Synergy-based Workload Management

Martina-Cezara Albutiu
supervised by Alfons Kemper (kemper@in.tum.de)

Technische Universität München
Boltzmannstr. 3
Garching, Germany
albutiu@in.tum.de

ABSTRACT

Workload management aims at the efficient execution of queries on a database. In this context, scheduling plays a crucial role. A vast number of scheduling approaches have been developed, most of them belonging to one of two categories: analysis and monitoring. However, they mainly either focus only on one possible kind of impact of queries on each other's execution time, or require an offline phase for information gathering. In contrast, the approach we pursue does not require any offline phase and flexibly adapts to any database system or hardware configuration. It bases on the fact that the multiple requests that are executed concurrently in a database for performance purposes may have a positive impact on the execution of each other, e.g. due to caching or complementary resource consumption, or impede each other's execution, e.g. in the case of resource contention. Both kinds of impacts are reflected by the execution time of the workload. We apply a monitoring approach to derive those impacts – called synergies – between request types fully automated at runtime from measured execution times. Thereby, our approach works completely independent from changing synergies or configurations and easily handles new query types.

1. INTRODUCTION

In database systems, a set of queries is executed concurrently in order to provide a good utilization of the system's resources like processors and memory. The concurrently running queries may interact with each other and thus influence each other's execution time positively or negatively. A common example for positive interactions between queries running in a database system are caching effects. Negative interactions may occur in the case of resource contention. By detecting and exploiting positive influences between queries the throughput in the database system can be maximized. On the other hand, negative query combinations with unpredictable execution times can be avoided. The influence of a query on another query – we call this the *synergy* between the two queries – depends on a multitude of factors, e.g. the system's hardware and configuration, implementation details of the database operators, and programs running on the same system as the database. Thus, the prediction of query influences is a complex task and prediction models are often based on probably arguable assumptions.

In this paper, we present a different approach considering the database as a black box. By monitoring the execution times of different query sets at runtime, we draw conclusions regarding the synergies of queries. This information can also instantly be applied by the scheduling component in order to maximize throughput.

VLDB '09, August 24-28, 2009, Lyon, France

2. RELATED WORK

Workload management, which includes scheduling techniques, is a part of many products like the HP Workload Manager for Neoview [3], the IBM Query Patroller for DB2 [7], the Microsoft SQL Server [6], and the Oracle Database Resource Manager [9]. Krompass et al. [4] present an overview of current workload management techniques and implementations. Common approaches for scheduling are FIFO and priority-based. However, none of the commercial products considers influences of queries on each other.

Approaches that take into account impacts of queries can be classified according to the technique of impact detection: those based on *analysis* examine the workload queries in order to determine sources of synergies. For example common subqueries [2, 10] need only be optimized once for all the queries in which they occur. But also shared data is a source of performance gains, e.g. by using cooperative scans as introduced by Zukowski et al. [11]. Analysis techniques focus on one specific source of synergy which is then exploited. However, there are a great number of different sources of positive and negative synergy to be taken into consideration which may also neutralize each other. Furthermore, the analysis of queries is often a complex task requiring lots of information which is not always provided by a database system. Approaches based on *monitoring* by contrast consider the database system and the queries as black boxes. Information is gained by observing the execution of queries and drawing conclusions. O'Gorman et al. [8] compare the number of disk accesses for the pairwise sequential and concurrent execution of TPC-H queries. This way, they identify caching effects without analyzing the underlying data of both queries. However, this approach only detects pairwise synergies and requires the synergy detection to take place

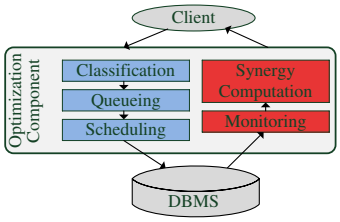


Figure 1: Architecture.

before the actual scheduling.

An approach which is similar to ours in terms of the computational model is that of Ahmad et al. [1]. However, their approach is only applicable to batch workloads and it requires an offline sampling phase.

3. APPROACH

Our approach for the detection and exploitation of query synergies is totally independent from the database system used and transparent to the clients issuing the database requests. In this section we present the underlying principles of the approach and the architecture of the optimization component.

Monitoring

We focus on a monitoring approach which considers the database system as well as the executed queries as black boxes. That is, no information about the resource requirements of requests or the configuration of the database system is available. Only measurements that can be obtained by monitoring the system serve as input to the optimization model. A short overall execution time indicates that the order in which the workload queries are executed permits to exploit sources of synergies like caching and at the same time avoids counterproductive effects like resource contention. Although it is not obvious what the sources of synergies are, the main goal of scheduling, i.e. minimizing the workload execution time, is met. Our approach therefore considers execution time as the main indicator for good scheduling.

Architecture

In order to realize the approach in a database independent way we use the architecture depicted in Figure 1: A middleware layer between client and database intercepts the client requests and forwards them to the optimization component. Here, the requests are first classified. The classification identifies queries only differing in parameter values but having the same SQL skeleton. Those queries are assigned the same type. Under certain assumption like uniformly distributed data queries having the same type will have the same execution characteristics (like CPU-intensive, short-running, etc.). Of course we are aware of the fact that in case of data skew a query q being short-running with parameter $x \in [0, 100]$ may become long-running with parameter $y \in [101, 110]$. The approach can easily handle such a case by including parameter values into the classification process, thus forming two types of queries q_1 and q_2 out of the SQL skeleton of q for the intervalls $[0, 100]$ and $[101, 110]$, respectively. Thereby, x and y can be determined using 2D classification techniques. The classified queries are then queued in the middleware. Whenever the database can accept new requests a scheduling algorithm chooses a set of queries out of the queue to be executed next. The size of the set corresponds to the multi-programming level (MPL) and is determined in advance such that the database load is maximized. After the execution of the query set has finished the monitored execution time is fed back into the optimization component where the synergies are computed.

Training and Optimization Phase

The approach consists of two phases: the training phase and the optimization phase. Depending on the phase, different scheduling algorithms are applied. During training, the main goal of scheduling is to determine synergies between queries. Having learned about those synergies, scheduling during the optimization phase makes use of them in order to maximize the throughput. In both phases, the knowledge base consisting of different query combinations and their execution times is continuously extended by feeding back new monitoring data. There is no general best point in time for switching from training to optimization phase. It depends on the database workload and the system load as well as on the amount of synergy information already obtained. The person in charge has to make a trade-off between completeness and accuracy of synergy information and the duration of unoptimized scheduling. However, as new information is continuously fed into the model not only during training but also during optimization, yet incomplete synergy information may be completed later on.

Synergy Computation

As stated above, we assume that the MPL has been determined beforehand in a way that maximizes the database throughput. This may be done based on expert knowledge. In order to accurately determine influences of certain queries on others, we execute the queries of a workload block-wise, i.e. only after all queries of one block have finished execution the next block is submitted to the database.

Within a block of MPL many queries, each of the queries influences the execution of each of the other block queries, either positively or negatively. In other words, there exist pairwise synergies between the queries of one concurrently executed block which either result in a shorter overall execution time (if the synergies are positive) or in a longer one (if they are negative). Thus, the overall execution time of a block of queries (rt, response time) can be represented by a linear combination of $\binom{MPL}{2}$ execution time shares s_{xy} of two query types x and y. The execution time for a block of four concurrently executed queries a, b, c and d is thus:

$$rt_{abcd} = s_{ab} + s_{ac} + s_{ad} + s_{bc} + s_{bd} + s_{cd}$$

Each share s_{xy} represents the influence that the concurrent execution of the queries x and y have on the overall execution time. If all of the block's queries have equal execution times and there are no synergies, the influences (shares) are equal. A block consisting of queries of different complexity (and thus different execution times) results in greater values of the shares containing complex queries and smaller values of the shares representing less complex queries. Also, if two of the queries have synergies, their share is smaller which also results in a smaller response time of the block.

The synergy of a query x with respect to another query y is determined by putting their execution time share s_{xy} in relation to the execution time shares of x with all known query types. The set of known query types is denoted by Q and may increase during runtime. We define the synergy syn_{xy} between two queries x and y, i.e. the impact on the

execution time of x caused by the concurrent execution of y, as the difference between the average execution time share of x with all other known query types q and the share of x and y:

$$syn_{xy} = \frac{\sum_{q \in Q} s_{xq}}{|Q|} - s_{xy}.$$

Yet unknown share values s_{xy} distort the computed synergy as subtracting 0 from $\frac{\sum_{q \in Q} s_{xq}}{|Q|}$ will result in the highest possible synergy value syn_{xy}. This is why we assume unknown values s_{xy} to be equal to the average of all known values for the given matrix row, so that the resulting synergy is 0. Contrary to the execution time shares, the synergy between two query types is not symmetric, i.e. $syn_{xy} \neq syn_{yx}$. Query x may benefit more from the concurrent execution of query y than the other way round, e.g. if the data needed by y is a superset of the data needed by x and caching is the main source of synergy. The determined synergies can be stored in a matrix, called the *synergy matrix*.

In order to determine accurate values of the execution time shares, each query pair is executed in several combinations with other query types in Q, thus collecting a number of response time values that form a linear system of equations of the following form:

$$rt_{abcd} = s_{ab} + s_{ac} + s_{ad} + s_{bc} + s_{bd} + s_{cd} + e_{abcd}$$

The slack variable e_{abcd} accounts for measurement errors or execution time fluctuation due to external influences. If a combination is executed repeatedly, the response time value rt is set to the average of the measured execution times. Thereby, the model adapts to execution time variations which may be accidentaly (e.g. by external influences) or reasonable (e.g. if the system configuration has been changed). The execution of new combinations provides new equations to the system thereby allowing a more precise solution to be determined.

By definition, solving a linear system of equations with n unknowns requires n equations. In our case, if we assume that $|Q| = 5$ there are $\binom{5}{2}+5 = 15$ different shares (of which $\binom{5}{2}$ are of the form s_{xy} with $x \neq y$ and 5 are of the form s_{xx}). Without considering the slack variables this would require 15 different combinations to be executed. From our example above you can conclude that at the beginning of the workload execution nearly every combination will contribute a few new unknowns and just one equation. Therefore, the linear system of equations is generally underdetermined and requires an optimization objective. We employ the principle of maximum entropy proposed by Markl et al. [5] for estimating selectivities of composed predicates. Following this principle we include all of the collected knowledge about combination execution times in the computation but don't make any further assumptions. The uncertainty due to the under-determination of the system of equations is distributed equally over the variables. Our objective function thus has two goals: to equally distribute the execution time over the different execution time shares and to minimize the slack variables:

$$opt = min \left\{ \frac{\sigma(S)}{avg(S)} + \sum_{e \in E} e \right\},$$

where σ and avg denote the standard deviation, respectively the average, E is the set of all slack variables e, and S the set of all shares s_{xy}.

The coefficient of variation $\frac{\sigma(S)}{avg(S)}$ models the equal distribution of the execution time over the shares. The error sum $\sum_{e \in E} e$ assures that the slack variables are minimized.

The following example briefly demonstrates the approach. We assume that the MPL is set to three and the system knows of four different query types a, b, c, and d by now. The following equations represent the executed combinations, the respective shares, and the time units the execution took:

$$rt_{abc} = s_{ab} + s_{ac} + s_{bc} + e_{abc} = 200$$
$$rt_{abd} = s_{ab} + s_{ad} + s_{bd} + e_{abd} = 180$$
$$rt_{acd} = s_{ac} + s_{ad} + s_{cd} + e_{acd} = 250$$
$$rt_{bcd} = s_{bc} + s_{bd} + s_{cd} + e_{bcd} = 260$$

After the combination a, b, and c is executed a second time, with a measured time of 190 time units, the equation system is updated by computing the average value of rt for the first equation:

$$rt_{abc} = s_{ab} + s_{ac} + s_{bc} + e_{abc} = 195$$

Figure 2 shows the development of the shares x_{xy} as well as the resulting synergy matrix.

Training and Optimization Algorithms

The scheduling algorithms during both the training and the optimization phase determine a group of MPL many queries to be executed next. During training, the main scheduling goal is to gather as much information as possible about synergies, i.e., to best fill the synergy matrix. This is done by systematically executing different query combinations and monitoring their execution times. Each of the monitored combination execution times provides either a new equation (and thus at least one new value in the synergy matrix) or contributes to a more meaningful average execution time of an already executed combination (and thus results in more precise values in the synergy matrix). We examined the following different scheduling algorithms for training:

FIFO The queries are executed in the order in which they arrive.

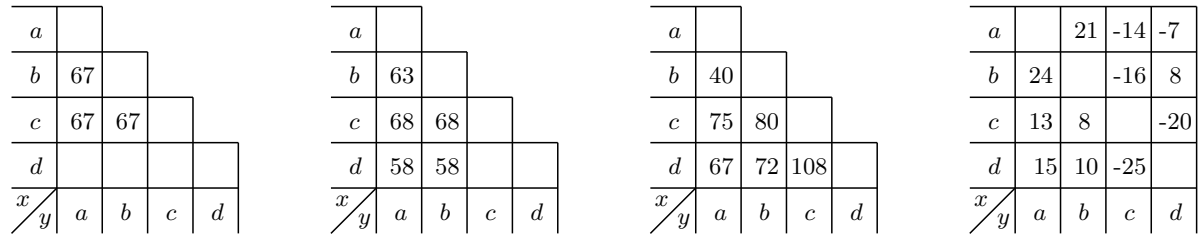
MaxUnknown The goal of this strategy is to quickly fill the synergy matrix by selecting combinations of queries which provide the most new synergy values.

MinUnknown The goal of this strategy is to fill the synergy matrix with as precise values as possible by selecting combinations of queries for which most of the variables in the system of equations are known.

MinComb This strategy selects combinations which have been executed the rarest. Following this strategy, the matrix will be enhanced by at least one new value after each combination execution. As soon as each of the combinations has been executed at least once the combinations' execution times are updated uniformly.

MinCombMaxUnknown The goal of this strategy is to combine the goals of *MinComb* and *MaxUnknown*, i.e. to quickly fill the matrix and uniformly update the system of equations.

MinCombMinUnknown The goal of this strategy is to combine the goals of *MinComb* and *MinUnknown*, i.e. to fill the matrix with precise values and uniformly update the system of equations.



(a) s_{xy} – one equation (b) s_{xy} – two equations (c) s_{xy} – four equations (d) synergies syn_{xy}

Figure 2: Development of the shares and the synergy matrix.

During the optimization phase, the main scheduling goal is to exploit the queries' synergies by selecting the query combinations to be executed based on the synergy matrix. Thereby, a greedy approach is employed which determines only one query combination to be executed at a time. The computation of a complete schedule for the workload has exponential complexity and therefore is not efficiently applicable. The scheduling algorithm *MaxSyn* chooses the queries with the highest combination synergy syn_c, which is computed as follows:

$$syn_c = \sum_{s_{xy} \in c} \left(syn_{xy} + syn_{yx} \right)$$

Some of the queries may have positive synergies while others may influence each other's execution in a negative way. Positive and negative synergies between query pairs within a combination can cancel each other out. However, the combination containing the most synergetic query pairs has the highest combination synergy value.

4. SIMULATION

We developed a database simulation framework which enables us (1) to evaluate different scheduling algorithms in a timely manner and (2) to verify the synergy matrix values. The second point is of particular importance as the presented approach is based on unsupervised learning and thus the solution cannot be validated using real data.

Simulation Framework

The simulation framework allows for the modelling of different types of queries, e.g. short-running, long-running, data intensive or processor intensive, and a given database system described by its cache size and resource capacities. Based on these parameters, the framework computes the execution time of the simulated queries corresponding to their resource requirements and impacts on each other. There are numerous sources of synergies, some of them like caching are obvious while others are not. In our simulation framework we had to choose a set of synergy sources on which to focus. Of course this set is not complete but it nevertheless provides a reasonable database simulation for our purposes. The following sources of (anti-)synergies are supported by the framework as described in Table 1: caching, complementary resource requirements, resource contention, lock contention, memory thrashing.

Caching and complementary resources are sources of positive synergy. Caching bypasses the slow disk access times and thus results in better performance. If computation intensive queries are executed concurrently with data intensive queries, they don't get in the way of each other with respect to resources, in contrary their execution results in a good utilization of the system resources. Resource and lock contention as well as memory thrashing are sources of negative synergy (anti-synergy). They prevent queries from progressing by not providing enough resource capacity and holding back needed data or paging it out of memory, respectively.

In the simulation framework, query types are identified by their total amount of used *processor cycles* and *disk operations*, and the required *main memory*. By varying the query type parameters we can model computation intensive queries like highly complex data analysis as well as disk intensive queries reading a lot of data. A high main memory value represents e.g. query types containing sorting or other main memory intensive functions. Furthermore, we model the *cache synergies* and *lock anti-synergies* between queries as partial binary functions. Cache synergies are given by $caching(q_1, q_2) = c, c \in \mathcal{N}$, where c is the number of disk operations that can be saved due to caching advantages. Lock anti-synergies are realized by reducing the number of processor cycles that can be assigned to a certain query by $locking(q_1, q_2) = l, l \in \mathcal{N}$. The simulated database is characterized by the processor and disk capacities, i.e. the processor and disk cycles the database can perform per second, and available main memory. Additional parameters are an *out of RAM penalty* and *maximum processor and disk quotas per query*.

The execution of a set of queries is simulated by (1) computing each query's share of the system resources and (2) determining the queries' progress assuming they are allocated the computed shares. The first step is conducted for the current point in time based on the queries' requirements and impacts on other queries and on the system parameters. In the second step, a short period of execution time is simulated during which the queries consume their resource shares. After that, the two steps are repeated for the new point in time.

Benchmarks

We conducted several benchmarks in order to efficiently evaluate different training algorithms in terms of their ability to fill the synergy matrix quickly and with accurate values. The more accurate the values in the synergy matrix are, the more precisely we can predict the execution time of a certain combination of queries during the optimization phase, thereby enabling us to schedule the queued queries in a way that maximizes throughput. The quality measure we employ in order to evaluate the accuracy of the synergy matrix values is the deviation of the estimated execution time

source of synergy	type	simulation
caching	positive	$caching(q_1, q_2) = c, c \in \mathcal{N}$, where c is the number of disk operations that can be saved due to caching advantages
complementary resource requirements	positive	query types with complementary requirements of different resources
resource contention	negative	query types with high requirements of the same resource
lock contention	negative	decrease of processor share (and thus progress) by $locking(q_1, q_2) = l, l \in \mathcal{N}$
memory thrashing	negative	limited database main memory, main memory requirements of query types, thrashing penalty

Table 1: Simulated sources of synergies.

of a combination c, denoted by $est\text{-}rt_c$, from the simulated (and later real) execution time $real\text{-}rt_c$, which we call the *error coefficient* ec_c:

$$ec_c = \frac{|est\text{-}rt_c - real\text{-}rt_c|}{real\text{-}rt_c}$$

The *prediction error* err_{pred} is defined as the average error coefficient over all possible query combinations of size MPL:

$$err_{pred} = avg(ec_c)$$

As this measure is meaningful only after the matrix has been filled for the most part we also consider a second measure, the *net prediction error*, which is computed the same way as the prediction error but only for not-empty matrix entries.

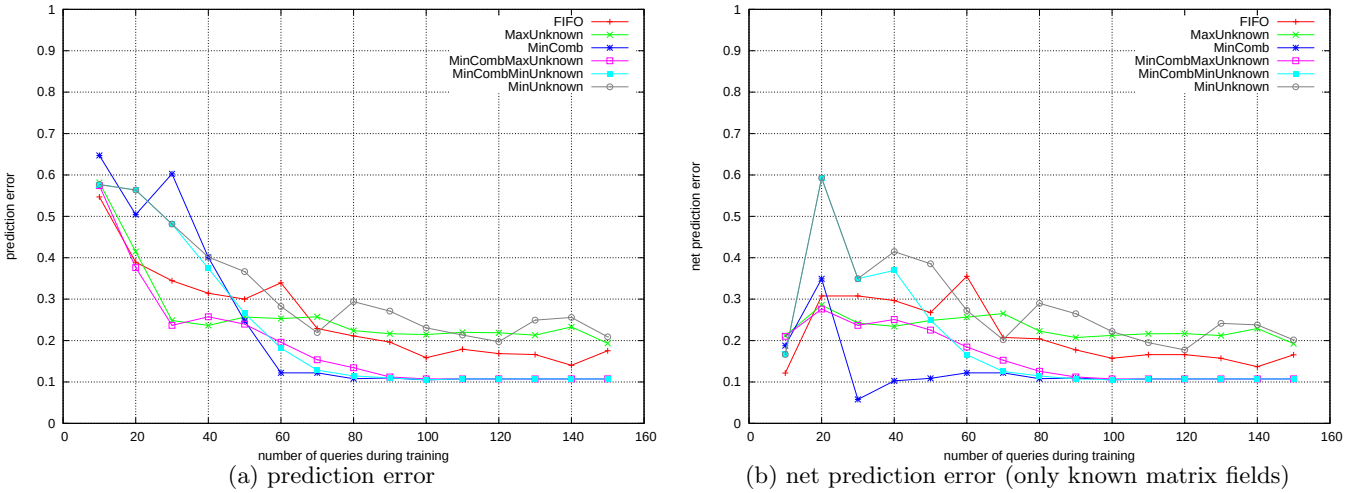
For our benchmarks we defined five different query types having similar numbers of disk operations. One of the types has constant disk and processor requirements over time while the remaining two pairs of queries are complementary in their requirements over time. The pairs also differ in their processor usage. All of the queries have more or less cache synergies and some of them have lock anti-synergies. We employed a MPL of three in the benchmarks.

Figure 3(a) shows the prediction error dependent on the number of queries during training for the different training algorithms. From a training phase of 70 queries, the prediction error is acceptable for each of the training algorithms, and does not change significantly anymore. *MinComb* and its variations *MinCombMaxUnknown* and *MinCombMinUnknown* perform best, *FIFO* lies in the middle field, and *MinUnknown* and *MaxUnknown* achieve the worst results. *MinCombMaxUnknown* and *MinCombMinUnknown* both quickly reach a small prediction error after only about 30 training queries while *MinComb* results in the greatest prediction error of all training algorithms with such a training length. When considering the net prediction error shown in Figure 3(b), the reason for this becomes obvious. *MinComb* has the smallest net prediction error while the net prediction error of *MinCombMaxUnknown* and *MinCombMinUnknown* is almost equal to the prediction error. Thus, the bad overall prediction error performance of *MinComb* follows from empty synergy matrix fields which of course result in an high error coefficient for the combinations containing the respective query pairs. The behavior of *FIFO* is as expected because by chance, as many good combinations (i.e. combinations that provide important information for the synergy matrix) as bad combinations (e.g. always the same combination not serving for new information) may be selected, thus resulting in a middle-rate performance. Also the prediction error of *MaxUnknown* is not surprising as this strategy quickly fills the synergy matrix with (unprecise) values, but does not effectively try to improve the values after that. The bad performance of *MinUnknown* compared to *MinCombMinUnknown* with regard to both the prediction error and the net prediction error can be explained by having a look at the implementation of those algorithms. Both of them prefer combinations with the least unknown variables, however the first criterion of *MinCombMinUnknown* is the number of occurrences for the different combinations with are determined in a sorted order. Therefore, it is more likely that combinations are selected which not only provide a precise new value but at the same time also provides new information for the already known variables which improves the precision of the synergy matrix in total.

The comparison of Figures 3(a) and 3(b) also confirmes another expectation. With five query types and the MPL being three there are 35 different combinations of queries (10 with different query types, 20 with two queries of the same type, and 5 with all three queries of the same type). Therefore, we expect the synergy matrix to be filled after a training phase of $35 \cdot 3 = 105$ queries. As the graphs in Figures 3(a) and 3(b) are quite equal for training phases of this length or longer, i.e. the net prediction error does not differ from the prediction error because of empty synergy matrix fields, our expectation is fulfilled.

5. CURRENT AND FUTURE WORK

At the moment we are conducting benchmarks on real database systems (in addition to our simulation framework) using the schema and data provided by the TPC-E and TPC-H benchmarks. However, as the trend is moving away from the separation of OLTP running on transactional databases and OLAP processed in data warehouses more and more to operational business intelligence, the diversity of workloads being executed in one database system will increase. In order to benchmark the applicability of our synergy model in such a scenario, we are planning to develop a benchmark environment which models highly heterogeneous information systems – hosting the data and handling the queries of various applications. Further, we plan to enhance the model as follows: As a first step the block size will be reduced so that the model will be scalable for the large MPLs expected in OLTP environments or infrastructures with high-performance hardware. Therefore, the model has to make use of groups, i.e. the combination size is not set to MPL, but to a smaller group size g. Of course, the MPL - g queries running concurrently with the group influence the group queries and the challenge will be to minimize those inter-group influences. Second, in order to fully exploit the MPL and thus the database resources, the block-wise ex-



(a) prediction error

(b) net prediction error (only known matrix fields)

Figure 3: Prediction error subject to the number of training queries and the training algorithm.

ecution has to be broken up, i.e. the number of concurrently executed queries in the database should always be filled up to MPL. Here, the challenge is to correctly gather information about synergies. As the queries are only partly executed concurrently, their impact on each other may vary depending on the amount of time they run together or on the queries' progress at the time they are processed in parallel. Another important point is that queries with negative synergies relating to most of the other queries run the risk of starvation as they are never scheduled. This problem has to be addressed by the scheduling component. Finally, in addition to only reacting to the arrival of queries at the middleware layer an advanced forecasting model which logs the frequency with which different query types are submitted and employs this knowledge in order to foresee the arrival of queries with high synergy values relating to already queued queries is planned.

In future work we are planning to develop a two-dimensional synergy model which considers a second measure in addition to execution time. The motivation therefore is that combinations resulting in the same execution time may have, e.g., completely different main memory requirements. This reveals the potential of increasing the MPL or freeing main memory (to be used for other applications) which may be communicated to the DBA.

6. CONCLUSION

Although workload management is a highly examined field in the database community, most of the up to now presented approaches focus on only one source of synergy, thereby ignoring the potential influences of other synergy factors. Others are not applicable at runtime as they require a preceding offline phase in order to gather information. We develop an approach which collects synergy information and flexibly integrates new query types at runtime. By using the execution time of query combinations as an indicator for synergies, we concentrate on optimizing the ultimate goal of scheduling and at the same time do not run the risk of ignoring any sources of synergy. Furthermore, our approach is completely independent of the underlying system's hardware or the database system used. In future work, the computed query synergies will be applied to efficiently schedule heterogeneous database workloads.

7. REFERENCES

[1] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala. Modeling and Exploiting Query Interactions in Database Systems. In *Proc. of the ACM 17th Intl. Conf. on Information and Knowledge Management (CIKM '08)*, Napa, California, USA, 2008.

[2] S. Choenni, M. L. Kersten, and J. F. P. van den Akker. A Framework for Multi Query Optimization. In *Proc. of the 8th Intl. Conf. on Management of Data (COMAD)*, pages 165–182, Madras, India, 1997.

[3] HP NeoView Workload Management Services Guide, August 2007.

[4] S. Krompass, A. Scholz, M.-C. Albutiu, H. A. Kuno, J. L. Wiener, U. Dayal, and A. Kemper. Quality of Service-enabled Management of Database Workloads. *IEEE Data Eng. Bull.*, 31(1):20–27, 2008.

[5] V. Markl, P. J. Haas, M. Kutsch, N. Megiddo, U. Srivastava, and T. M. Tran. Consistent Selectivity Estimation via Maximum Entropy. *The VLDB Journal*, 16(1):55–76, 2007.

[6] Microsoft SQL Server 2005 Books Online. `http://msdn2.microsoft.com/en-us/library/ms190419.aspx`, September 2007.

[7] B. Niu, P. Martin, W. Powley, R. Horman, and P. Bird. Workload Adaptation In Autonomic DBMSs. In *Proc. of the 2006 Conf. of the Center for Advanced Studies on Collaborative Research (CASCON '06)*, 2006.

[8] K. O'Gorman, D. Agrawal, and A. E. Abbadi. Multiple Query Optimization by Cache-Aware Middleware Using Query Teamwork. In *Proc. of the 18th Intl. Conf. on Data Engineering (ICDE)*, page 274, 2002.

[9] The Oracle Database Resource Manager: Scheduling CPU Resources at the Application Level. `http://research.microsoft.com/~jamesrh/hpts2001/submissions/`, 2001.

[10] N. Subramanian and S. Venkataraman. Cost Based Optimization of Decision Support Queries using Transient-Views. In *Proc. of the ACM SIGMOD Intern. Conf. on Management of Data*, pages 319–330, June 1998.

[11] M. Zukowski, S. Héman, N. Nes, and P. Boncz. Cooperative Scans: Dynamic Bandwidth Sharing in a DBMS. In *Proc. of the 33rd Intl. Conf. on Very Large Data Bases (VLDB)*, pages 723–734, 2007.