A Session-based Approach to Autonomous Database Tuning

Krisztián Mózsi, Attila Kiss

Eötvös Loránd University, Faculty of Informatics
Pázmány Péter sétány 1/C, 1117 Budapest, Hungary
mozsik@inf.elte.hu, kiss@inf.elte.hu

Abstract: By using autonomous tuning tools to optimize database systems, a lot of time-consuming, manual work can be automated. However, self-tuning database systems are trying to optimize global metrics of efficiency, they may set back rare, but critical functions of applications that use the database. The priority of application functions cannot be expressed in existing solutions, therefore, another approach may be needed. In this paper, a session-based method is presented, where application functions are represented as sessions, by building and using language models based on previous observations. With this technique, a similarity measure can also be defined, to interpret minor differences between sessions caused by program logic, as similarity. If usage patterns appear on user level as well, it is reasonable to construct user groups along similar behavior, to utilize such patterns. As the most significant part of an autonomous solution is forecasting, a method is also presented to predict future workload characteristics, by identified user groups. Then, this approach has been evaluated in practice, mainly to determine the optimal corpus size and validate session recognition.

Keywords: self-tuning database; autonomous system; dynamic database tuning; response time optimization

1 Introduction and Related Work

Optimization of database systems is quite a hard and diversified task, thus several subtopics can be identified. The main goal of them is usually common, namely to improve a numeric value, which characterizes database performance well. This metric may be throughput, response time or a sort of resource utilization of executed queries.

It is desirable to be concerned with the exploitable opportunities on the level of database management systems, and particularly the application which uses the database. Database administrators often try to find the most proper configuration and index set manually, but for this purpose, hundreds of parameters, helper

structures and settings should be chosen appropriately, furthermore future use cases and their correlations, patterns should be known in advance. This may be a very hard and time-consuming activity. Therefore, automatic solutions may be used to make this task less painful.

Offline advisor tools have already been implemented in commercial database management systems [1, 2]. These tools require a preliminary workload history, which describes future usage patterns well. This approach can only be used in really special cases, as it only generates an initially optimal configuration (e.g. index set), thus changes in workload patterns will not be taken into consideration.

In practice, it has proven to be a strict constraint to work with. Consequently a new, online approach was needed to handle dynamically changing workload patterns. The idea is based on the offline approach, but recommendations are generated repeatedly for predefined time intervals. This method would be more accurate when changes are tracked by observing the actual workload, thus new advises are generated by need. Two ways can be identified to manifest the recommended configuration changes: a database administrator can check these changes, or they can be executed in an automatic way. As generation and deployment might be a time-consuming task, adapting the database configuration to the workload change might be late [3].

By forecasting the expected change points, automatic, self-tuning database systems can be achieved. For this, it is essential that the change points are following some identifiable pattern and are not random. Such solutions are already implemented as part of commercial database systems, such as SQL Server 17+ and Oracle 18c auto-tuning functions, but an open-source solution called Peloton is also available.

Automatic tools are based upon an assumption that databases are often used by a higher-level application. Therefore, usage patterns appear along application functions. Nevertheless, these applications might have non-functional requirements that defines some features response time-critical. For these kinds of applications, it is not always a good idea to choose global response time as a value to optimize, because an automatic tool with such strategy would set back the response time of a rare, but critical function. As an example, consider a critical, insert-heavy operation, and a noncritical read operation on a concrete column of a table. Let us suppose that the read operation is called much more often than the insert. In this case, the global optimization technique is going to decide to create an index to decrease the response time of reads, even though response time of the critical function is going to increase because of the maintenance cost of the created index.

Nevertheless, all of the already existing tools mentioned above are trying to optimize a global value. Accordingly, a new approach is needed to get rid of this limitation.

The main contributions of this paper are the following:

- a method to overcome the above mentioned limitation,
- a technique to characterize users and their operations, to take users into consideration, if some of them behave similarly,
- a prototype implementation of the proposed approach, and its evaluation.

2 Sessions and User Groups

If priority values were assigned to each function of the application, the limitation, caused by the global optimization technique, could be solved during the deployment of indices. At the same time, as initial knowledge comes only from query logs, it is not trivial to identify these functions on the level of individual queries. For this, a language model is going to be constructed.

A function can be modeled as a sequence of query-transactions, which is called a session, and a transaction is defined as a sequence of queries. It is clearly visible that a method is needed to transform sequences of queries to sessions, based on the log, furthermore to create groups of the identified sessions, as multiple instances of the same – or almost the same – session are expected to appear in the log.

Sessions are issued by users of the upper-level application. It is desired to take users into consideration, if some of them are behaving similarly, for example, several bank clerks in an administration system of a bank are invoking similar functions of the system, as they fulfill the same work. A user can be described by the type of the invoked sessions, and users who invoke similar sessions can be grouped by this fact.

3 Life-Cycle of Self-Tuner Database Systems

The high-level behavior can be described by the observation-forecasting-reaction cyclically repeating triplet. Autonomous databases actively watch query history to reveal past correlations, then give predictions for the future based on the recognized patterns, and finally perform the corrections at a suitable time.

Autonomous database systems are expected to be fully automatic, that is, they should operate reliably without human intervention. This method does not require any database administrator to continuously fine-tune the system according to the

changes in the patterns. Note that the primary guarantee of the automatism is an appropriate forecaster component, which predicts based on past observations.

These systems proceed on a query log, which contains various information about past queries, such as the time when the query was executed, query text, executed physical plan and response time. The sequence of queries found in the log is characterized to retrieve relevant properties, on which a grouping could be based. A dimensionality reduction is needed, because making predictions for each query one by one would be too expensive. Sought solutions have identified three types of properties altogether, and each of them selected one, which suited their goals the most. By selecting physical properties to characterize queries, it is assumed that the most important attribute of a query is its resource utilization. It might be true, but it is not rewarding to build prediction systems upon physical properties, because changes of the schema, table sizes or the amount of resource affect these values. Another option is to choose logical properties, namely accessed columns and syntax tree of the executed query. Unfortunately, pattern recognition is quite hard in this case [4]. Grouping by similar arrival rate history of templates can be seen in QueryBot 5000 [4], which founds on the Peloton framework [3]. Firstly, the arrival rate patterns of query templates are collected. Then, a collection is created for each template by sampling these arrival rates, thus templates are clustered based on cosine similarity of the collections. It can be observed that queries of the same transaction belong to the same cluster, but disadvantages are exposed during clustering. Unfortunately, infrequent, but long-running queries are regarded as noise, furthermore similar transactions may belong to different groups because of minor differences caused by program logic.

After picking a predictable property regarding the constructed groups, forecasting models are built. By using such trained models, information about future behavior is obtained, then optimization is done based on that. Self-tuner systems should prepare for changing behavior, that is, new queries may appear, already observed ones may be replaced to another group or even disappear, therefore prediction models possibly become invalid.

4 Session Identification and Clustering

Identifying sessions is an appropriate way to group queries as well. In opposition to QueryBot 5000 [4], program logic differences are handled well, moreover, important queries are not considered as noise.

4.1 Preprocessing Query Logs

To handle similar queries together, the text of the queries should be canonized. The first step is to apply semantics-preserving transformations, such as regularizing alias names and transforming logical expressions into normal form. Existing solutions can be found [5] for this purpose. After this step, the parameters of the queries should be masked.

Equivalence of canonized queries can be defined based on the equivalence of the query texts, the similarity of execution time and size of the result table. By aggregating query templates that were considered equivalent, the number of considered data decreases. Unique identifiers are assigned for each template to reference them in the next steps.

4.2 Identifying Sessions

The goal would be to retrieve sessions from the previously cleaned data to recognize application functions. This process has two phases. Firstly, a statistical language model is built, then the bounds of sessions can be defined using the language model [6].

4.2.1 Language Modeling

However, defining a timeout would be an obvious way to separate a query sequence into sessions, it is not sufficient, as a timeout does not always indicate the beginning of a new session. More accurate splitting could be given by recognizing queries that frequently occur consecutively. For this purpose, a language model is going to be used, which is a probability distribution over arbitrary word sequences. By using a language model, the probability of a word sequence can be estimated.

Let us formulate the analogy with the language modeling domain, then clarify the previous statement formally. The identifiers of query templates can be regarded as words, and sessions made from them as sentences of a language. A sequence of sessions denotes the whole text. Let s = $<q_1, \ldots, q_n>$ be a session candidate sequence, consisting of template identifiers. The probability whether s is a valid sentence of the language can be defined using the chain rule.

$$P(s) = P(q_1) \dots P(q_n|q_1 \dots q_{n-1}) = \prod_{i=1}^{n} P(q_i|q_1 \dots q_{i-1}) \quad (1)$$

A type of language models is called n-gram, which assumes that the probability of the n^{th} element of a sequence depends only on the previous n-1 elements. That is, P(s) can be estimated as follows.

$$\prod_{i=1}^{n} P(q_i|q_1 \dots q_{i-1}) \approx \prod_{i=1}^{n} P(q_i|q_{i-n+1} \dots q_{i-1}) \quad (2)$$

The method works as follows. To train the language model, a long corpus is needed, which is preferably separated into sentences. Then, the sentences are separated to n-tuples of words, and also statistics are calculated. After that, the probability of n-grams can be determined from their relative frequency, and the probability of a sequence can be estimated using the equation above.

As training data for the model, session sequences are needed that describe patterns between queries well. Initially, sentence boundaries are not known, therefore well-separated training data is not available. Yet, login/logout of users or a predefined timeout would help to find boundary points, thus more accurate result can be achieved. This technique is called semi-supervised learning.

4.2.2 Neural Language Models

As n-gram models calculate probability by the number of query co-occurrences, complex correlations unfortunately cannot be noticed. For the purpose, neural language models can be used, which became popular in the field of language modeling [8]. Often recurrent neural networks, especially LSTMs are chosen. The task is to estimate the probability of a word based on a given context, which can be considered as a multiclass classification problem. Defining the length of the context (l_c) is not trivial, as it depends on the observable patterns. The simplest case is when $l_c = 1$, that is, the probability of each word is calculated by the previous word.

To train the model, 3-dimension input tensors are created with rank (batch size, l_c, 1), furthermore the expected template identifier is defined for each input. Note that the expected identifier should be one-hot encoded. The architecture consists of an embedding layer, which maps positive identifiers to vectors, a hidden LSTM layer, and finally a dense output layer with softmax activation function. Thus, for a template identifier input sequence with length l_c, a vector of probabilities is given as output, which can be thought of as a probability distribution. The i^{th} element of the vector is the probability that the i^{th} query template follows the input context based on the learnt language. Thus, the probability of a given template can be selected from this output vector. Note that if l_c is greater than the minimal size of a session, then a pseudo-identifier is needed to expand the size of the input sequence according to the rank of the input tensor.

4.2.3 Determining Session Boundaries

At this point, a trained language model is given, which is able to estimate the probability of arbitrary template sequences based on the foreshown training data. The primary assumption is that the level of uncertainty is roughly constant in a session, and it is measurable by empirical entropy [6]. Let SC be a sequence of template identifiers. Then, the probability, which is noted by P(SC), can be

estimated by the trained language model. In this case, empirical entropy is defined as follows.

$$H(SC) = -\frac{1}{n} \cdot \log P(SC) \tag{3}$$

With this, moving along an observed template identifier sequence, an identifier is added to SC in each step. In addition, the probability of the sequence is calculated in each step. Thus, the entropy can also be defined, and when it changes more than a predefined threshold, the start of a new session is found. Note that another dataset is needed for this operation than for language model training.

With this method, a long, raw sequence of query template identifiers can be separated into coherent subsequences, thus sessions can be retrieved that represent higher-level tasks of users.

4.3 Session Clustering

During the previous step, some potentially equivalent sessions are identified. Therefore, it would be desired to filter these duplicated sessions, and create groups which consist of similar sessions. This is a two-phase operation.

The first step is to form session classes. Let s_1 and s_2 be identified sessions. They belong to the same class if either of them is a subsequence of the other, or their Jaccard similarity is sufficiently high (e.g. exceeds a threshold limit, which is near 1), furthermore long-running queries should come from the same templates. High Jaccard similarity means that they share almost the same query templates. Let the representant element of the class be the longest session, which usually has the largest execution time as well. This step is inspired to correct minor, a few queries long sliding errors occurred during the identification phase, and to reduce the complexity of the next step. As it can be observed, the number of session representants to maintain is significantly reduced after this step, and converges to the number of task-types issued by users. The algorithm is the following. Initially, each session belongs to a separate class. Then, by using pairwise comparison, the condition of joining is checked. If it is true, the classes should be joined together. For this approach, the actual location of each session should be tracked. As pairwise comparisons are used, the method finishes in $O(n^2)$ time, where n means the number of sessions.

As queries often arrive by a program logic, several similar sessions may be found among class representants. For example, let s_1 = <1, 1, 2, 3, 4> and s_2 = <1, 1, 1, 2, 1, 4>, sessions represented by template identifiers. They can be considered similar, because clues of program logic are present. The number of the first elements are probably originated from a loop, and the cause of the difference between the second last identifiers may be a conditional statement. Let us call this program logic based similarity. This value can be measured by the Needleman-Wunsch algorithm [6, 7], which is a common method in bioinformatics. The goal

is to find the most exact matching between two sequences based on a score to maximalize. The scoring system is defined as follows. Matches are rewarded by 3 points, mismatches and insertion/deletion actions worth less (e.g. 1) points, thus sessions with more mismatches can be considered less similar. It is easily discernible that mismatches and insertion/deletion actions mean loops and branches.

The final output score is then defined as the weighted sum of points. The number of matches (nMatches), loops (nLoops) and branches (nBranches) are also considered during the score calculation. To construct a similarity measure (noted by PLS), this value should be normalized. For this, the highest reachable score is used.

$$PLS(s_1, s_2) = \frac{3 * nMatches(s_1, s_2) + nLoops(s_1, s_2) + nBranches(s_1, s_2)}{3 * \max\{|s_1|, |s_2|\}} \quad (4)$$

The borders of the function values and other similarity properties are easily verifiable.

$$0 \leq PLS(s_1, s_2) \leq 1 \quad (5)$$

By using program logic similarity and Jaccard similarity (noted by JS), a distance function is defined to group session classes.

$$d_{SC}(sc_i, sc_j) = 1 - \varepsilon_1 \cdot JS(sc_i, sc_j) - \varepsilon_2 \cdot PLS(sc_i, sc_j) \quad (6)$$

ε weights should be determined to sum up to 1. Since weighted similarity measures are subtracted from 1, d_{SC} is trivially a distance function.

Finally, an appropriate clustering algorithm should be picked. Methods that need the number of clusters as a parameter are certainly not suitable for this problem, as it cannot even be estimated in general. Furthermore, the algorithm should not reckon a session class as noise, just because it does not have enough neighbours to form a new group.

Regarding these aspects, using hierarchical, agglomerative clustering is reasonable. Initially, each session class forms a cluster, then by calculating the distance between them, some of them are merged. The procedure stops when there is only one cluster left. An appropriate clustering can be obtained by stopping earlier, using stopping criteria.

To put it into practice, besides the distance function between session classes, a distance function between clusters and stopping criteria is needed. Two possible options are identified to define distance between clusters, by reducting to elementwise distance. One of them is the centroid method, but choosing a central element is not trivial, as there is only a relative distance measure is defined between session classes. Therefore, single linkage method is chosen, which means the distance of the closest elements. Let $C_1 = \{sc_{11},\ldots,sc_{1m}\}$ and $C_2=\{sc_{21},\ldots,sc_{2n}\}$

be clusters of session classes. Then, D_{min} is defined as follows, based on d_{SC} distance.

$$D_{\min}(C_1, C_2) = \min\left\{d_{SC}\left(sc_{1i}, sc_{2j}\right) \middle| \; i \in [1..m], \; j \in [1..n]\right\} \quad (7)$$

Note that as $0 \leq d_{SC}(a,b) \leq 1$ for all a and b session class, $0 \leq D_{min}(A,B) \leq 1$ is also true for all clusters of session classes.

Using this function, distance-based stopping criteria can be also defined. The algorithm should stop when no more clusters are found near each other. Nearness is bounded by an ε_3 threshold.

5 User Clustering

If patterns are observable on the user level, it is desired to create user groups based on their behavior. The behavior is characterized by the distribution of the executed session types. Therefore. an n-dimensional vector is defined for each recognized user, where n is the number of recognized session clusters. The i^{th} element means the number of observed sessions which belong to the i^{th}session cluster. By dividing these values by the total count, ratios are obtained. These ratios are useful to determine the probability that an incoming template belongs to the given session group. Thus, a discrete probability distribution is defined for each user. Based on the similarity of the corresponding distribution, users can be categorized. Several methods can be found to define similarity between distributions. One of them is Hellinger-distance, which is defined using Bhattacharyya-coefficient [9]. Let p_1 and $p_2$probability mass functions of different discrete distributions. Then, Hellinger-distance is defined as follows.

$$H(p_1, p_2) = \sqrt{1 - BC(p_1, p_2)}, \; where \; BC(p_1, p_2) = \sum_{i=1}^{n} \sqrt{p_1(i) \cdot \; p_2(i)} \quad (8)$$

Since conditions are similar to clustering of session classes, hierarchical, agglomerative clustering is appropriate by using distance metric H here as well. Nearness is bounded by an ε_4 threshold.

6 Predicting Workload Patterns

The most important part of a self-tuning database system is the forecasting component, which predicts future behavior by recent recognitions. The concrete property to forecast depends on the chosen optimizable value and the method used during optimization. In QueryBot 5000 [4], a prediction model is created for each query template cluster, which predicts the amount of arriving queries for a given time interval ahead, based on recently collected arrival rates.

Similarly, it might be necessary to gain information about future queries for automatical index maintenance, e.g. how many of them will be executed and by which users. To utilize usage pattern similarity of users, let us create prediction models by user clusters. Then, the estimation of three potential properties could help answer the previous questions.

- Arrival rates, as in QueryBot 5000 [4]. In this case, an assumption should be made that the distribution of sessions is roughly constant in a group, as it is not predicted.
- The probability distribution of executed queries. Nonetheless, by picking this property, information about arrival rates is not available. Thus, a quite strong restriction should be introduced, namely, the number of executed queries cannot change significantly. Obviously, it is not true in general.
- The frequency distribution of executed queries. By combining the two properties above, information can be obtained about both distribution and quantity of sessions. Hence, this property may be the most practical choice.

Presence of patterns in the values is essential for proper forecasting, e.g. increasing or decreasing trends, cyclic behavior, periodically slow increasing then fast decreasing, etc. [4], otherwise reasonable and meaningful predictions cannot be given. However, unfortunately a general method to recognize all kinds of patterns universally does not exist, an ensemble model can be constructed [3, 4], which is a combination of prediction models to recognize as many patterns as possible.

As an example, let us briefly consider how may a prediction model be constructed to identify and forecast cyclic patterns of arrival rates and frequency distributions. For this purpose, just like for language modeling, LSTM is an appropriate choice [4]. For simplicity, let us give a solution for arrival rate prediction at first, then it is going to be generalized for frequency distributions easily.

Training data can be actually interpreted as parts of time series, linearly mapped to [0..1] interval. For the normalization, a maximal arrival rate value is required, which may be the largest value observed during the learning phase (session and user recognition), or a theoretical value, received as a parameter. Then, the sequence is split to its k+1 long subsequences. Finding the proper value of k depends on the length of the context which affects the $k+1^{th}$value. The model is then trained by the subsequences, by choosing an appropriate n_b batch size. For these, 3D input tensors are constructed with rank (n_b, k, n_f). In case of arrival rates, n_f = 1, but for frequency distributions n_f equals to the number of session clusters.

The architecture of the neural network is the following. As a hidden layer, two LSTMs are used. The activation function of the dense output layer is sigmoid,

therefore the output for a k-long sequence is a value, the predicted next element of the time series. In case of frequency distributions, this value is a vector, which represents the next, expected distribution of the following time unit. Since predictions should be usually made for multiple time units, the single forecasting method should be called continuously by utilizing the previously predicted value.

Note that groupwise model training can happen in parallel, even on separate processors, as these tasks do not depend on each other.

7 Live Phase

At this point, clustered users, their typical sessions, and a prediction for each user group are available. To generate helper structures, several already existing solution can be found [10, 11]. Nevertheless, it is practical to select significant queries that raise execution time the most as an input for the recommender system. Significant queries can be selected by defining a threshold, or by retrieving queries that run longer than the corresponding time slice. Before the final deployment, every index to be materialized should be checked whether it is consistent with predefined function priorities.

After model training, predictions and recommendations are generated periodically during the live phase of the system. Moreover, recognitions should be maintained based on continuously arriving query logs, as new queries, sessions, or even users may appear, or already existing ones disappear. They may change clusters, thus prediction models should be retrained.

8 Evaluation

A prototype has been created for demonstration and experimental purposes. The main goal is to compare and evaluate n-gram and neural network-based language models, by finding the minimal corpus size for training, and evaluating their session recognition ability. The advantage over already existing solutions is also demonstrated. Time complexity and the number of maintained models is specifically measured and compared to the most developed open source solution, Peloton [3]. Free parameters are also selected via example executions.

8.1 Experimental Setup

However, an appropriate workload history is indispensable for the evaluation, producing such history manually is quite infeasible. Thus, a program has been

created to generate workloads, by connecting to a test database, then executing transactions that follow a predefined pattern. Query logging should be enabled to retrieve the required data.

The concrete database schema, fitting data and transactions are received from the OLTPBenchmark project [12], which provides a framework for database performance testing. CH-benCHmark, which is built upon TPC-C standard, consists of OLTP and OLAP transactions, so it suits our needs. Furthermore, since a workload for several hours or days may be needed, rescaling should be done to transform the workload to a longer time interval. Generated workloads can be found in Table 1.

Table 1
Generated workloads for the evaluation

	test1.log	**test2.log**
Experiment 1	1 000 queries	a few seconds long history, 4 transactions
Experiment 2	10 000 queries	a few seconds long history, 4 transactions
Experiment 3	15 000 queries	a few seconds long history, 4 transactions
Experiment 4	43 000 queries	a few seconds long history, 4 transactions
Experiment 5	10 000 queries	4 hours long history with periodic patterns, 26 000 queries
Experiment 6	10 000 queries	6.5 hours long history with periodic patterns, 52 000 queries

Training queries for language models are in test1.log files, and test2.log files contain queries for session recognition and forecasting. With experiment 1, 2 and 3, the length of the corpus is determined, so that language models can identify session bounds sufficiently. With the other experiments, the execution time and dimensionality reduction are measured.

For the evaluation, a local PostgeSQL server was used, on an average PC configuration with 8 GB RAM and 4 cores. For logging, auto_explain option is set to true.

8.2 Determining Corpus Size

Firstly, let us identify the corpus size that is sufficient for the language models to recognize session bounds well. Beforehand, n for the n-gram model should be

defined. Let us choose n = 5, as training does not take too much time with this choice for large data, but a quite exact result is achievable.

At the first experiment, test1.log contains 1000 out of the 43000 queries, scaled to 1 day. Recognition of the neural language model is not exact, as initially 6 clusters were identified, because it has needlessly split sessions at multiple points. After the fine-tuning of the epoch number and the size of the LSTM layer, the number of clusters became 5, by less splitting. Although the 5-gram model identified 4 session clusters, it also split sessions at undesired points, just like the neural model.

During the second experiment, language models were trained with 10000 queries. The results were similar to the first experiment, but the loss function (mean squared error) of the neural model began to take its appropriate form.

Table 2
Summary of the evaluation results

	5-gram model	**Neural model**
Experiment 1	4 session clusters, but undesired splits	initially 6, then 5 session clusters, undesired splits
Experiment 2	4 session clusters, but undesired splits	5 session clusters, less undesired splits
Experiment 3	4 session clusters, proper session recognition	5 session clusters, identifying more correlations, form of loss function is correct
Experiment 4	4 session clusters, proper session recognition	4 session clusters, proper session recognition
Experiment 5	model training: 1637 ms recognition: 2907 ms clustering: 5825 ms 4527 recognizedsessions, 105 session classes, 23 clusters	model training: 58043 ms recognition: 38100 ms clustering: 10565 ms 3232 recognizedsessions, 82 session classes, 40 clusters
Experiment 6	model training: 1324 ms recognition: 3523 ms clustering: 5018ms 9122 recognizedsessions, 114 session classes, 21 clusters, 2 user groups	model training: 62927 ms recognition: 65481 ms clustering: 6867 ms 6447 recognizedsessions, 109 session classes, 41 clusters, 2 user groups

The 5-gram model recognized properly session bounds during the third experiment, hence a workload of 10000-15000 queries is necessary for corpus.

The neural model still identified 5 clusters, but the form of the loss function became correct. Based on the results of the fourth experiment, at least 40000-45000 queries are needed as a corpus to train the neural model, thus 4 clusters are identified. The fifth and sixth experiment cover two general cases, where particularly dimensionality reduction and execution time were inspected.

It is clearly visible that the 5-gram model often splits workloads for more sessions than the neural model, which can recognize more complex correlations. The neural model identifies more clusters by utilizing inter-transactional patterns, than the 5-gram model, which rather splits workloads along executed transactions.

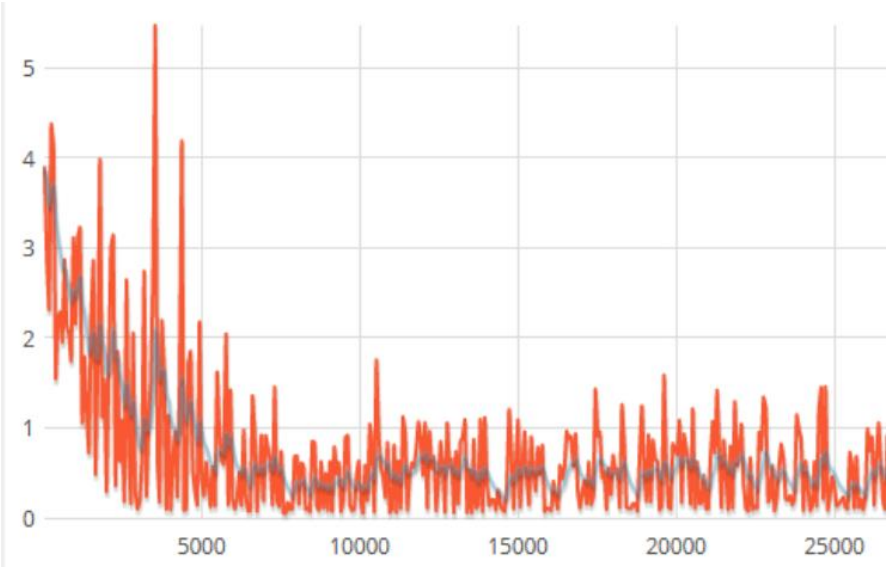

Figure 1
Loss function over the iteration count of experiment 3

8.3 Dimensionality Reduction and Execution Time

Workloads that have similar characteristics (e.g. cyclic behavior pattern, 2 user groups) and consist of 52000 queries, are split to 9122 sessions by the 5-gram model, and 6447 sessions by the neural model. QueryBot 5000 [4] templatizes incoming queries, then groups them by column access similarity, thus 334 templates are retrieved. The 5-gram and neural models construct 21 and 41 session clusters, respectively. Based on the clusters, user groups are created, then forecasting models are built. In contrast, QueryBot 5000 identifies 107 template clusters, and assumes that the top 5 largest clusters cover important patterns, therefore, prediction models are built only for them. It may be a strong

assumption, furthermore in this case, more models are created, thus space and time complexity is larger. Summarized results can be seen in Table 2.

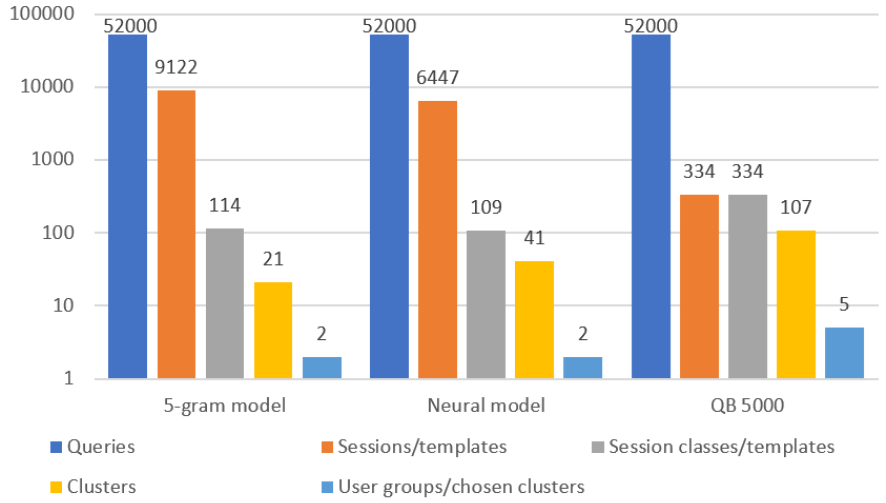

Figure 2
Comparing query number reduction on a logarithmic scale

Session recognition is the first subtask which may have significant execution time. As seen above in the presented approach, it is a two-phase process, which has the same goal as grouping query templates by the cosine similarity of their global arrival rate pattern histories, as seen in Peloton [3] and QueryBot 5000 [4]. Thus, the execution time of these subtasks should be compared.

Table 3
Comparison of execution times, 52 000 queries

	QueryBot 5000	**5-gram model**	**Neural model**
Preparing	8 162 ms (templatizing)	1 324 ms (model training)	62 927 ms (model training)
		3 523 ms (session recognition)	65 481 ms (session recognition)
Clustering	46 425 ms	5 018 ms (session)	6 867 ms (session)
		1 551 ms (user)	1 423 ms (user)
Total	≈ 54.5 sec	≈ 11.4 sec	≈ 136.5 sec

As we can see, the time complexity of QueryBot 5000 is larger than the 5-gram model, but the neural model performed much worse because of slower model training. As it can be seen in Table 2, the time cost of 5-gram model training and session recognition did not increase significantly because of larger data size. In the

case of the neural model, these phases have the most notable time complexity, but the time needed for clustering seems roughly constant.

By modifying the test data that is provided by the authors of QueryBot 5000 to contain similar sessions, it is verifiable that it does not recognize program logic similarities, and such sessions are going to be placed into different groups. On the other hand, by running experiment 5 and 6, it is shown that the demonstrated session-based method is able to recognize such similarities.

8.4 Evaluation Summary

Let us summarize the results of the evaluation above. It has been recognized that Peloton [3] and QueryBot 5000 [4] which are regarded as the most developed open-source, noncommercial self-tuning solutions available, cannot recognize similarity based on program logic. Thus, potentially similar user tasks are arranged to different clusters. This problem is eliminated by the session-based approach, and two language model implementations were compared. By comparing identified sessions and execution times, it became clear that neural models can also recognize inter-transactional patterns, but its cost is the notably slower training.

Several free parameters have been left undefined in the demonstrated method, which were selected empirically during the evaluation. The summary of these parameters and their suggested values can be found in Table 4.

Table 4
Suggested values of free parameters

Parameter	Value	Description
ε_1	0.3	The weight of Jaccard-similarity
ε_2	0.7	The weight of program logic similarity
ε_3	0.2	The maximal distance of a new element from a session group
ε_4	0.2	The threshold of Hellinger-distance for users in the same group

Conclusions

In this paper, a session-based approach was presented for autonomous database tuning, to let self-tuning systems take upper-level application requirements into consideration. In contrast with available solutions, the concept of session was made explicit, as it is an important notion for such systems. Because of this, sessions with minor differences due to program logic can be recognized as similar

much easier. For session recognition, details of a neural language model were specified, based on n-gram models. It was shown in practice that the neural model performs much slower due to the model training than the 5-gram model, but it did recognize inter-transactional, more complex patterns.

A method to characterize users and construct groups was elaborated, then for each user group a way to define appropriate forecasting models was presented, thus two properties of future workloads became predictable.

Further research would be beneficial to determine the correct training data size of the language models in general, as it can be different for each application. Experimenting with LSTM language models would be also practical. By building deep, more complex architectures, exciting results could be achieveable.

Actually, identifying sessions and clustering users is useful not solely for database tuning, but query recommender systems as well. When queries are manually written, it is desired to get automatic help, such as autocomplete or prediction of the next query, based on observed habits of the user group.

Acknowledgement

The project has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

References

[1] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, M. Ziauddin: AutomaticSQL Tuning in Oracle 10g, Proceedings of the 30th International Conferenceon Very Large Databases, 2004, pp. 1098-1109

[2] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya, M. Syamala:Database Tuning Advisor for Microsoft SQL Server 2005, Proceedings of the 30th International Conference on Very Large Databases (VLDB) 2004, pp.1110-1121

[3] A. Pavlo, G. Angulo et al.: Self-Driving Database Management Systems, CIDR2017, Conference on Innovative Data Systems Research, Vol. 10, 2017, pp. 781-792

[4] L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, G. J. Gordon: Query-based workload forecasting for self-driving database management systems, Proceedings of the 2018 International Conference on Management of Data, ACM, 2018, pp. 631-645

[5] G. Kul, D. T. A. Luong, T. Xie, V. Chandola, O. Kennedy, S. Upadhyaya:Similarity Metrics for SQL Query Clustering, IEEE Transactions on Knowledgeand Data Engineering, Vol. 30, 2018, pp. 2408-2420

[6] Q. Yao, A. An, X. Huang: Finding and Analyzing Database User Sessions,Database Systems for Advanced Applications 2005, 2005, pp. 851-862

[7] J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, E. Turricchia: Similarity Measuresfor OLAP Sessions, Knowledge and Information Systems (KAIS) Vol. 32, 2014, pp. 463-489

[8] M. Sundermeyer, R. Schlüter, H. Ney: LSTM neural networks for language modeling, Thirteenth annual conference of the international speech communicationassociation, 2012

[9] K. G. Derpanis: The Bhattacharyya Measure. Mendeley Computer, 1(4) 2008, pp. 1990-1992

[10] K.-U. Sattler, M. Luehring, K. Schmidt, E. Schallehn: Autonomous Management of Soft Indexes, Data Engineering Workshop, 2007 IEEE 23rd International Conference on. IEEE, 2007, pp. 450-458

[11] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis: On-line index selection for shifting workloads. ICDEW '07 Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop, 2007, pp. 459-468

[12] D. E. Difallah, A. Pavlo, C. Curino, P. Cudre-Mauroux: OLTP-Bench: Anextensible testbed for benchmarking relational databases, Proceedings of theVLDB Endowment, Vol. 7, 2013, pp. 277-288