Generating Shifting Workloads to Benchmark Adaptability in Relational Database Systems

Tilmann Rabl[1], Andreas Lang[1], Thomas Hackl[2], Bernhard Sick[3], and Harald Kosch[1]

[1] Chair of Distributed Information Systems
[2] InteLeC-Zentrum
[3] Computationally Intelligent Systems Group
University of Passau,
Germany
`{rabl,langa,hackl,sick,kosch}@fim.uni-passau.de`
`http://www.fim.uni-passau.de`

Abstract. A large body of research concerns the adaptability of database systems. Many commercial systems already contain autonomic processes that adapt configurations as well as data structures and data organization. Yet there is virtually no possibility for a just measurement of the quality of such optimizations. While standard benchmarks have been developed that simulate real-world database applications very precisely, none of them considers variations in workloads produced by human factors. Today's benchmarks test the performance of database systems by measuring peak performance on homogeneous request streams. Nevertheless, in systems with user interaction access patterns are constantly shifting. We present a benchmark that simulates a web information system with interaction of large user groups. It is based on the analysis of a real online eLearning management system with 15,000 users. The benchmark considers the temporal dependency of user interaction. Main focus is to measure the adaptability of a database management system according to shifting workloads. We will give details on our design approach that uses sophisticated pattern analysis and data mining techniques.

Key words: Benchmarking, Adaptability, Polynomial Approximation, Time Series Generation

1 Introduction

More and more database systems feature autonomic processes for optimization and adaptation. Nearly all major database vendors offer offline database design advisors [1, 2, 3] and recent research considers the online tuning of database systems [4, 5]. Certainly the query workload is the most important variable for physical tuning during runtime. New developments in database benchmarks start to face this trend. For example, TPC-DS [6] features a new query generator that allows to generate a large set of queries which are syntactically different but semantically similar [7]. Still synthetic query streams are usually homogeneous in the frequency of queries and the ratio between different query types, while real database workloads tend to be bursty [8]. Traditionally the workload is seen as a set of SQL query classes and the physical design is tuned accordingly. However, new approaches define it as a sequence [9] or chain [10] of statements. This offers new opportunities to adapt the database system. Nevertheless, there is only little research on how to analyze the efficiency of such systems. To the best of our knowledge there is only one publication that introduces a benchmark for autonomic database tuning [11], yet this benchmark also only features homogeneous workloads.

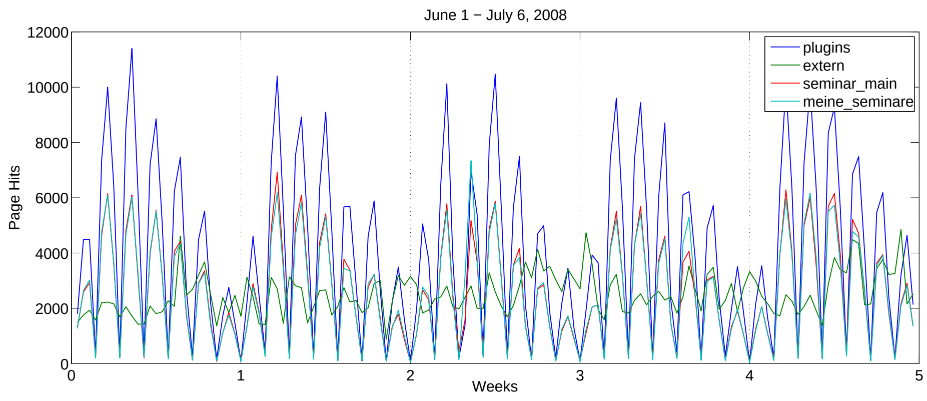

Fig. 1. Most accessed web sites in June 2008 per 6 hours.

Even though database access in most cases is triggered by human interaction, programs generate the actual SQL code. Therefore most queries are very similar and can be divided in relatively few distinct classes. Within these classes usually only simple parameters, like predicates change. Due to user interaction the occurrence of the classes depends on timetables. The most important examples are the day and night rhythm and the week cycle. In figure 1 this can be seen clearly for the accesses of an online eLearning portal (see section 2 for more details). It is easy to see that there is a daily and a weekly period. Each of the website accesses displayed will generate at least one and in most cases a sequence of SQL queries. For one website the queries will only differ in form of variables. Apart from the workload difference between day and night and workday and weekend, shifts in the workload between the single classes can also be seen. In figure 2 an average of the days in the data above is pictured. Not all websites are accessed in the same pattern. Thus, depending on the time of day the database will have different access rates and different access patterns.

Similar access patterns can be seen for any user accessed information system, see for example the access rates at the Wikimedia clusters[1] in figure 3. This

[1] The Wikimedia foundation is a non-profit organization that hosts various websites, most notably the online encyclopedia Wikipedia.

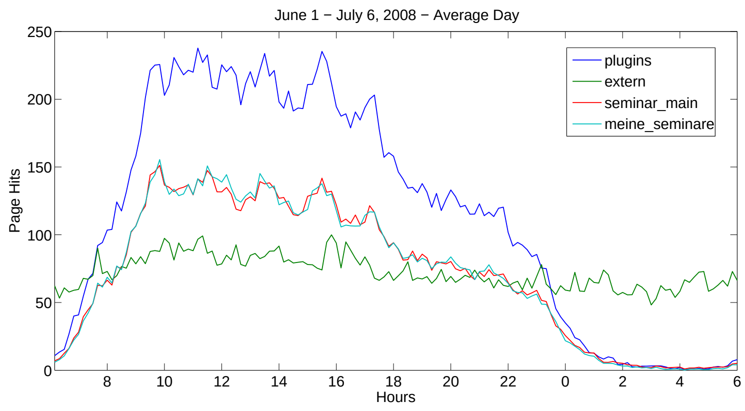

Fig. 2. Most accessed websites in June 2008, average day per 10 minutes.

periodic behavior gives chances for optimizations. On the one hand peak loads get more predictable and in times of low access the database can be prepared for the higher load. Such preparations could be index tuning or data restructuring. On the other hand clusters can be scaled according the access rates, in order to save energy or use the free capacity for other time-independent tasks.

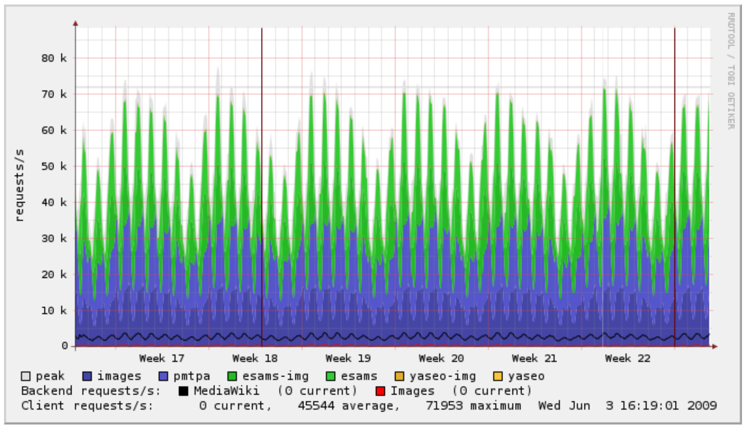

Fig. 3. Requests per second at the Wikimedia clusters in April 2009 in Europe (green) and the USA (blue) (image source: `http://en.wikipedia.org/wiki/Most_viewed_article`).

In this paper we introduce a benchmark that is based on a real online information system. We designed it to test our dynamic allocation algorithms for cluster databases [12]. The main design focus was to build a realistic workload model that reflects user dependent workload patterns. The implemented query generator is able to simulate realistic workloads that shift in quantity and ratio of the statements. Another focus lay on the data generation. To generate datasets in arbitrary sizes, we analyzed the value and reference distributions in the original database and built an data generator that supports different probability distributions.

The rest of the paper is organized as follows, in section 2 we will give details about the eLearning information system which is the basis of our benchmark model. After that we will describe the benchmark database layout, query set and workload definition. In section 4, we will show possible benchmarking objectives, like measuring the adaptability of a database system, before concluding with future work in section 5.

2 Application Domain

As the focus of the benchmark lies on changing workloads, online information systems are a promising application domain. Usually it is very hard to get any detailed information about the structure and especially the workload of such systems, since they are treated as industrial secrets. We are in the fortunate position to have access to a sufficiently large online eLearning platform that is used at the University of Passau, which is therefore the basis of our benchmark.

Stud.IP[2] is a popular eLearning management system. It started as a simple forum and evolved into a full-featured Course and Campus Management System over the years. The system supports the complete course life cycle, beginning with creating the course, filling it with data, assigning times and rooms, specifying application procedures and exporting the data into PDF or HTML. Online communication and cooperation are encouraged by providing a forum for each course, wiki, messaging system, chat and online material. Today, 38 universities and 16 other institutes, are using Stud.IP[3], one of them is the University of Passau.

Stud.IP is written in PHP and uses a MySQL database. New functions can easily be added by using the provided plug-in interface. The database schema consists of 198 tables.

On a normal day during the semester, between 50 and 100 parallel users are online at any given time. At the beginning of a new semester, this number is drastically higher, normally there are about 200-300 users online at the same time. The normal MySQL load is at about 1,200 database requests per second as each PHP page generates several database requests.

In the spring semester of 2009, there are 1,734 courses with a total of 15,047 registered users of which 1,374 have a teacher role. Among those users, 672 teachers and 7,072 users in student role logged in at least once during the semester. 6,921 of those student role users are registered in courses with a total of 63,895 course registrations.

[2] Stud.IP - `http://www.studip.de`
[3] `http://www.studip.de/nbu.php?page_id=9cd4b3aac2bfe40abc26fcc0ba6254ce`

Since the launch of Stud.IP in fall 2006, 8,907 courses were entered, 222,349 course registrations processed, 52,017 documents uploaded and 178,070 internal messages sent. The database has 7,688,642 entries and is 1.3 GB in size.

3 Benchmark

The basic benchmark design is generic, so that a variety of database systems could be modeled. The current implementation is based on the Stud.IP eLearning platform. The database schema is a reduction of the original schema to the core functionality. The data generation is hard coded to this database layout, but it supports various database sizes. To generate realistic data the attribute value and reference distributions were analyzed and modeled with probability distributions.

The main contribution of the benchmark is the query generation. Since the goal of the benchmark is to represent temporal dependencies in the database access, attention was especially paid to modeling query streams. The benchmark emulates the access behavior of students on Stud.IP based on web server logs from the University of Passau. In the following we will detail on the analysis of the original system and the according realization in the benchmark.

3.1 Database Design

The database schema is only a fraction of the Stud.IP schema as it is used at the University of Passau. For simplicity reasons it is reduced to the core functionality, thus it only consists of 25 tables compared to the nearly 200 tables in the production system. The schema can be seen in figure 4. In the following, we will give a brief explanation of the main tables and the relationship between them.

The tables `users` and `user_info` store all information about the users, which may be students, teaching staff or employees. `seminar` contains information about seminars, which may be lectures, tutorials or seminars. Which user is registered in which seminars is stored in `seminar_user`.

Each seminar has one or more courses, which are stored in `courses`. Each course has one or more lecturers which are stored in `course_lecturer`. In a course students can work in teams, for example for assignments. Each team is stored in `teams`. The relation `courses_user` stores in which course and team a student is.

The tables `dokumente` and `folder` represent all existing documents. These documents and folders are linked to each other via `eigeneDateien_links`. This relation links seminars to their root folder, folders to subfolders and folders to documents. The table `permissions` manages and stores user permissions to documents and folders.

Each user has an `inbox` and an `outbox` which stores references to all messages he has received or sent. The messages themselves are stored in the table `messages`.

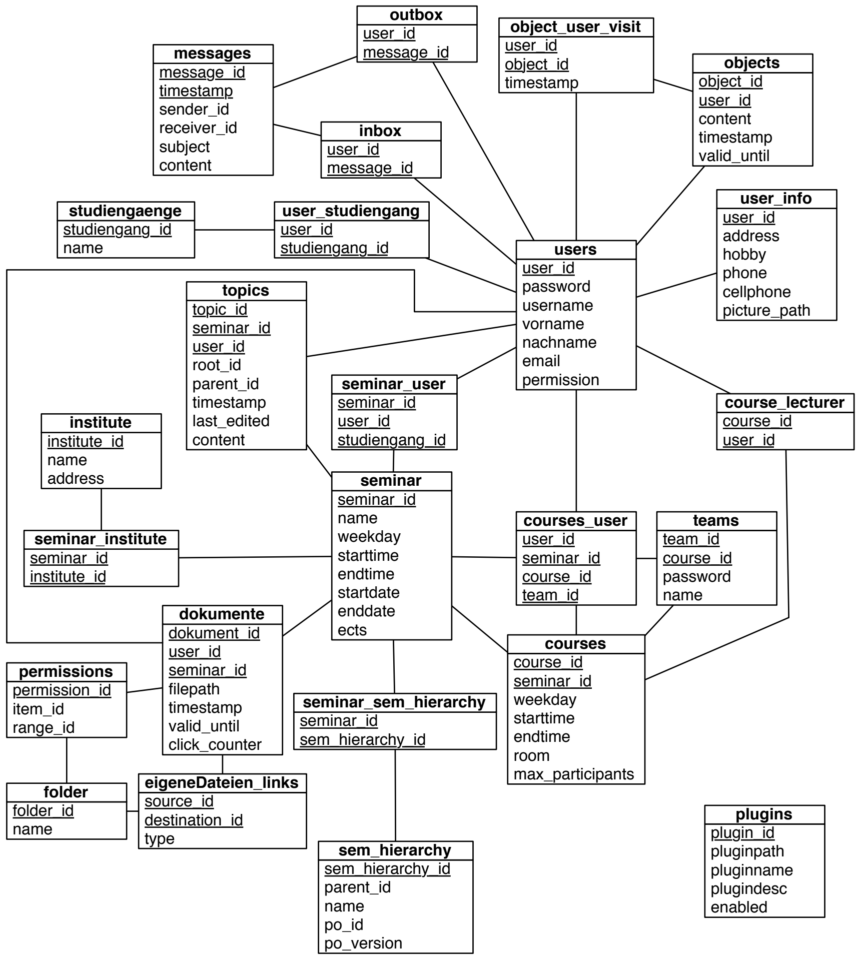

Fig. 4. The database schema of the benchmark.

All objects a user can visit, i.e. a document, a course, etc., are modeled in the table `objects`. The last visits for each user are stored in `object_user_visits`.

The table `user_studiengang` references the users to the table `studiengaenge` in which all degree programs are stored. Therefore, it describes which user is enrolled in which degree program.

`institute` stores all institutions of the university. Each seminar belongs to an institute and this association is stored in `seminar_institute`.

A seminar can be credited for different degree programs, each of which can have different versions of examination rules. These connections are stored in the `sem_hierarchy` table. The seminars are linked to this table via `seminar-_sem_hierarchy` in order to define which seminar can be credited for which degree program and which examination rule.

Additionally, there is a table which stores all information about Stud.IP plug-ins called `plugins`. It contains the path of each plug-in, the name, if it is enabled and an unique id.

3.2 Data Generation

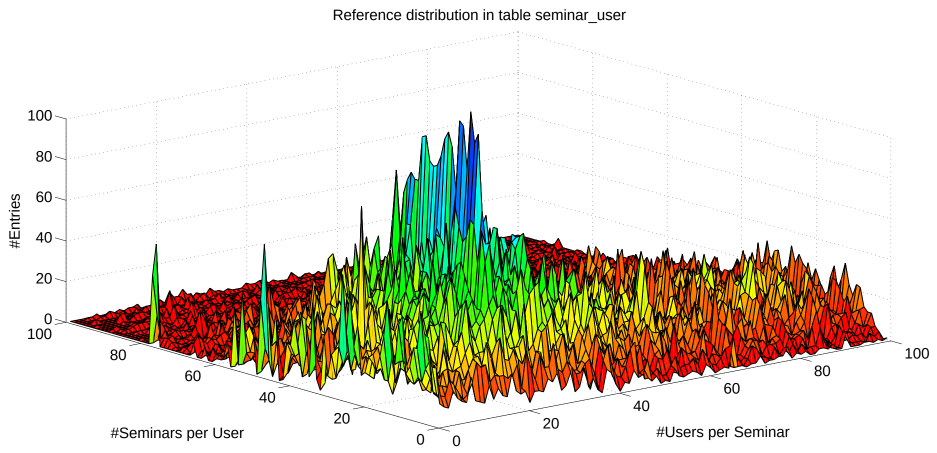

Fig. 5. Distribution of the references in table *seminar_users*.

To populate the schema above a generator for arbitrary sized data was implemented. Different scaling factors can be specified for each table, to enable non-linear scaling. The generated data has similar distribution properties to the original data. To achieve this, we have analyzed the value and reference distributions between the tables. For the table `seminar_users` a reference distribution can be seen in figure 5. As described above this table stores the relationship between users and seminars. We use maximum likelihood estimation to fit standard probability distributions to the data. For now our data generator only supports normal and log-normal distributions, since they model most distributions sufficiently (for a discussion about log-normal distribution see [13]). Figure 6 shows that the distribution of the number of seminars a user is registered for can be modeled by a log-normal distribution, even though a gamma distribution would produce a better fit. The distribution of the number of users per seminar does not match the log-normal distribution very good, but still sufficiently. This can be seen in figure 7. Similar observations about reference distributions were made by Hsu et al. in [8]. They used the Hill equation to model the references, which is related to the log-logistic distribution.

Our data generation differentiates between entity and relationship tables according to the entity-relationship modeling [14]. Entity tables can be generated directly by the given distributions, while relationship tables are generated with knowledge of the according entity tables. The basic entity data generation works

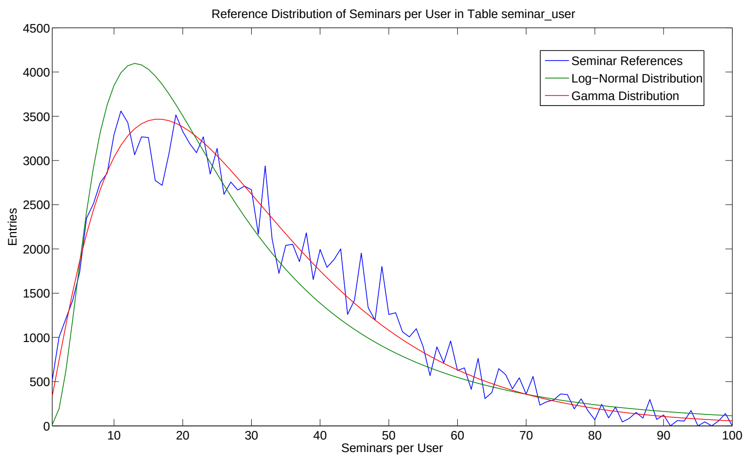

Fig. 6. Distribution of seminars per user in table *seminar_user*.

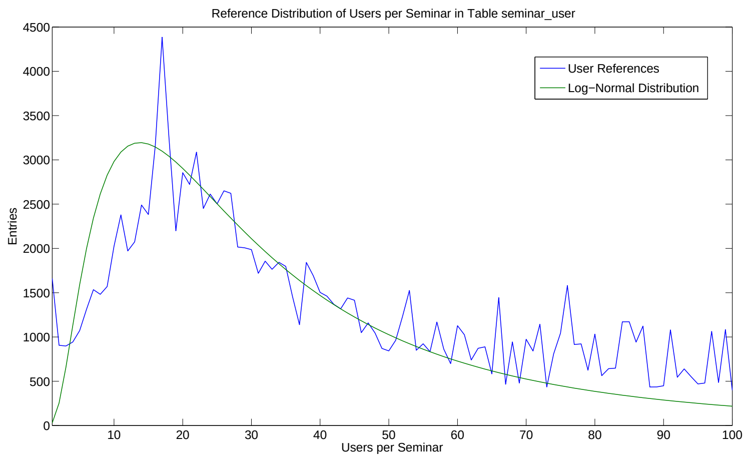

Fig. 7. Distribution of users per seminar in table *seminar_user*.

similar to dbgen or MUDD [15]. For each attribute we specify a domain and a distribution. Whenever possible we use real data from the Stud.IP database or other sources. Table `user` is for example defined as follows.

Each user gets a unique, consecutively numbered id first. The name of a user is generated by selecting a first name and a last name randomly. These names can be listed in a separate configuration file. Additionally, each user gets a unique username, which consists of his last name and a serial number. For his email address, a domain is added to the username. A password is also generated for every user by calculating the MD5 hash of the unique username. For a seminar, the process is similar.

Relationship tables are generated based on the entity tables. Thus for each referenced entity table the references are copied according to the modeled distribution. Additional attributes are generated in the same way as for entity tables.

3.3 Query Set

We extracted a set of 30 common queries from the original system. The queries are different in their characteristics and workload. Yet all queries must be processed within seconds. We changed the query syntax to comply with the SQL 92 standard. In the following we will give some examples of possible queries. The first example selects information about a user. This is usually done at the login for the users start page or if a user homepage is visited:

```
SELECT s.name, u.vorname, u.nachname, ui.address, ui.phone,
    u.email
  FROM users u, user_info ui, user_studiengang us,
      studiengaenge s
 WHERE u.user_id = ui.user_id
   AND us.user_id = u.user_id
   AND us.studiengang_id = s.studiengang_id
   AND u.user_id = $user_id;
```

The next query is executed, if a user browses the seminars he is registered for. This is one of the most common actions in Stud.IP. The query is rather expensive.

```
SELECT seminar.name, courses.weekday, courses.starttime,
    course.endtime, user.vorname, user.nachname
  FROM seminar_user, seminare, object_user_vists,
      seminar_sem_tree
 WHERE seminar_user.seminar_id = seminare.seminar_id
   AND object_user_vists.object_id = seminar_user.seminar_id
   AND object_user_vists.user_id = serminar_user.user_id
   AND seminar_sem_tree.seminar_id = seminar_user.seminar_id
   AND seminar_user.user_id = $user_id
UNION
SELECT seminar.name, courses.weekday, courses.starttime,
    course.endtime, user.vorname, user.nachname
  FROM course_lecturer, courses, seminare, object_user_vists,
      seminar_sem_tree
 WHERE course_lecturer.course_id = courses.course_id
   AND seminare.seminar_id = courses.seminar_id
   AND object_user_vists.object_id = seminar_user.seminar_id
   AND object_user_vists.user_id = serminar_user.user_id
   AND seminar_sem_tree.seminar_id = seminar_user.seminar_id;
   AND course_lecturer.user_id = $user.id;
```

Additionally there are update queries, which are executed whenever an object, i.e. a seminar, a document etc. is visited. As well as inserts, when new courses or users are created. An example is the following query which is executed when a user accesses an object.

```
UPDATE object_user_visits
   SET last_access = NOW()
 WHERE user_id = $user_id
   AND object_id = $object_id;
```

3.4 Query Generation

To benchmark our adaptation techniques we will generate sample query streams that are artificial but reflect a realistic user behavior. For that purpose we propose a new kind of random generator for time series.

We start from the assumption that the essential shape of a time series can be modeled by means of an approximating polynomial. Here, a time series describes the aggregated user behavior over one day, for instance, with values each reflecting the number of accesses in a time interval of 60 minutes. Thus, we have time series with 24 measurements starting at 5 am in the morning, for instance, when the number of accesses is lowest (close to zero). That is, we are given a time series consisting of $N + 1 = 24$ observations y_n at points in time x_n with $n \in \{0, \dots, N\}$. These points are assumed to be equidistant in time. In general, an optimally (in the least-squares sense) approximating polynomial $p_{\boldsymbol{a}}$ of degree K can be represented by a linear combination of $K + 1$ basis polynomials p_k:

$$p_{\boldsymbol{a}}(x) = \sum_{k=0}^{K} a_k p_k(x), \tag{1}$$

with a weight vector $\boldsymbol{a} \in \mathbb{R}^{K+1}$, $\boldsymbol{a} = (a_0, a_1, \dots a_K)^{\mathrm{T}}$, where T denotes the transposition of the parameter vector.

In principle, the basis polynomials $p_k(x)$ ($k \in \{1, \dots K\}$) could be *monomials*. Here, however, we claim that they must have the following properties:

1. They must have different and ascending degrees $0, \dots, K$.
2. The leading coefficient (coefficient of the monomial with the highest degree) of each basis polynomial must be one.
3. Each pair of basis polynomials p_{k_1} and p_{k_2} (with $k_1 \neq k_2$) must be *orthogonal* with respect to the inner product

$$\langle p_{k_1} | p_{k_2} \rangle = \sum_{n=0}^{N} p_{k_1}(x_n) p_{k_2}(x_n). \tag{2}$$

   That is, $\langle p_{k_1} | p_{k_2} \rangle = 0$ for all $k_1 \neq k_2$.

It must be mentioned that the choice of these basis polynomials depends on the points in time when samples are observed. If the observations were made at equidistant points in time, the choice depends only on their number $N + 1$ if we assume—without loss of generality—that the first observation is made at time 0—otherwise we simply shift the time series to this point.

In the context of a representation with orthogonal basis polynomials, the a_k are called *orthogonal expansion coefficients*. Each time series—or the polynomial representing this time series, to be precise—can now be regarded as one point in a particular space (we call it *shape space*) which is spanned by the orthogonal expansion coefficients. Due to the particular representation of the approximating polynomial sketched above, these orthogonal expansion coefficient can be interpreted as optimal (in the least-squares sense) estimators of *average* (a_0), *slope* (a_1), *curve* (a_2), *change of curve* (a_3), etc. of the time series.

The description of appropriate techniques for the determination of orthogonal basis polynomials and the efficient computation of the orthogonal expansion coefficients for a given time series is out of the scope of this article. We refer to our previous work published in [16, 17] which is based on mathematical background outlined in [18, 19].

Assume now, we want to construct a random generator for time series describing the user behavior on Mondays which are working days. Then, a set of sample time series is needed to build this generator (ideally, about 25 or more). The time series are all approximated as described above (e.g., with polynomials of degree 6). In our experiments it turned out that the representations of the sample time series all originating from a particular kind of day (e.g., public holiday, working Friday, etc.) can be regarded as being nearly *normally distributed* in the shape space. More precisely, to model this distribution we need the functional form of a *multivariate Gaussian distribution*

$$\mathcal{N}(\boldsymbol{a}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{(K+1)/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\exp\left\{-\frac{1}{2}(\boldsymbol{a}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{a}-\boldsymbol{\mu})\right\} \tag{3}$$

with a $(K+1)$-dimensional center (or mean) $\boldsymbol{\mu}$ and a $(K+1)\times(K+1)$-dimensional matrix $\boldsymbol{\Sigma}$. To find the model parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ from a sample data set, we assume that the points in the shape space are independent and identically distributed and apply a standard *maximum likelihood* technique (cf., e.g., [20]) to determine their values.

A model for a specific set of time series can then be used as a random generator for time series in the following way:

1. A random number generator parameterized by means of the multivariate Gaussian is used to generate random numbers which are points in the shape space distributed according to Eq. 3.
2. Using the (known) orthogonal basis polynomials, these points can be transformed into the respective polynomials.
3. The polynomials can be evaluated at the desired points in time (e.g., at points corresponding to time intervals of 60 minutes).
4. Random noise can be added, e.g., white noise with a standard deviation corresponding to the average approximation error for the set of sample time series.

Altogether, we obtain an arbitrarily large set of artificial time series which all have an essential shape that is similar to the shapes of the time series contained in the set of (real) samples which has been used to build the random generator. An example of an polynomial approximation for Mondays during the lecture period can be seen in figure 8.

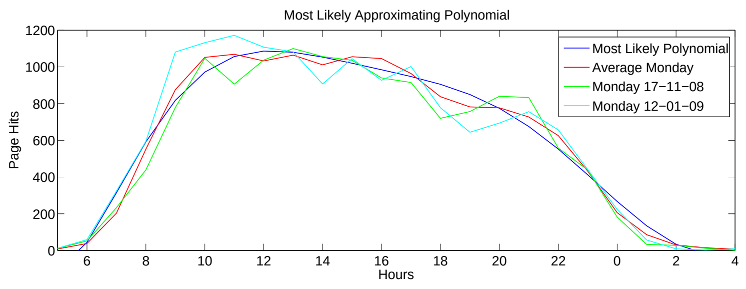

Fig. 8. The most likely approximating polynomial for Mondays during the lecture period.

Each day of the week has different access rates, which can be seen in figure 1. We therefore build single models for every day of the week. This way we can also easily simulate holidays and outliers with anomalous accesses.

3.5 Scaling Time

An important factor for the usability of a benchmark is its runtime [21]. The smallest unit of time that has periodical access rates is usually one day. To test adaptability several periods have to be processed. Since this is too long for most benchmarking purposes, we propose to scale time. With a scaling factor of 1/7 a complete week can be simulated within 24 hours. Depending on the application under test, even smaller factors could be reasonable. An other possibility to shorten runtime is to use a reduced week that only consists of three days.

Of course the system under test should be aware of the time scaling factor. Since daily and weekly periods are usual in information systems, good tuning processes will use this previous knowledge for periodical tasks.

4 Benchmarking Objectives

Depending on the benchmark objective, different test cases can be built. Shifting workloads give lots of opportunities to test automatic and autonomic systems. Usually the metric is *transactions per second* or *average response time* for a given database size, depending on the optimization goal (e.g. the *QphDS@SF* metric in TPC-DS [6]). It has to be mentioned that whichever is used, the other should also be monitored. In the following we will give four examples on how we use the benchmark.

4.1 Basic Performance

The most common benchmarking objective in database systems is to test the speed, i.e. transactions per second or similar. A good baseline for such a test is the peak performance of the system without any automatic tuning and without any workload shifts. We concur with Bruno, who argues that only primary indexes should be used for a reasonable baseline [22].

To test if the system can automatically produce a better throughput in a real time environment, alternating workloads can be used. This way the system has phases of high load, which can be used to measure the peak performance. In phases of low load, the system has time to optimize its table structure, scale itself or tune the indices without risking serious performance bottlenecks. Throughout the test the ratio of different query classes stay constant. After some periods the peak performance should increase and should be better than the baseline performance.

4.2 Adaptivity

As stated in the introduction a major goal was to measure adaptability. The idea is to test how well a system can adjust itself to the workload. Partially this is already tested by the throughput test above. But as we have shown before, the rates of query classes change within a single day. This can be simulated by shifting workloads. So different query sets are defined and for each set a separate time series is generated. Also the workload is different for each day of the week. Either a complete week can be simulated, or only a reduced week consisting only of two working days and one weekend day, which should suffice in most cases. With this test, a system under test that is aware of the temporal dependencies in the workload, should get a better performance than a system that is not.

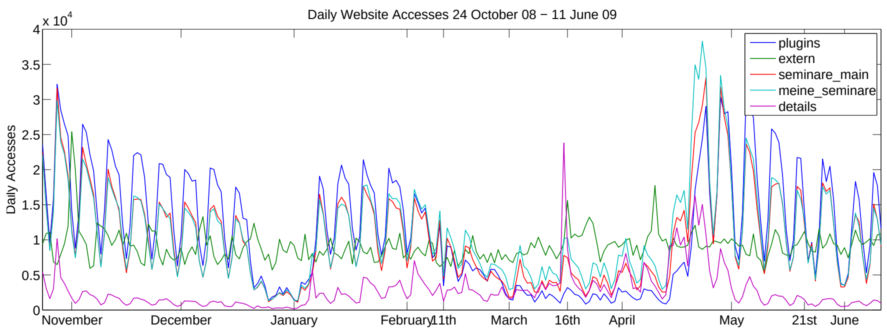

Fig. 9. Most accessed websites in Stud.IP between October 24, 2008 and June 10, 2009 per day.

Changes in the workload behavior can be introduced to further test the adaptivity. In figure 9 the most frequently accessed websites in Stud.IP between October 08 and May 09 can be seen. It is easy to see that there are sections with very different characteristics. The diagram starts shortly after the beginning of the lecture period, which lasts until the first week of February. The next lecture period started at April 20. Additionally the Christmas break from December

24 until January 06 can be seen. So for an eLearning system at a university a week can be classified in one of the three classes, lecture period, free period and holidays. All three of these sections are well-defined and their limits are previous knowledge. This form of test is in some respects already implemented in current benchmarks, TPC-DS for example consists of four consecutive phases with very different characteristics (i.e. load, query run, data maintenance, query run - cf. [23]). However, our form of query generation also makes it possible to model the trends within one phase. Such a trend can be seen in the fall term 2008 where the workload constantly decreases and then slightly increases at the end of the term.

4.3 Robustness

To test the robustness of an autonomic system outliers can be introduced. In figure 9 these can be seen in form of legal holidays on May 21 and June 1 and in form of unpredictable outliers for example on February 11 (server maintenance) or March 16 (unexpected user behavior). An autonomic system should be able to identify such outliers and handle them correctly. So, it should not change its configuration completely based on the single day. Yet it also must not have a serious performance collapse. For legal holidays this could also be supported by previous knowledge. Outliers can be modeled like other days and either triggered randomly (maintenance) or at previously defined points in time. To test robustness the performance before and after an outlier can be compared and the time until the original performance is reached again. To find out if a system is *over adapted*, the performance during an outlier day can be used.

4.4 Energy and Space Efficiency

The shifting workloads can of course be used to test the energy and space efficiency of a system. An autonomic system might be able to reduce its space and energy consumption in phases of low load. To measure the energy efficiency an *transaction per watt* metric, as introduced in [24], can be used.

5 Conclusion

Autonomic tuning is an ongoing field of research in the database community, new evaluation methods are therefore needed. The benchmark introduced in this paper features a new way to model database workloads. With the polynomial representation of every week-day a good compromise between realistic access rates and comparable patterns is found. This opens new possibilities to test automatic and autonomic tuning. The benchmark is based on an online eLearning application that was analyzed extensively.

For future work we will first examine and tune the benchmark. We will improve our eLearning benchmark and analyze how our techniques can be used in other benchmarks as well (e.g. TPC-C, TPC-H). To ease the adaptation of our benchmark we will implement more generic query and data generators. These generators will be controlled by configuration files that make adoption of the schema or value domains and distributions more easy. As online information systems are usually evolving over their life time, an interesting extension will be the introduction of schema evolution. We will include possibilities to alter the current table definitions and add new tables. This will add further challenges to physical design tuning. To learn about realistic schema evolution, we will further monitor the development of the Stud.IP installation at the University of Passau.

6 Acknowledgments

The authors would like to thank Marco Sitzberger and Yang Chen for their help on analyzing the Apache logs and Thiemo Gruber for his help with the implementation of the polynomial approximation.

References

1. Zilio, D.C., Rao, J., Lightstone, S., Lohman, G.M., Storm, A.J., Garcia-Arellano, C., Fadden, S.: Db2 design advisor: Integrated automatic physical database design. In: VLDB '04: Proceedings of the Thirtieth International Conference on Very Large Data Bases, Morgan Kaufmann (2004) 1087–1097
2. Dageville, B., Das, D., Dias, K., Yagoub, K., Zaït, M., Ziauddin, M.: Automatic sql tuning in oracle 10g. In: VDLB '04: Proceedings of the Thirtieth International Conference on Very Large Data Bases, Morgan Kaufmann (2004) 1098–1109
3. Agrawal, S., Chaudhuri, S., Kollár, L., Marathe, A.P., Narasayya, V.R., Syamala, M.: Database tuning advisor for microsoft sql server 2005. In: VDLB '04: Proceedings of the Thirtieth International Conference on Very Large Data Bases, Morgan Kaufmann (2004) 1110–1121
4. Bruno, N., Chaudhuri, S.: An online approach to physical design tuning. In: ICDE '07: Proceedings of the 23rd International Conference on Data Engineering, IEEE (2007) 826–835
5. Wiese, D., Rabinovitch, G., Reichert, M., Arenswald, S.: Autonomic tuning expert: a framework for best-practice oriented autonomic database tuning. In: CASCON '08: Proceedings of the 2008 conference of the center for advanced studies on collaborative research, New York, NY, USA, ACM (2008) 27–41
6. Nambiar, R.O., Poess, M.: The making of tpc-ds. In: VLDB '06: Proceedings of the 32nd international conference on Very large data bases. (2006) 1049–1058
7. Poess, M.: Controlled sql query evolution for decision support benchmarks. In: WSOP '07: Proceedings of the 6th International Workshop on Software and Performance, ACM (2007) 38–41
8. Hsu, W.W., Smith, A.J., Young, H.C.: Characteristics of production database workloads and the tpc benchmarks. IBM Systems Journal **40**(3) (2001) 781–802
9. Agrawal, S., Chu, E., Narasayya, V.: Automatic physical design tuning: Workload as a sequence. In: SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM (2006) 683–694

10. Holze, M., Ritter, N.: Autonomic databases: Detection of workload shifts with n-gram-models. In: ADBIS '08: Proceedings of the 12th East European conference on Advances in Databases and Information Systems. Volume 5207/2008 of Lecture Notes in Computer Science., Berlin / Heidelberg, Germany, Springer-Verlag (2008) 127–142
11. Consens, M.P., Barbosa, D., Teisanu, A.M., Mignet, L.: Goals and benchmarks for autonomic configuration recommenders. In: SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, ACM (2005) 239–250
12. Rabl, T., Pfeffer, M., Kosch, H.: Dynamic allocation in a self-scaling cluster database. Concurrency and Computation: Practice and Experience **20**(17) (2007) 2025–2038
13. Mitzenmacher, M.: A brief history of generative models for power law and lognormal distributions. Internet Mathematics **1**(2) (2004) 226–251
14. Chen, P.P.S.: The entity-relationship model — toward a unified view of data. ACM Transactions on Database Systems **1**(1) (1976) 9–36
15. Stephens, J.M., Poess, M.: Mudd: a multi-dimensional data generator. In: WOSP '04: Proceedings of the 4th international workshop on Software and performance, New York, NY, USA, ACM (2004) 104–109
16. Fuchs, E., Gruber, C., Reitmaier, T., Sick, B.: Processing short-term and long-term information with a combination of polynomial approximation techniques and time-delay neural networks. IEEE Transactions on Neural Networks (2009) (accepted – to appear).
17. Fuchs, E., Gruber, T., Nitschke, J., Sick, B.: On-line motif detection in time series with SwiftMotif. Pattern Recognition **42**(11) (2009) 3015–3031
18. Elhay, S., Golub, G.H., Kautsky, J.: Updating and downdating of orthogonal polynomials with data fitting applications. SIAM Journal on Matrix Analysis and Applications **12**(2) (1991) 327–353
19. Fuchs, E.: On discrete polynomial least-squares approximation in moving time windows. In Gautschi, W., Golub, G., Opfer, G., eds.: Applications and Computation of Orthogonal Polynomials. Volume 131 of International Series of Numerical Mathematics. Birkhäuser, Basel, Switzerland (1999) 93–107 (Proceedings of the Conference at the Mathematical Research Institute Oberwolfach, Germany, March 22-28 1998).
20. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, New York, NY (2006)
21. Blackburn, S.M., McKinley, K.S., Garner, R., Hoffmann, C., Khan, A.M., Bentzur, R., Diwan, A., Feinberg, D., Frampton, D., Guyer, S.Z., Hirzel, M., Hosking, A.L., Jump, M., Lee, H., Moss, J.E.B., Phansalkar, A., Stefanovic, D., VanDrunen, T., von Dincklage, D., Wiedermann, B.: Wake up and smell the coffee: evaluation methodology for the 21st century. Communications of the ACM **51**(8) (2008) 83–89
22. Bruno, N.: A critical look at the tab benchmark for physical design tools. SIGMOD Record **36**(4) (2007) 7–12
23. Poess, M., Nambiar, R.O., Walrath, D.: Why you should run tpc-ds: A workload analysis. In: VLDB '07: Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment (2007) 1138–1149
24. Poess, M., Nambiar, R.O.: Energy cost, the key challenge of today's data centers: A power consumption analysis of tpc-c results. Proceedings of VLDB Endowment **1**(2) (2008) 1229–1240