Guiding Transaction Design through Architecture-Level Performance and Data Consistency Prediction

Philipp Merkle
Software Design and Quality Group
Karlsruhe Institute of Technology (KIT)
76131 Karlsruhe, Germany
merkle@kit.edu

Abstract: Designing transactional software which operates not only in a timely fashion but also preserves data consistency is challenging. While it is easy to preserve data consistency by choosing a high isolation level, this can quickly become a performance bottleneck due to limited concurrency. Conversely, relaxing the isolation between concurrent transactions may lead to data inconsistencies. Solving this trade-off systematically requires quantitative knowledge on the relation between transaction performance and the likelihood of data consistency violations under a given isolation level. Architecture-level performance prediction is a promising approach to address the first half of this trade-off but often neglects the influence of transactions. The second half—data consistency—is not addressed at all by existing approaches. Therefore, we plan to integrate transaction modelling into the Palladio approach for component-based software quality prediction. This creates the opportunity to predict not only performance metrics more accurately, but also to estimate data consistency violations.

1 Introduction

Transactional software is built to support use cases where data inconsistencies are intolerable or acceptable only to a certain extent. Transactions serve as a bundling mechanism for operations to be executed in an atomic, consistent, isolated and durable (ACID) fashion. Transaction isolation (the "I" in ACID) ensures that concurrent transactions do not affect each other concerning the data they access, or that at least the mutual influence is limited to a specified level—the so-called isolation level. For almost any relational DBMS, one can adjust the isolation level: globally for all transactions, per database session or even on a per-transaction basis.

Relaxing isolation is popular for optimising performance. Higher performance, however, comes often at the expense of data inconsistencies. This trade-off becomes apparent especially for the highest level of isolation: serialisability. Under this isolation level, concurrent transactions never interfere on a functional level. The database state resulting from executing two transactions t_1 and t_2 concurrently must have the same effect as executing them in a serial order; i.e. executing first t_1 followed by t_2 or vice versa. This inherently limits concurrency, mostly due to locks on shared data items. Serialisability, however, prevents all kinds of consistency violations due to concurrent data access including the well-known

dirty read and *lost update* anomalies. When choosing a relaxed isolation level, one must be well aware of potential data inconsistencies.

It is up to the software engineer to balance the transaction design between high performance and high consistency. For this, it is essential to understand how design alternatives influence performance and consistency. Design alternatives for transactions include transaction boundaries, database statements issued within these boundaries, the isolation level as well as the database schema.

Architecture-level software quality prediction is a promising approach to obtain quality metrics, on which design decisions can be based upon. For this, we plan to use the Palladio approach for component-based quality prediction [BKR09] as a foundation. Our goal is to integrate transaction modelling into the Palladio approach to obtain transaction quality metrics. These include throughput, abort and retry rates, as well as the degree of data consistency.

The scientific contribution of our research is a model-based approach for transaction quality analysis, integrated with architecture-level quality prediction. The envisioned approach includes a meta-model for transaction modelling, along with a corresponding model solver for predicting transaction quality metrics.

2 Research Goal and Questions

With our research, we intend to support software engineers in designing responsive yet consistency-preserving transactional information systems. We assume a component-based development process (cf. [KH06]) because it is well-suited for quality analyses [KBH07]. In our envisioned approach, component developers specify for each component its transactional behaviour—however, without having to provide each database statement in its full depth. Instead, a suitable abstraction is used. A model solver such as a simulator operates on these specifications to analyse how assembled components in a system influence each other. The solver yields predictions on the overall system performance and on the degree of data consistency.

The sketched approach raises various research questions, which are presented below. These questions mainly focus on performance influences of transactions while a detailed discussion of consistency is left for future work. Each research question comprises a motivation and a suggested solution. More details on the overall approach can be found in Sec. 3.

RQ1: What are the major performance factors of a transaction? For creating a suitable performance abstraction of transactions, it is vital to understand what factors influence a transaction's performance the most, and how they interact. We distinguish between factors stemming from a developer's design space, the deployer's configuration space, and performance factors due to concurrency. Design related factors are transaction boundaries, the complexity of encapsulated database statements and the isolation level under which these statements operate. Configuration related factors stem from the choice be-

tween different database management systems and from rich configuration options thereof. Concurrency factors include lock contention for database objects (e.g. tables, rows or indices) and contention for processing resources such as processors and storage devices. Contention is basically caused either by concurrent transactions or by competing software systems deployed along with the database on the same physical or logical machine.

Not all factors and interactions between them can be studied in the given timeframe, which makes a preselection inevitable. The selection process will favour design related factors since our goals in particular include early feedback on the quality of transaction design. Taking these factors as a starting point, we investigate their interrelation using two transaction processing benchmarks, TPC-W[1] and Apache Day Trader[2]. For automated and systematic experiments, we employ the SPA benchmark harness[3]. Initial experiments show how an increased abort rate due to serialisability leads to a significant drop in throughput.

RQ2: What is an appropriate abstraction level for modelling transactions? The purpose of this research question is to find a good balance between ease of modelling and prediction accuracy. A detailed model is hard to create in early development stages since many details are not known yet; it may, however, yield accurate predictions. By contrast, a highly abstract model is rather easy to create, e.g. from previous experience, but likely reduces prediction accuracy due to neglected factors. An example of a highly abstract model is the mere number of database statements issued by a single transaction; read versus write access is neglected as is the statements' complexity. An example for a detailed model is a full-fledged logical database schema along with a statistical data distribution per table column.

For specifying transaction boundaries and the isolation level, there is not much space for abstraction and we do not see a need to do so. Finding an appropriate abstraction for database statements is more challenging. We believe that probabilistic modelling may be well suited. For each database statement, the software engineer would characterise a number of parameters (e.g. read versus write access and read selectivity) using random variables. For example, one could specify that the customers table is read in 80% of the cases and written in the remaining cases; the selectivity of a read access could be set to 10%. An accompanying schema specification characterises the customers table in terms of its size. On this basis, a model solver could reason on the probability for a read-write conflict under a given concurrency level.

RQ3: How can transaction modelling be integrated with architecture description languages? As has been motivated before, we plan to base our approach on the existing Palladio approach. The Palladio approach includes the Palladio component model (PCM)—a meta-model for component-based software architectures—along with various analytical and simulative solvers capable of predicting the performance for instances of the PCM. Much research effort has been spent to make the PCM a conceptually clean language for

[1] http://www.tpc.org/tpcw/
[2] https://cwiki.apache.org/GMOxDOC20/daytrader.html
[3] http://sdqweb.ipd.kit.edu/wiki/Storage_Performance_Analyzer

architectural descriptions while being at the same time suited for different quality analyses. Preserving these properties throughout the process of integrating transaction modelling is challenging and is addressed by this research question.

Integrating transaction modelling means to extend the expressiveness of the PCM by introducing additional metaclasses on the metamodel level. For transaction demarcation, one could think of three additional actions BeginTX, CommitTX and AbortTX integrated into PCM's control flow abstraction. In between, a DatabaseStatement might issue a database demand similar to plain resource demands in the PCM. The database statement will likely be aligned with SQL statements but is in no case a full-fledged SQL statement.

RQ4: How can transaction-oriented quality prediction and architecture-level quality prediction be combined? Several analysis approaches exist for performance prediction based on a PCM model. Analytical approaches include layered queuing networks and queuing Petri nets. Simulative approaches include discrete-event simulators pursuing a generative [BKR09] or interpretive [MH11] approach. These solvers differ in expressiveness imposed by the underlying formalism, but also in maturity and coverage—some support just a subset of the PCM meta-model.

One of the main arguments for analytical approaches is their speed, which usually outperforms simulative approaches by magnitudes; however, at the cost of limited predictive power due to what is called the state space explosion problem. If one of the analytic approaches proves to be sufficiently powerful, we will stick to it for predicting transaction quality metrics. Otherwise, discrete-event simulation will serve this purpose.

3 Envisioned Approach

The artefacts involved in our approach and their interrelation are depicted in Fig. 1. The central component is the *transaction simulator*, which imitates a DBMS's transaction manager. Supplied with a transaction model and a DBMS profile, the transaction simulator predicts transaction quality metrics, including throughput, abort rates and the likelihood for various consistency violations.

The *transaction model* contains abstract behaviour characterisations for each transaction to be simulated. An example characterisation is the probability to access a certain database table, along with the total number of database accesses issued by that transaction. The transaction model references elements from the architecture model to allow for assigning transactional behaviour to certain architectural entities like a component specification.

The *DBMS profile* takes into account that transaction managers (TM) behave different, leading to different implications on performance and consistency. A locking-based TM, for instance, differs fundamentally from a TM based on multi-version concurrency control (MVCC); while the former acquires read locks in most isolation levels, the latter avoids read locks entirely. Even in the class of locking-based TMs, there are wide differences in the granularity of locks. While one TM features a sophisticated lock hierarchy, another

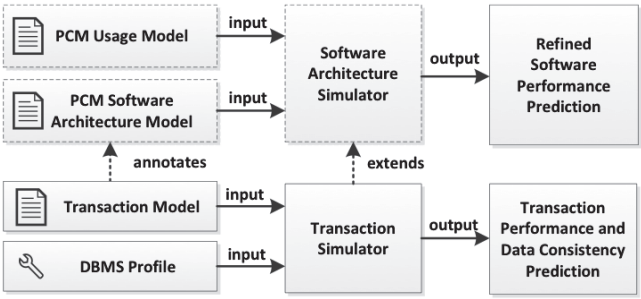

Figure 1: Artefacts in our approach and their interrelation. Dashed boxes indicate existing artefacts.

TM operates with fine-grained row-level locks only. This is why the transaction simulator is parametrised by a DBMS profile, which captures the characteristics of a DBMS. This way, the DBMS profile factors out product specifics from the transaction simulator.

To obtain system-level predictions, we integrate the transaction simulator into an existing but slightly modified *software architecture simulator*. For this, we use one of the simulators developed in the scope of the Palladio approach, e.g. EventSim [MH11], which has been specifically developed for extensibility. The architecture simulator expects two inputs: a PCM software architecture model, whose expressiveness is enhanced by the transaction model, and a corresponding PCM usage model. The *software architecture model* specifies the components and how they are assembled and deployed; the *usage model* specifies common use cases (called usage profiles), i.e. the workload. Whenever the architecture simulator encounters a transaction, it delegates the responsibility to the transaction simulator. This way, the usage profile propagates through the architecture down to transactions, where the transaction simulator keeps track of transaction-induced contention such as lock contention.

4 Related Work

A general overview of software performance evaluation (SPE) is provided in [BDMIS04]. SPE for component-based systems is surveyed in [Koz10]. From [Koz10], it becomes apparent that performance prediction approaches with a focus on middleware (including transactions) often neglect the influence of business logic and vice versa. A notable exception is the work by Menascé and Gomaa [MG00], who predict performance of a transaction-intensive client/server system. Using their CLISSPE language, they create detailed models of transaction behaviour along with information on the database schema and the DBMS. The high level of detail, however, makes it difficult to use their approach in early development stages. Data consistency prediction is covered by a few publications, e.g. [FGA09]. However, we do not know of approaches aimed at predicting data consistency on the system level.

5 Conclusion

In this paper, we argued for representing transactional behaviour explicitly in approaches for software quality prediction. Such an integrated prediction creates the opportunity to predict not only performance metrics more accurately, but also provides an estimate of data consistency violations. These metrics are supposed to help software engineers in finding a suitable balance between performance and data consistency. We base our approach on the existing Palladio approach for component-based software quality prediction, which has been successfully applied in a number of case studies.

References

[BDMIS04] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: a survey. *Software Engineering, IEEE Transactions on*, 30(5):295 – 310, may 2004.

[BKR09] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82:3–22, 2009.

[FGA09] Alan Fekete, Shirley N. Goldrei, and Jorge Pérez Asenjo. Quantifying isolation anomalies. *Proceedings of the VLDB Endowment*, 2(1):467–478, 2009.

[KBH07] Heiko Koziolek, Steffen Becker, and Jens Happe. Predicting the Performance of Component-based Software Architectures with different Usage Profiles. In *Proc. 3rd International Conference on the Quality of Software Architectures (QoSA'07)*, volume 4880 of *Lecture Notes in Computer Science*, pages 145–163. Springer-Verlag Berlin Heidelberg, July 2007.

[KH06] Heiko Koziolek and Jens Happe. A QoS Driven Development Process Model for Component-Based Software Systems. In *Proc. 9th Int. Symposium on Component-Based Software Engineering (CBSE'06)*, volume 4063 of *Lecture Notes in Computer Science*, pages 336–343. Springer-Verlag Berlin Heidelberg, 2006.

[Koz10] Heiko Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):634–658, 2010. Special Issue on Software and Performance.

[MG00] Daniel A. Menascé and Hassan Gomaa. A method for design and performance modeling of client/server systems. *Software Engineering, IEEE Transactions on*, 26(11):1066–1085, 2000.

[MH11] Philipp Merkle and Jörg Henss. EventSim – An Event-driven Palladio Software Architecture Simulator. In *Proc. Palladio Days 2011*, Karlsruhe Reports in Informatics ; 2011,32, pages 15–22, Karlsruhe, 2011. KIT, Fakultät für Informatik.