Analytical Considerations for Transactional Cache Protocols

Daniel Pfeifer
Institute for Program Structures and Data Organisation (IPD)
Universität Karlsruhe, Germany
pfeifer@ira.uka.de
April 19, 2005

Since the early nineties transactional cache protocols have been intensively studied in the context of client-server database systems. Research has developed a variety of protocols and compared different aspects of their quality using *simulation systems* and applying semi-standardized benchmarks. Unfortunately none of the related publications substantiated their experimental findings by thorough analytical considerations. We try to close this gap at least partially by presenting comprensive and highly accurate analytical formulas for quality aspects of two important transactional cache protocols.

We consider the non-adaptive variants of the "Callback Read Protocol" (CBR) and the "Optimistic Concurrency Control Protocol" (OCC). The paper studies their cache filling size and the number of messages they produce for the so-called *UNIFORM* workload. In many cases the cache filling size may considerably differ from a given maximum cache size – a phenomenon which has been overlooked by former publications. Moreover for OCC, we also give a highly accurate formula which forecasts the transaction abortion rate. All formulas are compared against corresponding simulation results in order to validate their correctness.

Categories and Subject Descriptors: H.2.4.o [**Information Systems**]: Database Management—*Systems, Transaction Processing*; H.3.4.b [**Information Systems**]: Information Storage and Retrieval—*System and Software, Distributed Systems*; C.4 [**Performance of Systems**]: Optimization

General Terms: Caching, Client-Server, Protocol, Transaction Management

Additional Key Words and Phrases: Page-Server, Performance, Analysis

1. INTRODUCTION

In the early nineties database researchers have started to study client-server database systems. At this type of systems, clients may access a central database system over the network in order to perform transactions. For read and write access a client downloads a fixed unit of data from the server database and stores it in a local cache. Depending on the type of database system the data units may be pages or objects (or both). For simplicity this paper considers only pages as data units. Figure 1 illustrates the basic architecture of a client-server database system. A client may access serveral pages within the same transaction. At transaction commit, all client changes must eventually be propagated to the server and must be written to the server's database.

It is widely accepted that client-server database systems should meet three important requirements:

—Client transactions should be serializable at the database server. (Serializability is a well-known consistency requirement for database systems (see [Bernstein et al. 1987]).)

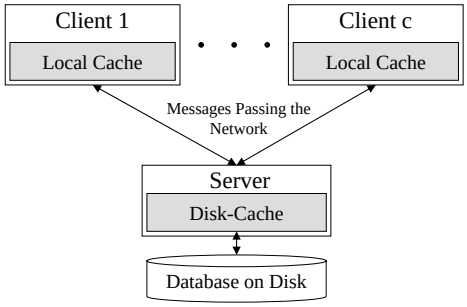

Fig. 1: Basic Architecture of a Client-Server Database System

—The so-called external consistency ensures that a valid serialization order of committing transactions is the same as the commit order at the server ([Adya et al. 1995]).[1]

—Execution of client requests should be as efficient as possible. This includes high transaction throughput, low transaction latency and a low probability of transaction aborts.

Obviously, the protocol for processing client operations has a crucial impact on these requirements. Any such protocol which caches downloaded pages at the client side and which meets the first two of the above requirements is called a *transactional cache protocol*. *In the following we refer to it more shortly as a protocol*.

One of the goals of various research contributions in the field of client-server database systems was to invent highly efficient protocols or to compare existing protocols with respect to their efficiency. Since many parts of a client-server database system are already fixed by its core architecture and by hardware parameters, the essential algorithm behind a protocol can only optimize a few efficiency-related parameters. These include

—the number and the size of messages transferred between client and server,

—the degree to which message processing happens synchronously or asynchronously, respectively and

—the probability of transaction aborts.

Traditionally, the quality of protocols is evaluated by simulation models and so, the related results are experimental in nature. There is only little work on predicting quality aspects of protocols analytically. Unfortunately there are drawbacks in considering simulation results *only*:

—Detecting implementation errors and false results is difficult. Analytical solutions can help to substantiate experimental findings and vice versa.

—Experimental results do not give plausible explanations *why* a protocol behaves in a certain way. Conversely, analytical solutions are usually *based* on plausible explanations and make them explicit.

—Unlike simulation results, analytical findings can help to optimize system parameters instantly at system runtime.

In this paper we analyze quality parameters of two important and widespread protocols, namely the ”Callback Read Protocol” (CBR, [Wang and Rowe 1991]) and the ”Optimistic

[1] In many cases it is still acceptable if the serializiation order and commit order deviate from each other within certain bounds.

Concurrency Control Protocol" (OCC, [Adya et al. 1995; Gruber 1997]). We assume fixed transaction lengths and use the so-called *UNIFORM* workload. This workload assumes that a random transaction operation accesses every database page with the same probability. Moreover, the probabilities for read and write access of database pages are constant (but may differ from each other).

Given this framework, we derive highly accurate formulas to predict the following quality parameters of CBR and OCC:

—the average number of messages exchanged between a client and the server with respect to a committing transaction and

—the average filling size of a client cache after (infinitively) many operations. Note that a client cache does not always fill up to its maximum size since page roll-ins and removals may compete at the cache. This phenomenon has not been studied in any former publications.

In order to verify our analytical solutions, we constructed a client-server database simulation system according to [Franklin et al. 1997]. As it will be shown, there is a strong correspondence between the experimental and the analytical results and so, the related results give mutual evidence for their correctness.

We only consider the *non-adaptive* versions of CBR and OCC. The adaptive versions of these protocols, namely ACBL and AOCC, may dynamically reduce the units of data for operations from pages to objects in order to reduce the chance of conflicts ([Zaharioudakis et al. 1997; Gruber 1997]).

Adaptivity does not affect the number of messages per committing transaction and so the respective formulas from below also hold for ACBL and AOCC. However, our analysis of cache filling sizes does not apply to ACBL and AOCC.

Additionally, we give a highly accurate formula to predict transaction abort rates when applying OCC. Note that there is a body work in estimating transaction abort rates of optimistic transaction protocols (see [Thomasian 1998]). Unfortunately the related solutions are either mathematically complex or too inaccurate. In contrast, we improve a simple and well-known formula for estimating abort rates of optimistic protocols such that it produces highly accurate results for OCC.

The rest of this paper is structured as follows: Section 2 references important publications in the field of transactional caching and discusses how protocols have been analyzed by former publications. Section 3 analytically derives the expected cache filling size for CBR as well as OCC and verifies the solutions' correctness using simulation. Section 4 and Section 5 focus on the number of messages and OCC's transaction abort rate, respectively. The paper closes with a brief conclusion.

2. RELATED WORK

Currently, [Franklin et al. 1997] is one of the most authoritative contributions in the field of client-server database systems. The authors roughly compare about nine different protocols and classify them according to a taxonomy. Moreover they compare some of the protocol's efficiency aspects using a simulation system.

The simulation system is based on a closed queuing simulation model for client-server database systems which was first introduced by [Wilkinson and Neimat 1990]. The core concept of this simulation model was adopted by many papers in the field of transactional cache protocols including [Wu et al. 2004; Chu and Winslett 1994; Chung et al. 1997;

Franklin et al. 1997; Gruber 1997; Adya et al. 1995; Özsu et al. 1998; Zaharioudakis et al. 1997; Wang and Rowe 1991; Franklin and Carey 1992; Carey et al. 1991; Carey et al. 1994]. Therefore, it represents a quasi-standard for comparing corresponding protocols. Concerning the simulation model itself, the main difference across these publications are the are architectural tuning parameters, e.g. network latency, processor speed, disk speed or the number of instructions to perform protocol related actions.

There has also emerged a quasi-standard for the type of transaction workloads which are used in the context of client-server database simulation systems. The most important ones are *UNIFORM*, *HOTCOLD*, *PRIVATE*, *HICON* and *FEED* and they were introduced by [Franklin and Carey 1992; Carey et al. 1991]. [Wu et al. 2004; Carey et al. 1994; Zaharioudakis et al. 1997; Gruber 1997; Özsu et al. 1998; Franklin et al. 1997; Adya et al. 1995] use all or subsets of these workloads in order to study the quality of protocols. Every workload tries to reflect more or less typical access patterns of potential client applications or puts focus on certain quality aspects. In this paper we limit ourselves to the *UNIFORM* workload because it well exhibits the quality aspects we want to study. Also, due to its simplicity, it is well suited for analytical considerations.

The protocols that we regard in this paper, namely CBR and OCC, have been intensively studied using simulation ([Franklin et al. 1997; Adya et al. 1995; Gruber 1997]). We refer to [Franklin et al. 1997] and [Adya et al. 1995; Gruber 1997] for a detailed description of CBR and OCC as well as their derivatives ACBL and AOCC. CBR is widely accepted as the leading algorithm because it combines good transaction throughput with low abort rates ([Wu et al. 2004; Adya et al. 1995]). OCC and its derivative AOCC provide very good transaction throughput but for certain workloads, they can lead to high abort rates ([Wu et al. 2004; Adya et al. 1995; Gruber 1997]).

To the best of our knowledge there are no publications which present conclusive and accurate analytical models for quality parameters of protocols. [Gruber 1997] discusses a basic model which explains why the number of messages per transaction differ between OCC and CBR. However, the model is far from enabling the prediction quantitative results. [Wu et al. 2004] introduce ADCC – a peer-to-peer-based protocol. The authors analytically compare the message cost of CBR and ADCC for the case of write/write conflicts. Neither does this analysis predict the total message cost per committing transaction nor is it validated by experiments.

[Franklin et al. 1997] depicts the cache hit rate for simulation results which are based on the *UNIFORM* workload. The related chart displays the effect of a cache filling size which is below the maximum cache size but the phenomenon is not discussed any further.

3. CACHE SIZE

For many workloads, a client's cache filling size can essentially differ from the cache's maximum cache size even after many operations. Assuming the *UNIFORM* workload, this section presents formulas to predict this effect for CBR and OCC. To verify their accuracy, the formulas' results are compared with corresponding simulation results.

3.1 Analysis

Let c be the number of clients running operations of concurrent transactions in random order. Let n be the total number of operations that have been processed across all clients since system start. Let p_r be the probability that an operation performs a read access and p_c be the chance that an operation is part of a committing transaction.

We consider a certain client C and its cache. Let $s(n)$ be C's current cache filling size after n operations and $s_{\max}$ be its maximum cache size. Let D be the total number of database pages at the server. Further, let $p_{hit}(s)$ be the cache hit rate at client C which depends on the caches filling size s.

Since we assume a *UNIFORM* workload, we can determine the probability of a cache hit for the $n+1$-th operation occurring at client C as follows:

$$p_{hit}(s(n)) = s(n)/D. \tag{1}$$

When considering all concurrent requests of all clients, there is fraction of requests that increases the cache size at C and another fraction that decreases the cache size. Given the cache is not yet full, an increasing cache size is caused by every page miss at client C. On the other hand, a page invalidation (at OCC) or a page callback (at CBR) reduce the cache size by one. The corresponding actions occur in random order with specific probabilities and cause an asymptotic cache filling size s_{exp} after infinitively many calls. Obviously, s_{exp} ranges in $[0, s_{max}]$.

To determine s_{exp}, we proceed as follows: At first, we derive an equation describing how the expected cache filling size $s(n+1)$ (for operation $n+1$) depends on $s(n)$. The equation ignores the minimum cache size 0 and the maximum cache size s_{max}. Then, we consider the related asymptotic solution $\lim_{n\to\infty} s(n)$ which gives a stable fixed point s_{fix}. As the final solution we obtain $s_{exp} = \max(0, \min(s_{fix}, s_{\max}))$.

In the following we discuss the equation for $s(n+1)$ and the related fixed point s_{fix} for OCC as well as CBR.

3.1.1 *OCC.* For OCC, the equation for $s(n+1)$ must reflect the chance of rolling in as well as invalidating pages. The following non-trivial facts affect formula's structure:

—The chance that operation $n+1$ is generated by client C is $1/c$.

—Pages which are rolled in and written by an aborting transaction do not affect $s(n+1)$ since OCC will remove the page again at the transaction's abort time.

—If an aborting transaction C writes a page which is already cached, the page will eventually be invalidated an so the operation reduces the cache size by one.

—Another client's operation which is part of committing transaction and which writes a page being in C's cache, causes an invalidation and reduces C's cache size by one.

This leads to

$$\underbrace{s(n+1)}_{\text{New Cache Size}} = \underbrace{s(n)}_{\text{Old Cache Size}} + \underbrace{1/c}_{\text{Local Operation}} \cdot \underbrace{(1-p_{hit}(s(n)))}_{\text{Cache Miss}} \cdot \underbrace{(p_c + (1-p_c)\cdot p_r)}_{\text{Commits or Aborts and Reads}}$$

$$- \underbrace{1/c}_{\text{Local Operation}} \cdot \underbrace{p_{hit}(s(n))}_{\text{Cache Hit}} \cdot \underbrace{(1-p_r)}_{\text{Writing}} \cdot \underbrace{(1-p_c)}_{\text{In Aborting Transaction}}$$

$$- \underbrace{(1-1/c)}_{\text{Other Client's Operation}} \cdot \underbrace{p_{hit}(s(n))}_{\text{Cache Hit at Client } C} \cdot \underbrace{(1-p_r)}_{\text{Writing}} \cdot \underbrace{p_c}_{\text{In Committing Transaction}}.$$

Using equation 1 we can substitute $s(n)/D$ for $p_{hit}(s(n))$ and try to find a fixed point for $\lim_{n\to\infty} s(n)$ by setting $s_{fix}^{OCC} = s(n+1) = s(n)$. Solving the resulting equation we obtain:

$$s_{fix}^{OCC} = D \cdot (p_c + (1-p_c)\cdot p_r)/(1 + p_c \cdot (c-1)\cdot (1-p_r))).$$

The fixed point is globally stable because

$$|\partial s(n+1)/\partial s(n)| = |1 - 1/(c \cdot D) - (1 - 1/c) \cdot (1 - p_r) \cdot p_c/D| < 1.$$

Eventually we obtain the asymptotic cache filling size $s_{exp}^{OCC} = \max(0, \min(s_{fix}^{OCC}, s_{\max}))$.

3.1.2 *CBR.* The equation for $s(n+1)$ is similar to the one for OCC. However, another client's write operation may reduce C's cache size by one due to a callback (and not due to an invalidation as with OCC). The other client's operation does not have to be part of a committing transaction. This gives:

$$\underbrace{s(n+1)}_{\text{New Cache Size}} = \underbrace{s(n)}_{\text{Old Cache Size}} + \underbrace{1/c}_{\text{Local Operation}} \cdot \underbrace{(1 - p_{hit}(s(n)))}_{\text{Cache Miss}} \cdot \underbrace{(p_c + (1 - p_c) \cdot p_r)}_{\text{Commits or Aborts and Reads}}$$

$$- \underbrace{1/c}_{\text{Local Operation}} \cdot \underbrace{p_{hit}(s(n))}_{\text{Cache Hit}} \cdot \underbrace{(1 - p_r)}_{\text{Writing}} \cdot \underbrace{(1 - p_c)}_{\text{In Aborting Transaction}}$$

$$- \underbrace{(1 - 1/c)}_{\text{Other Client's Operation}} \cdot \underbrace{p_{hit}(s(n))}_{\text{Cache Hit at Client } C} \cdot \underbrace{(1 - p_r).}_{\text{Writing}}$$

The related fixed point is

$$s_{fix}^{CBR} = D \cdot (p_c + (1 - p_c) \cdot p_r)/(1 + (c - 1) \cdot (1 - p_r)))$$

and analogously to the OCC-case, it is globally stable. The asymptotic cache filling size is $s_{exp}^{CBR} = \max(0, \min(s_{fix}^{CBR}, s_{\max}))$.

3.2 Evaluation

For the verification of all analytical findings of this paper, we implemented a simulation system according to [Franklin et al. 1997] and tried to stick as closely as possible to the specification such as given by the paper. This includes the hardware, software, database and workload parameters.[2] Only the following parameters differ from the corresponding ones in [Franklin et al. 1997]:

—In this report, the number of concurrent clients varies between 1 and 40, whereas in [Franklin et al. 1997] it only varies between 1 and 25.

—Here, the maximum size $s_{\max}$ of a client cache can be 15%, 25% or 35% of the server database size, whereas [Franklin et al. 1997] uses 5% and 25% instead.

To compute an analytical value that corresponds to a simulation result, we use the corresponding base parameters from [Franklin et al. 1997]. E.g. the write probability 0.2 for the *UNIFORM* workload corresponds to $p_r = 0.8$ in the above equations. Similarly, the parameter "DatabaseSize" from [Franklin et al. 1997] corresponds to D from above. The only parameter in the above formulas which cannot derived from the specification of [Franklin et al. 1997] is the probability p_c. To determine it, we used the following formula and corresponding values as obtained from simulation runs:

$$p_c = ops \cdot \text{"Committed Transactions Per Run"}/\text{"Total Operations Per Run"}.$$

[2]The related implementation is written in Java and is publicly available ([Pfeifer 2005]). It is well suited as a framework for implementing further transactional cache protocols.

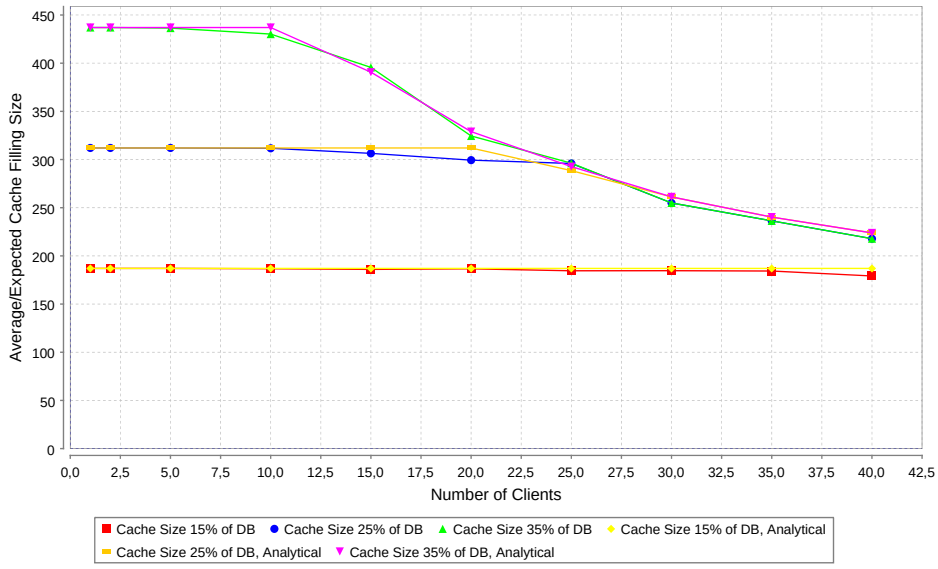

Fig. 2: Average/Expected Cache Filling Size for OCC Using Simulation and Analysis

Figure 2 compares the average cache filling size using simulation versus the expected cache filling size using analysis for OCC. At simulation, the average is computed from all sizes measured at client caches after an appropriate warmup period. The warmup period ensures that the cache filling sizes become stable since the caches are empty at simulation start.[3] The good fit between the graphs is evidence of the correctness of the OCC-related formula from above.

Figure 3 compares the average cache filling size using simulation versus the expected cache filling size using analysis for CBR. Similar to Figure 2, the graphs substantiate the correctness of the CBR-related formula from above.

The simulation confirms that under load, a client cache does not necessarily fill up to its maximum. In general, it may be important to consider this phenomenon when designing, configuring and optimizing client-server database systems because it negatively affects the cache hit rate and may result in unused and thus wasted client cache storage. To illustrate this, Figure 4 shows the cache hit rate for OCC (after the warmup period) with respect to the experiments from Figure 2. (The values are essentially proportional to the ones from Figure 2.)

4. MESSAGES FOR COMMITTING TRANSACTIONS

This section estimates the number of messages exchanged between client and server with respect to committing transactions.

4.1 Analysis

4.1.1 *OCC.* For OCC, estimating the number of messages m^{OCC} for a committing transactions is straight forward. m^{OCC} only depends on the number of operations ops per

[3]We varied the warmup period in tests prior to the actual simulation to guarantee reliable and representative results.

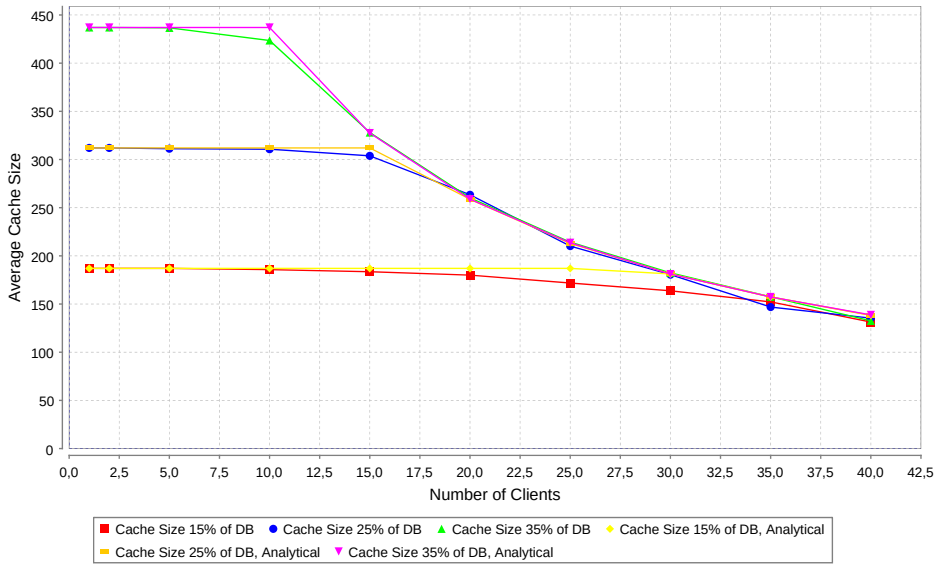

Fig. 3: Average Cache Filling Size for CBR Using Simulation and Analysis

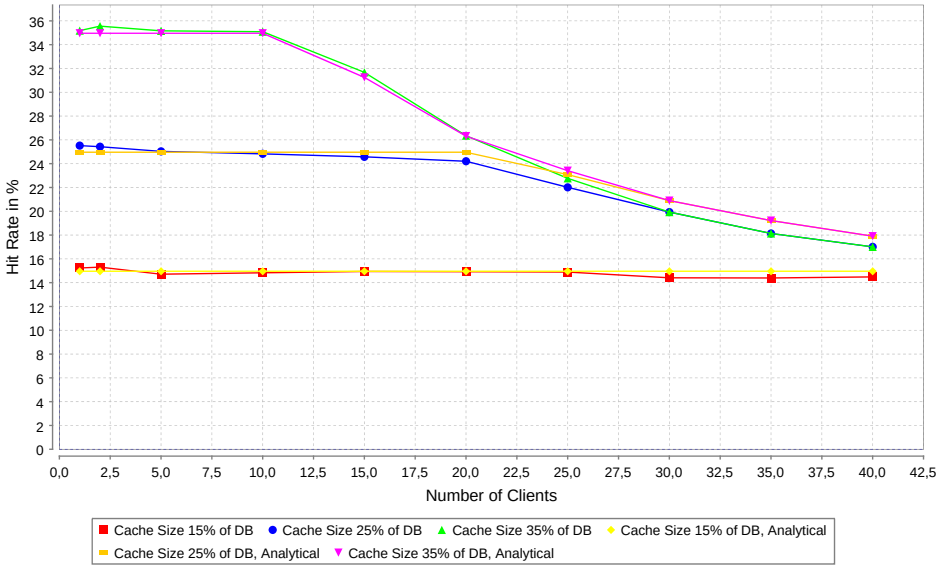

Fig. 4: Cache Hit Rate for OCC Using Simulation and Analysis

transaction and the expected cache hit rate $p_{hit}(s_{exp}^{OCC})$:

$$m^{OCC} = \underbrace{2 \cdot ops \cdot (1 - p_{hit}(s_{exp}^{OCC}))}_{\text{2 Messages for Every Cache Miss (Back and Forth)}} + \underbrace{2.}_{\text{Commit Operation}}$$

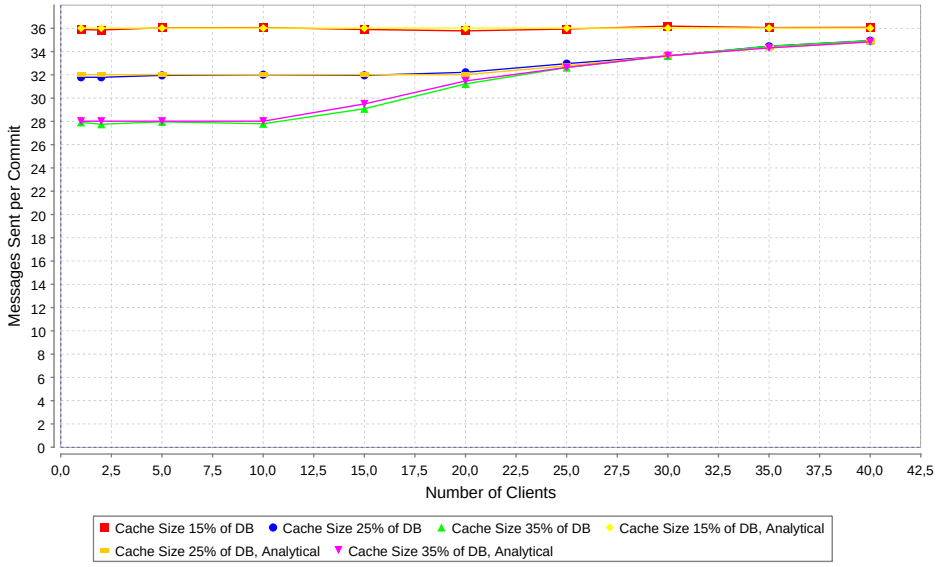

Fig. 5: Average Number of Messages per Committing Transaction for OCC Using Simulation and Analysis

4.1.2 *CBR.* For CBR, one can estimate the number of messages m_{CBR} as follows:

$$m^{CBR} = 2 \cdot ops \cdot \Big(\underbrace{(1 - p_{hit}(s_{exp}^{CBR}))}_{\text{Cache Miss}} + \underbrace{p_{hit}(s_{exp}^{CBR}) \cdot (1 - p_r)}_{\text{Cache Hit but Writing}}$$

$$+ \underbrace{(1 - p_r) \cdot p_{hit}(s_{exp}^{CBR}) \cdot (c - 1)}_{\text{Callbacks (Write Ops. with Cache Hits at Other Clients)}} \Big) + \underbrace{2.}_{\text{Commit Operation}}$$

At CBR, a client C needs to acquire a write lock from the server for almost every write operation (and this results in two additional messages). The case where the client already possesses the write lock is ignored because it is rare: The client must have written the respective page before but in the same transaction. This event is unlikely since $ops \ll D$. About $p_{hit}(s_{exp}^{CBR}) \cdot (c-1)$ clients cache the page that C is about write, which result in the above state number of callback messages.

4.2 Evaluation

Figure 7 and 6 compare simulation results with analytical results for the average/expected number of messages per committing transactions using OCC and CBR. The parameters for the simulation runs are the same as in Section 3.2. For clarity, at CBR, we omitted the graphs where the maximum cache size is 25% of the database size. (Still, the corresponding results are similar.)

Again, there is a good fit between the respective graphs which is evidence of the correctness of formulas from above.

5. TRANSACTION ABORTION RATE FOR OCC

This section improves a well-known formula from [Thomasian 1998], to get an accurate prediction of transaction abortion rates when using OCC.

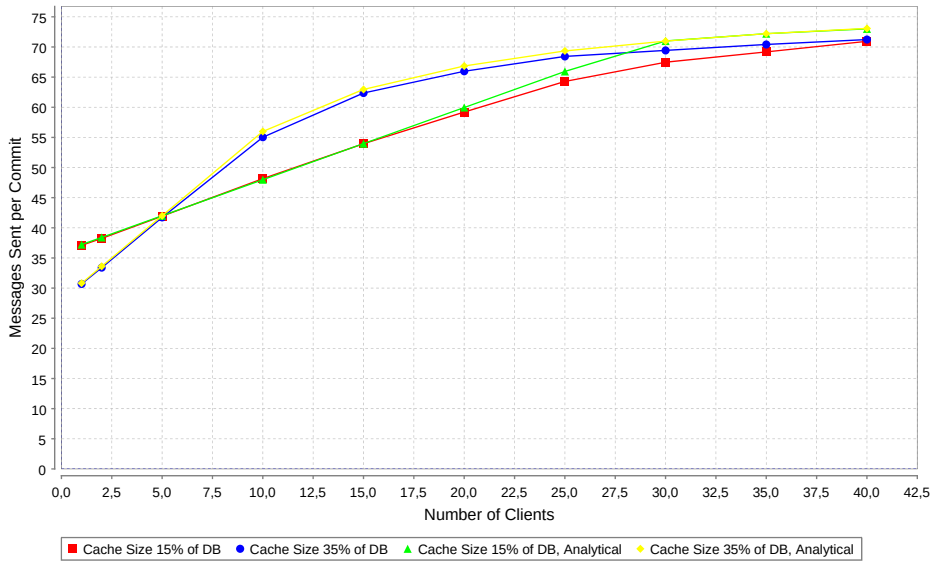

Fig. 6: Average Number of Messages per Committing Transaction for CBR Using Simulation and Analysis

5.1 Analysis

Let T be a transaction running at client C. We want to determine the probability p_{abort} that T aborts.

Let us assume that T will execute all of its ops operations. While T is running, there are about $2 \cdot (c-1)$ concurrent transactions from other clients which overlap with T. About half of them commit before T and they can cause T to be aborted. The latter transactions execute about half of their operations while T is running. Considering these facts and applying the formula from Section 5.2 of [Thomasian 1998], one obtains the following (naive) solution:

$$p_{abort\ naive} = 1 - (1 - ops/D)^{1/2 \cdot ops \cdot (c-1) \cdot (1-p_r)}.$$

The formula ignores the fact that only *committing* transactions can cause the abortion of T and therefore the factor $1 - p_{abort}$ is missing in the exponent. An improved solution is given by the following equation:

$$p_{abort} = 1 - (1 - ops/D)^{1/2 \cdot ops \cdot (c-1) \cdot (1-p_r) \cdot (1-p_{abort})}.$$

Note that the equation refers to p_{abort} on both sides. Solving it analytically involves the so-called Lambert W-function.

5.2 Evaluation

Figure 7 compares simulation results with analytical results for the ratio of aborted versus started transactions in percent. The parameters for the simulation runs are the same as in Section 3.2. For clarity, we only show the graphs where the maximum cache size is 25% of the database size. The results for the alternative maximum cache sizes (15% and 35%) are similar – in fact the cache sizes hardly exert any influence on the corresponding results.

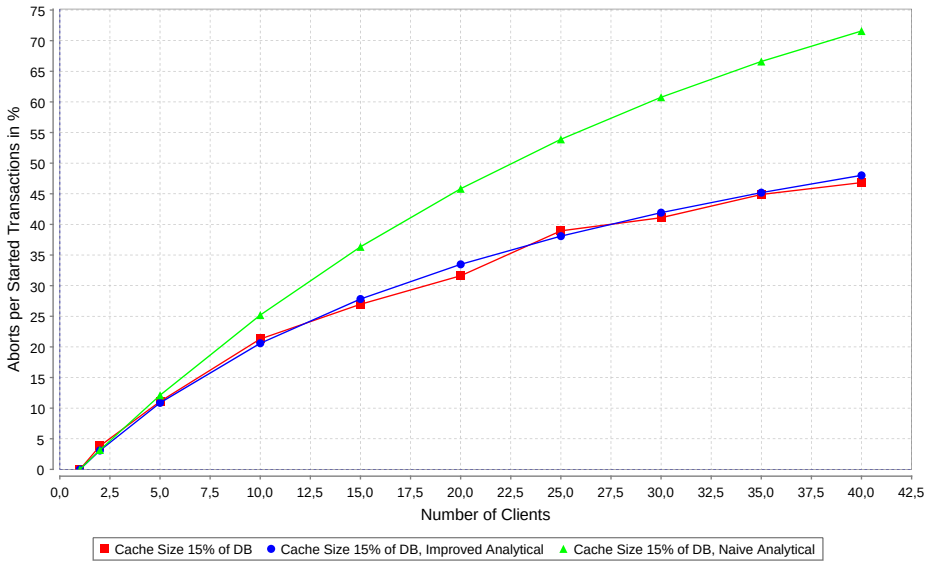

Fig. 7: Aborted Versus Started Transactions in Persent for OCC (No Early Aborts)

As one can see, the improved analytical solution predicts the analytical results very well, where as the naive approach does not. For the improved analytical solution, the numerical values of p_{abort} are computed using the (mathematical) secant method.

The version of the OCC protocol applied for Figure 7 does *not* perform early aborts. When allowing early aborts, a server terminates a transaction as soon it is guaranteed that the transaction will not pass its validation phase ([Gruber 1997]). In many cases the server can detect this event even before the respective transaction has performed its entire set operations. Early aborts have an impact on the average number of concurrent transactions and therefore, they reduce the accuracy of the analytical solution from above. To illustrate this fact, Figure 8 also presents simulation results of an OCC version using early aborts.

5.3 Conclusion

This paper has presented a simple but highly accurate analytical framework for predicting important quality parameters of the two transactional cache protocols OCC and CBR under the *UNIFORM* workload.

Firstly, we considered the filling size of a client cache: Our model explains why the cache filling size can considerably differ from the maximum cache size even after many operations and also and predicts this effect. The phenomenon is important because it may lower the cache hit rate and affect a system's efficiency. Surprisingly, it has not been considered by former publications in the field of transactional caching.

Secondly, we gave comprehensive formulas to predict the number of messages that a client and the server exchange when processing a committing transaction. The formula for CBR highlights the quantitative effect of callbacks.

At last, we improved a well-known formula for estimating transaction abort rates of optimistic protocols such that it becomes highly accurate with respect to OCC. In this context, it matters to model the fact that only committing transactions can cause other transactions to be aborted.

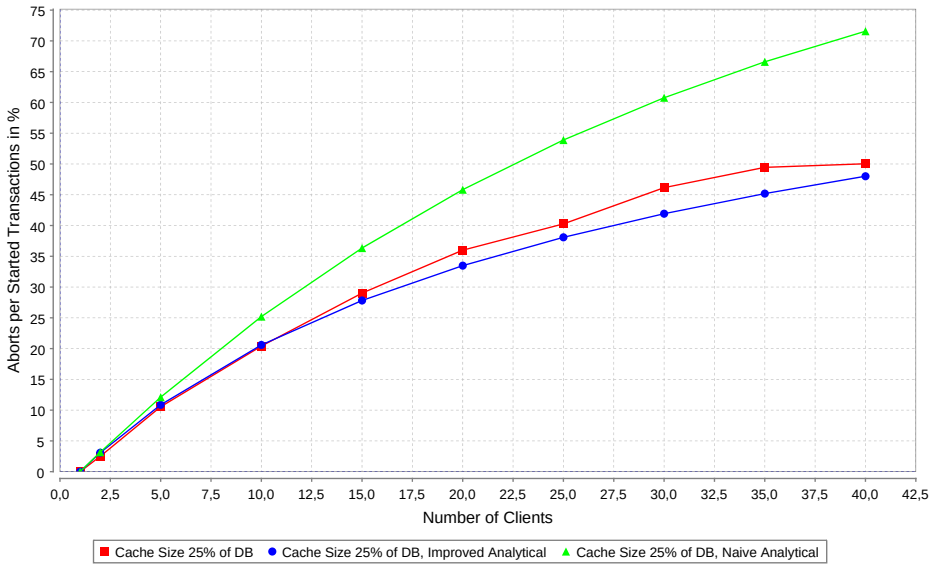

Fig. 8: Aborted Versus Started Transactions in Persent for OCC (Early Aborts)

All analytical solutions where compared with corresponding simulation results. Analytical and experimental findings gave mutual evidence for their correctness.

The stated formulas had a simple structure. One reason for this is the simple nature of the *UNIFORM* workload. Unfortunately, more complex workloads such as *HOTCOLD* also demand for more complex mathematical modelling tools and models (e.g. Markov models). Since more complex models are inherently more error-prone, they are also less reliable when trying to validate experimental studies. Therefore, in order to check the correctness of a dynamic system, it is favorable to have both, simple but comprehensive analytical system models, which cover basic behavior and sophisticated models to handle advanced cases. For the field of transactional cache protocols, this paper has taken a first step in this direction.

REFERENCES

ADYA, A., GRUBER, R., LISKOV, B., AND MAHESHWARI, U. 1995. Efficient optimistic concurrency control using loosely synchronized clocks. In *Proceedings of the 1995 ACM SIGMOD Conference on Management of Data*. ACM Press.

BERNSTEIN, P., HADZILACOS, V., AND GOODMAN, N. 1987. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley.

CAREY, M. J., FRANKLIN, M. J., LIVNY, M., AND SHEKITA, E. J. 1991. Data caching tradeoffs in client-server dbms architectures. In *Proceedings of the 1991 ACM SIGMOD Conference on Management of Data*. ACM Press, 357–366.

CAREY, M. J., FRANKLIN, M. J., AND ZAHARIOUDAKIS, M. 1994. Fine-grained sharing in a page server oodbms. In *Proceedings of the 1994 ACM SIGMOD Conference on Management of Data*. ACM Press, 359–370.

CHU, S. I. AND WINSLETT, M. 1994. Minipage locking support for object-oriented page-server dbms. In *CIKM '94: Proceedings of the third international conference on Information and knowledge management*. ACM Press, New York, NY, USA, 171–178.

CHUNG, I., LEE, J., AND HWANG, C.-S. 1997. A contention based dynamic consistency maintenance scheme for client cache. In *CIKM '97: Proceedings of the sixth international conference on Information and knowledge management*. ACM Press, New York, NY, USA, 363–370.

FRANKLIN, M. J. AND CAREY, M. J. 1992. Client-server caching revisited. In *International Workshop on Distributed Object Management*. 57–78.

FRANKLIN, M. J., CAREY, M. J., AND LIVNY, M. 1997. Transactional client-server cache consistency: Alternatives and performance. *ACM Transactions on Database Systems 22*, 3, 315–363.

GRUBER, R. 1997. Optimism vs. locking: A study of concurrency control for client-server object-oriented databases. Tech. Rep. MIT-LCS-TR-708, MIT. Februar.

ÖZSU, M. T., VORUGANTI, K., AND UNRAU, R. C. 1998. An asynchronous avoidance-based cache consistency algorithm for client caching dbmss. In *Proceedings of the 24th Conference on Very Large Databases*. Morgan Kaufmann, 440–451.

PFEIFER, D. 2005. DBCacheSim – a simple simulation framework for transactional client server database cache protocols. `http://www.ipd.uka.de/~pfeifer/`.

THOMASIAN, A. 1998. Concurrency control: methods, performance, and analysis. *ACM Comput. Surv. 30*, 1, 70–119.

WANG, Y. AND ROWE, L. A. 1991. Cache consistency and concurrency control in a client/server dbms architecture. In *Proceedings of the 1991 ACM SIGMOD Conference on Management of Data*. ACM Press, 367–376.

WILKINSON, W. K. AND NEIMAT, M.-A. 1990. Maintaining consistency of client-cached data. In *Proceedings of the 16th Conference on Very Large Databases*. Morgan Kaufmann, 122–133.

WU, K., FEI CHUANG, P., AND LILJA, D. J. 2004. An active data-aware cache consistency protocol for highly-scalable data-shipping DBMS architectures. In *Proceedings of the 1st Conference on Computing Frontiers*. ACM Press, 222–234.

ZAHARIOUDAKIS, M., CAREY, M. J., AND FRANKLIN, M. J. 1997. Adaptive, fine-grained sharing in a client-server oodbms: A callback-based approach. *ACM Transactions on Database Systems 22*, 4, 570–627.