rules. Apply Machine Learning and evaluate the system;
- (September) Submit paper with evaluated machine learned rules to WWW 2011;
- (September-October) Write up and submit dissertation.
Questions?

Thank You for Your Attention!