Improving Performance in Replicated Databases through Relaxed Coherency[†]

Rainer Gallersdörfer Matthias Nicola

RWTH Aachen, Informatik V, Ahornstr. 55, D-52056 Aachen, Germany
{gallersd,nicola}@informatik.rwth-aachen.de, Fax: +49-241-8888-321

Abstract

Applications in finance and telecommunications (intelligent network, network management, mobile computing) cause renewed interest in distributed and replicated data management. Since synchronous update of replicated data is experienced to degrade distributed systems performance substantially, relaxing the requirement of *coherency* (*mutual consistency*) has become a favorable approach to achieve high performance in replicated databases. In this paper we present formal concepts for specifying relaxed coherency which allows to calculate an independent measure of relaxation, called *coherency index*. We incorporate this metric into a detailed analytical queueing model which emphasizes on the quality of replication to evaluate the impact of relaxed coherency on the performance of replicated databases. The model considers response time, throughput, scalability and network traffic as performance criteria. As it turns out, performance improvements through relaxed coherency depend significantly on various system parameters. We closely examine the trade off between consistency and performance, and show that in many situations a slight relaxation of coherency can increase performance remarkably. Finally, we verify these results by benchmarking an implementation of relaxed coherency.

1 Introduction

In a distributed environment data is replicated in order to achieve shorter response times, higher throughput and increased availability and reliability in case of failures. The approach is to place data where it is to be processed as well as storing redundant data at independent sites. In the literature benefits of replication are generally accepted and a lot of methods have been proposed for ensuring consistency in distributed systems mostly concerned with the problem of providing atomicity [12]. Unfortunately, these algorithms either destroy the property of availability by synchronous access to other sites or lengthen response times by introducing remote access. One concept to avoid these problems is known as *relaxed coherency* which means buying performance or independence for giving up "up-to-date-ness" of the data as far as the application can tolerate it. Stock trading and telecommunication are some out of several examples where controlled inconsistency is expected to gain high performance [2,33]. Naturally, mobile databases and federated systems are asking for independence.

Our research was initiated by INDIA[1], a joint project with PHILIPS regarding database support for a flexible telecommunication control architecture called *intelligent network (IN)* [15]. The analysis of the application showed that coherency relaxation is needed for performance but that the specific needs concerning consistency and actuality of data and transactions are more subtle than is covered by current models. The first contribution of this paper is therefore a detailed classification scheme by which coherency conditions can be specified.

Based on this analysis, the INDIA project designed and implemented a technique called *Atomic Delayed Replication* (ADR) and tested it in an experimental IN environment; the approach is described in [15]. A large set of measurements were conducted with the system, consisting of millions of transactions and thus demonstrating the scalability and stability of the implementation. However, due to the enormous overhead, measurement was limited to just a few database nodes (up to five). Additionally, it proved difficult to compare various strategies as each design decision has to be

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.
Proceedings of the 21st VLDB Conference
Zurich, Switzerland, 1995

† This work was supported by Philips Research Laboratories Aachen, 52021 Aachen, Germany.

[1] INDIA stands for Intelligent Networks as a Data Intensive Application

implemented explicitly. The second contribution of this paper is therefore a detailed analytical model that can be used to analyze the impact of various design parameters on replicated databases in terms of response time, throughput, scalability and network load.

Since an implemented system and a lot of measurement data are available, we are thirdly also able to validate the model against these benchmark results.

Section 2 starts with a discussion about coherency in database systems and classifies coherency conditions to show how application requirements can be described. In section 3 a replicated database system is modeled by an open queueing network. Our model is more detailed than most others found in literature, especially concerning quality of replication, distribution of data access and composition of workload. In section 4 we present the analytical results under certain parameter variations (meaning different designs of fragmentation, replication and system size). A validation of the model is given in section 5 by presenting benchmark results that were forecasted by our analytical model. Section 6 concludes.

2 Coherency in replicated databases

In this section we define coherency in replicated databases and discuss related work on specifying relaxed coherency. Then we classify concepts describing the different dimensions along which relaxation of coherency can be formalized.

2.1 Definition of terms

In replicated databases we distinguish between a logical data item (e.g. *D*) and its physically stored representatives called *replicas* (e.g. D_1, D_2, ..., D_n). Focusing on one of these instances, D_1, we call all other physical instances the *copies* D_2, D_3, ..., D_n of D_1 [7]. Usually, all the replicas of a logical data item are expected to have the same value. This property is known as *coherency* or *mutual consistency*. Assuring *mutual consistency* is beyond classical *consistency control* and therefore called *replica control* [26]. A system that ensures both *mutual consistency* and *serializability* is said to provide *one copy serializability* [5]. This is a commonly known correctness criteria for concurrent transactions accessing replicated data and is often enforced by the two-phase-commit (2PC). As the 2PC was experienced to degrade systems performance substantially, relaxing the requirement of mutual consistency has become a favorable approach to achieve high performance in replicated databases [9].

It must be possible to specify the allowed *degree* of deviation of replicas since relaxed coherency may be employed with respect to application semantics only. *Coherency control* then has to ensure that this specification of relaxed coherency is not violated. In the following we sketch some approaches in the literature for specifying relaxed coherency and then describe our classification of coherency conditions that will be analyzed later on.

2.2 Specification of relaxed coherency

Specifying the degree of relaxation must be formalized so that the influence of relaxed coherency on the performance can be measured. Our analysis will show that the relaxation of coherency can improve the performance of replicated databases. In the following section we first describe proposals for such a specification and then build a classification of coherency conditions.

2.2.1 Related Work

Current technology like Sybase Replication Server, Oracle Symmetric Replication Technology, Ingres Replicator and transaction monitors like DEC-ACMS provide basic capabilities for the physical replication of data fragments. Unfortunately they allow relaxation of coherency at fixed levels only: classical 2PC, primary-secondary approach, optimistic using timestamps or completely uncontrolled. These commercial systems do not provide mechanisms which support application specific coherency conditions. On the other hand such commercial systems represent a mature base for implementing coherency control on top of them; indeed, our implementation is based on Sybase Servers.

In the literature Alonso et al. described *coherency conditions* in client-server environments where the data is stored and controlled in the servers [2]. Since clients are usually workstations or personal computers with storage and processing capacity, parts of the information is cached at the clients to off-load both servers and network. Updates can be applied to data at the server side meaning a primary copy paradigm only. Aging of cached data is allowed to avoid complicated algorithms ensuring cache coherency at the clients. The cached data items are called *quasi copies* and are similar to materialized views [22] and database snapshots [1,27]. Generally, caching promises reduced network and server load as well. In the quasi copy approach the degree of coherency relaxation has to be specified at the client side by means of coherency conditions like e.g.

$$\forall \text{ times } t \geq 0 : \quad \exists\, k : (0 \leq k \leq \alpha) \wedge (x'(t) = x(t-k))$$

which states that a client's quasi copy x' may not be older than α time units compared to servers original x. This means defining a time window α which can be expressed by the unary predicate $W(x) = \alpha$.

A more general approach called *identity connections* was described by Wiederhold and Qian [44,45]. Identity connections are directed or undirected relations between replicas of a logical data object or between source and derived data. In the second case a function has to be supplied describing how to compute the derived data object. Every identity connections must be provided with a *temporal constraint* that defines the allowed degree of divergence. Such constraints may be time points and events or time intervals. Appropriate propagation of updates between coupled data items has to ensure the enforcement of identity connections.

Sheth and Rusinciewicz presented an extension of identity connections for multi database systems called *database dependency descriptors (D^3)* [34,35]. A D^3 is a tuple <S, U, P, C, A> where S is a set of source data and U a target data object. P is an expression in relational algebra which describes the intended value of U, C is a coherency predicate specifying *when* P must be enforced and A is a set of restoration procedures that provide actions to update the target object U. Though identity connections and database dependency descriptors are readily formalized for practical use, proving global correctness is difficult as explicitly coupling of copies in pairs leads to a large number of specifications. With *ASPECT* Lenz et al. proposed an application oriented specification of consistency requirements for replicated data [26]. Here the application shall specify the relaxation of coherency with respect to the logical data object and not to other copies. Therefore the application must neither care about other copies nor about their existence. Unfortunately there is no formal notation for ASPECT. Furthermore it is not clear how to derive the current value of a logical data object.

2.2.2 Classification of coherency conditions

Lots of ideas to specify relaxed coherency have been proposed in the literature [2,4,8,9,26,34,35,44,45]. In this section we classify those concepts and denote them through coherency conditions which are an extension to the formalism in [2]. Generally, our coherency conditions are binary relations between replicas x and x' of the same logical data item. For each class of coherency conditions we will try to calculate a numerical value named *coherency index* (*k*) to measure the degree of allowed divergence. This enables us to evaluate the influence of coherency relaxation in the performance analysis independently from the class of condition.

1. Delay conditions:

The first class of conditions are time oriented, expressing how long the system may wait until the propagation of an update has to be initiated. The time window *m* describes the maximum delay:

$$D(x,x') = m \quad (m \geq 0)$$

For example, *D(x,x') = 60 seconds* means, that any update on data item x must be propagated to x' within a minute. Thus, delay conditions specify how much time a replica may lag behind another replica of the same logical data object. To be general, a coherency condition need not be symmetrical (D(x,x') ≠D(x,x')). If data item x is updated $\lambda_u^{(x)}$ times per second not every update but only the latest state of x has to be propagated. Hence, the actual *probability of propagation* is

$$k = 1/((\lambda_u^{(x)} \cdot m) + 1)$$

which yields an important foundation of increased performance due to relaxed consistency. This $k \in [0;1]$ is named *coherency index.* Small values of *k* express high relaxation and expected low costs for update propagation. With $k = 0$ ($m \rightarrow \infty$) values of replicated data objects age unlimited. For $k = 1$ ($m = 0$) all updates have to be propagated immediately; note that this does not imply synchronous change.

2. Periodic conditions:

The second class of coherency conditions is also time oriented. Periodic conditions specify that a copy x' of x must be updated with the latest value of x every *m* time units, no matter whether the value of x has changed or not. Periodic conditions are denoted by

$$P(x,x') = m \quad (m > 0)$$

If the rate of updates $\lambda_u^{(x)}$ on data item x is greater than $1/m$, not every update but only the latest state of x has to be propagated. This leads to a coherency index $k = 1/(\lambda_u^{(x)} \cdot m)$. If $\lambda_u^{(x)}$ becomes less than $1/m$, *k* exceeds its limit 1 which indicates that superfluous updating on x' is taking place. Thus it is not obvious, why periodic conditions might be preferable to delay conditions. The advantage of updating replicas periodically is that undetected losses of update messages due to network or site failures can be avoided. This prevents replicas from aging arbitrarily. Although $\lambda_u^{(x)} < 1/m$ might be avoided by reasonably estimating $\lambda_u^{(x)}$ and carefully choosing *m*, a realization of periodic conditions could update x' *only if* x was modified during the period. Such an implementation yields a coherency index $k = 1/((\lambda_u^{(x)} \cdot m) + 1)$ and makes periodic conditions similar to delay conditions.

3. Time point conditions:

This class of conditions introduced by Wiederhold and Qian [44, 45], is a special case of periodic conditions. A time point condition is expressed as

$$T(x,x') = \tau$$

where τ denotes a repeating time point like *every hour* or *daily at eight o'clock*. A formal syntax for specifying time points is proposed in [34]. If the duration between two successive time points amounts to m time units, the coherency index results in $k = 1/(\lambda_u^{(x)} \cdot m)$.

4. Version conditions:

Version conditions specify relaxed coherency considering the number of updates occurring on data item x as the dimension of deviation. The denotation

$$V(x,x') = i \quad (i \geq 1)$$

means, that after the ith update on replica x its copy x' must be updated with the current value of x. The resulting coherency index is $k = 1/i$.

5. Numerical conditions:

If the value of a logical data item is numeric, the deviation between its replicas can be bounded by the difference of their values. Considering absolute, relative or percent difference[2], we write numerical conditions as

(a) $N_a(x,x') = \delta \quad (\delta \in R)$
(b) $N_r(x,x') = \delta \quad (\delta \in R)$
(c) $N_{\%}(x,x') = \delta \quad (\delta \in R)$

meaning that x' must be updated with the current value of x whenever

(a) $|x' - x| < \delta$
(b) $\left|\frac{x' - x}{x}\right| < \delta$
(c) $\left|\frac{x' - x}{x}\right| \cdot 100 < \delta \ \%$

happens to be violated. Though checking this kind of coherency conditions requires to know the values of both replicas, the saving may be remarkable: consider a stock market information system where prices change continuously within a small range, but users are only interested in deviations of more than ±5 %. A general coherency index cannot be calculated as long as it is impossible to estimate the amount of modifications to numerical data.

6. Object conditions:

This class of conditions is object based. The notation is:

(a) $O_{\#}(x,x') = i \quad (i \geq 1)$
(b) $O_{\%}(x,x') = q \quad (q \in [1;100])$
(c) $O(x,x') = a \quad (a \in \text{subobjects}(x))$

meaning that x' must be updated with the current value of x whenever

(a) at least i subobjects of x have been modified
(b) at least q percent of the subobjects of x have been modified
(c) subobject a of x has been modified

since the latest update of x' with the value of x. We expect such conditions to be useful for engineering applications (computer aided design) and other distributed environments based on object oriented databases. But object-subobject relationships exist in the relational model as well e.g. table-column, table-row, row-attribute. Imagine replica x' as a read-only table which is used for statistical calculations. Then it might be reasonable to update x' only if more than q% of the rows in x have been modified because otherwise a significant change of the statistical results is not to be expected. As for numerical conditions a general coherency index is difficult to derive.

7. Event conditions:

Finally, updating replicated data can be event driven which we denote like

$$E(x,x') = \varepsilon$$

where ε is any event observable by the database system. This is the most general class of coherency conditions and its realization is tightly related to mechanisms developed in active databases [13]. A formal syntax for specifying events is beyond the scope of this paper but can be found in [19, 33].

3 Modeling a replicated database

In section 3.1 we present a queueing network model for replicated databases and sketch how to obtain important performance criteria to judge relaxed coherency in section 3.2. It is based on the *coherency index (k)* introduced in section 2.2.2. So it is generally applicable to each condition although we have provided example calculations of k for the conditions 1-4 only. As stated for the periodic conditions this k heavily depends on the real implementation. Details are provided in [31].

3.1 Parameters

In our model a replicated database consists of n identical *local databases* (or *sites*) and d logical data items where $d = m \cdot n$ for $m \geq 1$. Initially, each logical data item is represented by *exactly one* physical replica, such that every site contains m distinct replicas. Actual replication

[2] Note that relative and percental difference are redundant but useful for ease of notation.

of data items will be considered later on. This distributed database is modeled as an open queueing network. Its nodes are identical $M/H_2/1$-systems characterizing the local databases. We assume only one service channel and no real parallel transaction processing at each local database system. This is founded by the trend of having lots of commodity systems building the distributed environment [14,36]. The length of the queues is assumed to be unlimited as we do not expect the buffering of incoming transactions to be a bottleneck in a distributed database. As generally accepted we assume jobs to be served in FCFS discipline [6].

We model the arrival of transactions to the distributed system by a Poisson process with parameter λ^{global} because Poisson streams have been found to be a good approximation for a large number of users submitting jobs independently [6,20,21,24]. Although read-only transactions are easier to process for a database system than updates [18], many models consider updates only [10,11,28,37]. We model the percentage of queries in the overall workload by the parameter $a_q \in [0;1]$. This means that λ^{global} is split up into two separate Poisson streams with rates $\lambda_q^{global} = a_q \cdot \lambda^{global}$ and $\lambda_u^{global} = (1-a_q) \cdot \lambda^{global}$. Furthermore, we assume that arriving transactions are distributed uniformly across the n local databases, so each site is loaded with a Poisson process of updates (parameter $\lambda_u = \lambda_u^{global}/n$) and queries ($\lambda_q = \lambda_q^{global}/n$). Achieving such a load distribution in a real system is the task of designing the distributed database, so that each local database serves the same number of users. These users submit their transactions to the local system only, which may need to forward the execution to another site due to a lack of appropriate local data. Additionally, our model is based on the assumption that updates are executed according to the *primary copy approach.* Furthermore we assume a good design in the sense that each transaction accesses data items of only one local database because transactions are expected to reference logically dependent data items which should be grouped together [10]. The *quality of data distribution* is modeled by the probability ($loc \in [0;1]$) that a transaction can be executed at the local site. With probability $1 - loc$ a transactions has to be forwarded to one of the remaining sites each of which being chosen with equal likelihood. Expressing that local data objects are accessed s_{loc} times as often as remote sites we define

$$loc(n) = s_{loc} \cdot 1/n \quad (s_{loc} \in [1;n]).$$

Although most models of replicated databases assume *full* replication [17,28,37-42], we believe that *partial* replication is necessary to achieve high performance. Therefore we model the degree of replication by the parameter $r \in [0;1]$ describing the percentage of logical data items that are fully replicated across the sites. That means that *if* a data item is replicated a copy exists at each site. Updates executed at a local database therefore have to be propagated to all other sites with probability r. Considering relaxed coherency the *probability of propagation* decreases to $k \cdot r$.

The *quality of replication* depends on the preference of queries and updates to access replicated data. More precisely we call a replication schema *of high quality* if queries are accessing replicated data as well as updates are performed on nonreplicated data to a very high degree. These preferences are modeled by the parameters $qr \in [0;1/r]$ and $ur \in [0;1/r]$ respectively. Here a value of $qr = 1/r$ ($ur = 1/r$) describes that queries (updates) are accessing replicated (nonreplicated) data only; meaning an optimal replication schema. The value 0 expresses the opposite extreme while the value 1 describes no preferences. Since the access patterns are expected to change with the degree of replication, we model them by the following functions:

$$qr(r) = \frac{1}{r^{s_{qr}}} \quad \text{and} \quad ur(r) = \frac{1}{r^{s_{ur}}}$$

This allows us to describe the ability of creating a good (or bad) design by tuning the parameters s_{ur} and s_{qr}. Taking the functions $loc(n)$, $qr(r)$ and $ur(r)$ into consideration, we clearly refrain from assuming uniformly distributed access to data objects across the database. This is another major difference to most models proposed in literature [3,10,11,17,28,29,37-42,43]. The *probability of propagation* now amounts to $r \cdot k \cdot ur$.

Usually, the time required to process a transaction is mainly determined by the disk service time [25]. This in turn is closely related to the number of data objects referenced. Since transactions that access a small number of data items are expected to occur more frequently than transactions that reference a large number of data objects [17,37] the service times can be assumed to be exponentially distributed. In order to distinguish between updates and queries we model the query (update) service time to be exponentially distributed with mean t_q (and t_u respectively). This leads directly to the two phase hyperexponential distribution of service time for the combined flow [23,24].

The communication network is assumed to affect the performance by introducing a constant delay in every intersite communication. Therefore the network is modeled by an infinite server, which means that all intersite communication is served in parallel with a constant service time of $t_n^{message}$ seconds for short messages (like sending a „prepare to commit" message) and t_n^{data} seconds for transmitting data (e.g. query results). The distinction between short and long messages is missing in many proposed models for distributed databases [10,11,28,29,37-42].

Parameter	Description	Base setting
n	Number of sites (system size)	50
M	Number of primary copies at each site	3000
b	Average number of data items modified by an update	15
λ^{global}	Global arrival rate (transactions per second)	100
a_q	Percentage of read-only transactions (queries)	0.9 (i.e. 90%)
λ_q^{global}	Global arrival rate: queries/sec	90
λ_u^{global}	Global arrival rate: updates/sec	10
λ_q	Local arrival rate at each site: queries/sec	1.8
λ_u	Local arrival rate at each site: updates/sec	0.2
loc	Quality of data distribution	0.04
s_{lok}	Adjusting the function *loc(n)*	2
r	Degree of replication	[0 ; 1]
qr	Preference of queries reading replicated data	$= 1/r^{s_{qr}}$
ur	Preference of updates changing replicated data	$= 1/r^{s_{ur}}$
s_{qr}	Adjusting the function *qr(r)*	0.6
s_{ur}	Adjusting the function *ur(r)*	-0.8
k	Coherency index	[0 ; 1]
$t_n^{message}$	Communication delay for short messages	100 ms
t_n^{data}	Communication delay for transmissions of data	400 ms
t_q	Average query service time	75 ms
t_u	Average update service time	250 ms

Table 1: System parameters and base values.

3.2 Performance values

Since our queueing network is amenable to product form solution [23], we can analyze each node separately to obtain the performance measures. Additionally, all sites of the network behave identically and transitions of jobs between the nodes are symmetrical. Therefore we examine only one site to calculate the performance values of the whole system.

The overall average response time $\overline{R}$ can be derived using the average response time for queries ($\overline{R}_q$), the average response time for updates ($\overline{R}_u$) and the overall rates of operations (queries and updates respectively) at a local database as

$$\overline{R} = \frac{\lambda_q^{total} \cdot \overline{R}_q + \lambda_u^{total} \cdot \overline{R}_u}{\lambda^{total}}$$

The appendix contains the formal derivation of this result. In steady state the number of arriving transactions equals the number of departing transactions (*flow in = flow out*). Thus, the *throughput* of the distributed database equals the global arrival rate λ^{global}. However, the overall throughput of the system is bounded by the capacity of the local sites. Therefore we derive the *maximum throughput D* by solving the equation $\lambda_q^{total} t_q + \lambda_u^{total} t_u = 1$ for λ^{global} which results in

$$D = \lambda^{global} = \left(\frac{a_q}{n} \cdot t_q + (1 + (n-1) \cdot r \cdot k \cdot ur) \cdot \frac{(1-a_q)}{n} \cdot t_u \right)^{-1}$$

The average number of messages per second in the distributed system includes the shipment of user transaction as well as update propagation messages and amounts to

$$\overline{N} = 2 \cdot n \cdot (1 - \ell_q) \cdot \lambda_q + 2 \cdot n \cdot (1 - \ell_u) \cdot \lambda_u + n \cdot (n-1) \cdot (r \cdot k \cdot ur) \cdot \lambda_u$$

with ℓ_q (ℓ_u) as the probability for local execution of queries (updates).

4 Results

Table 1 summarizes the parameters of our model including the base settings used to obtain the following graphs. The values are carefully estimated using

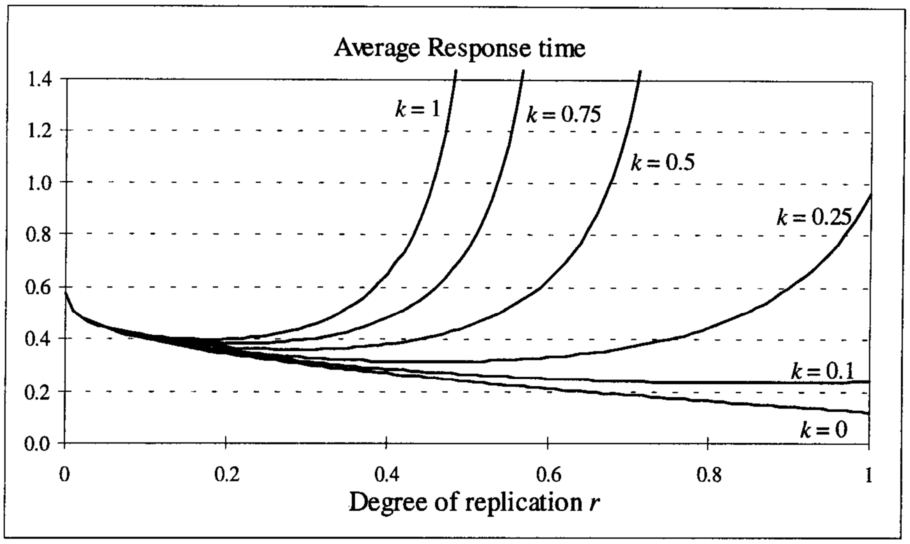

Figure 1: Average response time.

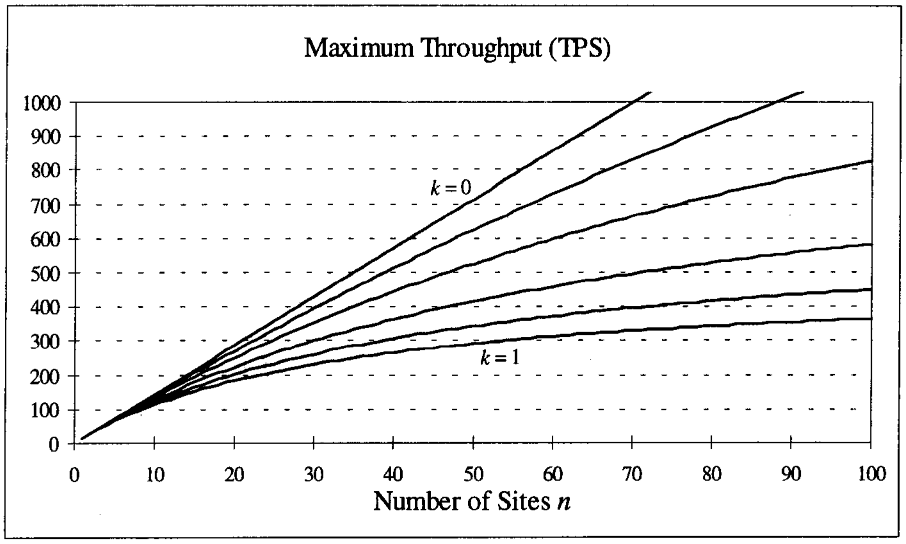

Figure 2: Maximum throughput vs. system size.

measurements in a real system and considering assumptions of other models found in the literature. Naturally the base settings influence the numerical results in our graphs but we regard the absolute values as a secondary outcome. We rather consider the *shapes* of the curves as the primary results since they indicate the general gains of relaxed coherency and stay stable under variation of base values [31]. Figure 1 shows the average response time versus the degree of replication for several values of the coherency index k. The curve for $k = 0$ shows the behavior with no propagation at all. The case $k = 1$ represents asynchronous but immediate propagation of updates. In this situation the response time can be reduced remarkably by replicating 20% of the data as this leads to increased local access. However, extending the degree of replication for sake of reliability rapidly saturates the local databases with propagated updates ($r > 0.4$). Now, relaxing the coherency requirements ($k < 1$) lessens the effort of update propagation and thus gains an improvement in response time. For $k = 0.25$, update propagation is reduced by a factor of 4 which allows the response time (at $r = 0.5$) to be decreased by 50% compared to the non-replicated case ($r = 0$). When coherency is relaxed drastically ($k \leq 0.1$), response time can be minimized by means of full replication.

In figure 2 the maximum throughput is shown as a function of n where the coherency index is taking the values 0, 0.1, 0.25, 0.5, 0.75 and 1 (from top to bottom). Although most evaluations found in literature consider at most 10 sites [11,28,29,37-43] we run the number of sites up to 100 since we agree that future parallel database systems will consist of hundreds of sites [14,32]. If the percentage of updates is not negligible (like 10% in figure 2), throughput does *not* increase linearly with the number of sites due to propagation (when $k > 0$). On the

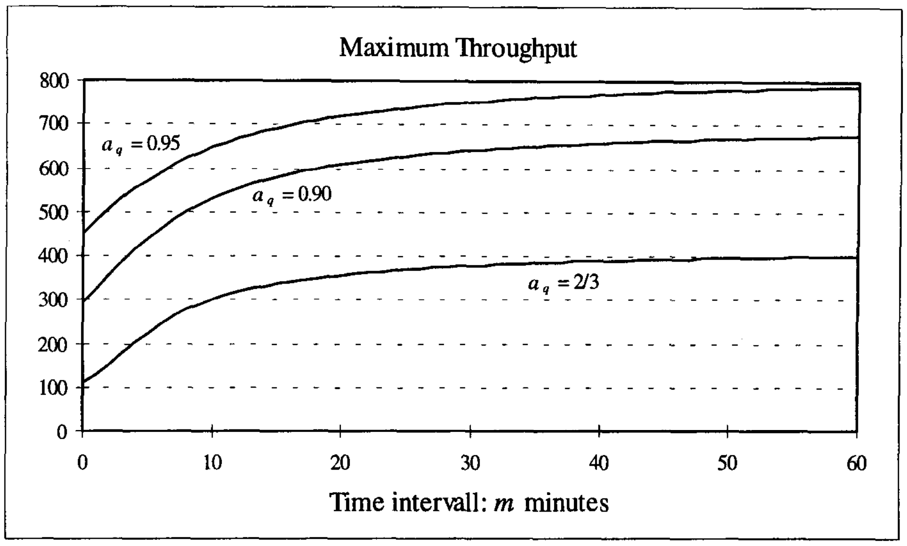

Figure 3: Maximum throughput using delay or periodic conditions.

other hand the graphs for $k < 1$ reveal that relaxing coherency may improve scalability up to ideal linearity [14]. Figure 2 also shows that for a given number of sites throughput can be increased by relaxing coherency and the larger the system the greater the gain. Therefore, relaxed coherency applies to very large distributed databases of terabyte scale.

The challenging question for any practical employment of relaxed coherency is: What magnitude of performance improvement can be expected when relaxing coherency to a certain extent? Lets assume that every pair of primary and secondary copies (x,x') obeys to a global delay condition D(x,x') = m. Approximating the update arrival rate on a primary copy x as $\lambda_u^{(x)} = (b \cdot \lambda_u)/M$ leads to a coherency index

$$k = \frac{1}{(\lambda_u \cdot b \cdot m/M) + 1}$$

Using this result the throughput versus the time interval of delay (or periodic[3]) conditions is depicted in figure 3. As it turns out the transaction processing capacity of the system depends heavily on the percentage of read-only transactions (a_q): Decreasing the number of updates leads to a higher maximum throughput but reduces the gain of relaxed coherency. E.g. Considering 5% updates in the workload, throughput grows from 455 TPS to about 790 TPS if update propagation is disabled for an hour (or reduced to „once an hour“). This is an increase of 75%. If the fraction of read-only transactions is reduced to 66% the maximum throughput can reach 400 TPS only, but this is an increase of almost 300% and delaying update propagation for 5 minutes doubles the throughput already. Since a coherency index of less than 0.25 can be achieved by delaying updates for 10 minutes, the cases $k = 0.25$ or $k = 0.1$ in the previous figures are likely to come true. Furthermore we believe that many applications in distributed environments can accept data which is 5 to 30 minutes old.

In figure 4 we illustrate the impact of replication and relaxed coherency on the network traffic. In the case of immediate propagation ($k = 1$) a moderate degree of replication (e.g. $r = 0.2$) reduces the number of messages due to less remote access of data items, but extended replication floods the network as updating of replicas outweighs the advantage of local access. However, the curves for k < 1 indicate that even in a highly replicated system network traffic can be kept low if relaxed coherency is used: considering 80% replication the number of messages per second may be reduced to 200 (which equals the traffic in the non-replicated case) if a coherency of $k = 0.5$ can be established e.g. by implementing periodic conditions with a well tuned time window. If $k < 0.1$ is achievable, network traffic (as well as response time) can be minimized by full replication. This is very encouraging since extensive replication is necessary to build a reliable and fault-tolerant distributed system.

5 Comparing analytical and benchmark results

Modeling a computer system implies making lots of assumptions and simplifications. Thus, the analytical results are in need for validation. We tried to verify our analytical model by benchmarking the implementation of *Atomic Delayed Replication* (ADR) which is a novel technique for replica management and concurrency control in distributed databases. ADR is based on the idea

[3] Remember from Section 2.2.2 that periodic and delay condition result in nearly the same coherency index and thus improve performance equally well.

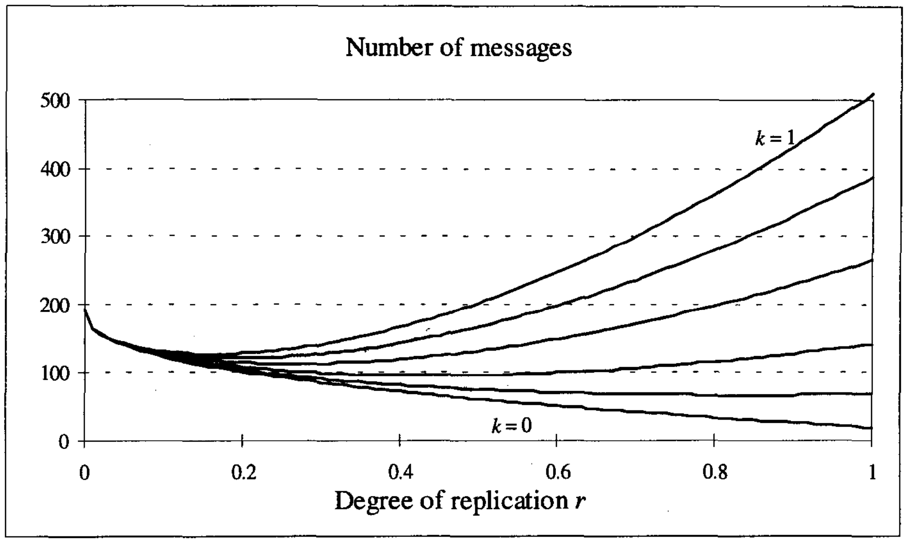

Figure 4: Average number of messages per second $\overline{N}$.

of asynchronous propagation with controlled levels of consistency to achieve shorter response times and increased throughput with high autonomy while keeping the possibility for a flexible fragmentation and replication design with respect to scalability [15]. The first implementation of ADR was realized on top of SYBASE System 10 running under SunOS 4.1.3 [30]. This happened in the context of the INDIA project which aims at efficiently supporting intelligent networks (IN) in telecommunication with replicated databases.

Our computing environment consists of SUN Sparc Station 10 as both database server machines and client environments. The local databases and their logs are placed on separate disks and controllers to avoid a bottleneck in the I/O-system. The database size was set to about 100 MB for each local database system. The database schema corresponds to a currently existing realization of a complete IN system developed by PHILIPS research laboratories. We benchmarked the ADR system with up to five local databases. As currently no standardized load models for the IN application exist, we decided to use two kinds of transactions: One is a short read-only transaction typical for the execution of phone calls. The other is a more complicated write transaction changing the profile of an intelligent network service (i.e. virtual private network). We measured the throughput of the whole system while decreasing the percentage of read-only transactions from 100% down to 0%. The duration of the benchmark was one week; millions of transactions were executed and showed the stability of the implementation [16].

The implementation of ADR reproduces changes of primary copies at the secondary copies asynchronously as eager as possible corresponding to a coherency index $k = 1$. However, it gives user transactions preference over propagation transactions at peak load situations like our benchmark runs. This leads to a coherency index $k = 0$ which is used for the comparison. To further describe the benchmark using our analytical model, we set the service times as $t_q = 50$ ms and $t_u = 100$ ms because the transactions in our synthetic workload are rather short. Varying the number of sites and the percentage of read-only transactions can be captured by $n \in [1;5]$ and $a_q \in [0;1]$. Using these values, our analytical model reveals a maximum throughput as shown in figure 5, which also shows the benchmark results for comparison.

Obviously, the analytical model is characterizing the real implementation very well. Therefore we believe in the quality of the analytical results presented in section 4. Further validation of the model through sensitivity analysis is provided in [31]. Figure 5 illustrates the ability of linear scaleup when using relaxed coherency like in ADR. Examining the measured values very critically we discovered some difficulties with the communication protocols used for the implementation if the percentage of reads is high. Thats why some curves in the picture seem to be convex. We remedied this pitfall and gathered more accurate results for selected read/write mixes with an improved implementation.

Summarizing our experiences we found that relaxation of coherency assures linear scaleup even along increased write access which is not possible using classical mechanisms for the management of replicated data. This result is an important step towards distributed databases with replicated data.

6 Summary and outlook

In this paper we defined a measure called *coherency index* for the allowed divergence of replicated data. It is based on our classification of coherency conditions

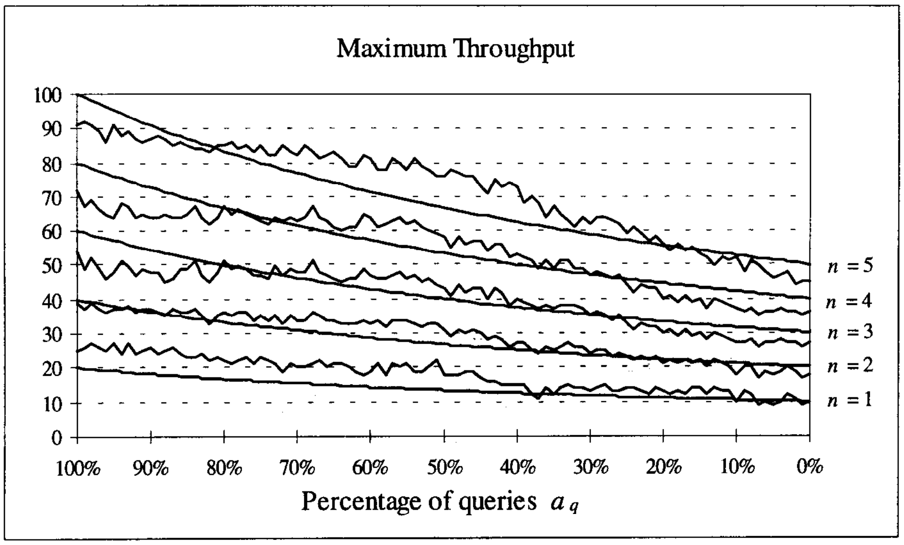

Figure 5: Comparing analytical and benchmark results.

characterizing relaxed mutual consistency as well as strategies for update propagation. According to the classification we gave examples how to calculate the *coherency index*. Then we developed a detailed queueing network model based on this coherency index to evaluate the impact of relaxed coherency on the performance of replicated databases. The results show, that relaxing classical consistency requirements can improve response time, throughput, scalability and network traffic considerably. Furthermore, our analysis reveals that these gains of relaxed coherency grow when the number of sites or the degree of replication is increased. Examining the trade off between consistency and performance we discovered, that slight relaxations of coherency can increase performance remarkably: delaying update propagation for 5 minutes may double throughput. Finally, we benchmarked an existing implementation of relaxed coherency and found that our analytical model is forecasting reality quite accurately. Currently we are extending our implementation of the method ADR incorporating periodic conditions. We expect to validate the analytical results in more detail by running the benchmark on this refinement.

References

[1] M.E. Adiba, B.G. Lindsay: „*Database Snapshots*", VLDB 80, pages 86-91.

[2] R. Alonso, D. Barbará, H. Garcia-Molina: „*Data Caching Issues in an Information Retrieval System*", ACM Transactions on Database Systems, Vol. 15, No. 3, September 1990, pages 359-384.

[3] Sujata Banerjee, Victor O K Li, Chihping Wang: „*Performance analysis of the send-on-demand: A distributed database concurrency control protocol for high-speed networks*", Computer Communications, Vol. 17, No. 3, March 1994, pages 189-204.

[4] D. Barbara, H. Garcia-Molina: „*The case for controlled inconsistency in replicated data*", Proceedings of the 1st Workshop on the Management of Replicated Data, Houston, November 1990, pages 35-38.

[5] Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman: „*Concurrency Control and Recovery in Database Systems*", Addison Wesley 1987.

[6] Gunter Bolch: „*Leistungsbewertung von Rechensystemen mittels analytischer Warteschlangenmodelle*", (in german), B.G. Teubner Stuttgart, 1989.

[7] Wojciech Cellary, Erol Gelenbe, Tadeusz Morzy: „*Concurrency Control in Distributed Database Systems*", Elsevier Science Publishers, Holland 1988.

[8] S. Ceri, M.A.H. Houtsma, A.M.Keller, P. Samarati: „*A Classification of Update Methods for Replicated Databases*", Technical Report STAN-CS-91-1392, Stanford University, Oktober 1991.

[9] S.F. Chen, C. Pu: „*An Analysis of Replica Control*", Proceedings of the 2nd Workshop on the Management of Replicated Data, Los Alamitos, November 1992, pages 22-25.

[10] B. Ciciani, D.M. Dias, P.S. Yu: „*Analysis of Replication in Distributed Database Systems*", IEEE Transactions on Knowledge and Data Engineering, Vol. 2, Nr. 2, Juni 1990, pages 247-261.

[11] B. Ciciani, D.M. Dias, P.S. Yu: „*Analysis of Concurrency-Coherency Control Protocols for Distributed Transaction Processing Systems with Regional Locality*", TSE, Vol. 18, Nr. 10, 1992, pages 899-914.

[12] Susan B. Davidson, H. Garcia-Molina, Dale Skeen: „*Consistency in Partitioned Networks*", ACM Computing Surveys, Vol. 17, Nr. 3, September 1985, pages 165-178.

[13] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, D. McCarthy, A. Rosenthal, S. Sarin, M.J. Carey, M. Livny, R. Jauhari: „*The HiPAC project: Combining active databases and timing constraints*", SIGMOD 88, pages 51-70.

[14] D. DeWitt, J. Gray: „*Parallel Database Systems: The Future of High Performance Database systems*", Communications of the ACM, Vol. 35, Nr. 6, Juni 1992, pages 85-98.

[15] Rainer Gallersdörfer, Matthias Jarke, Karin Klabunde: „*Intelligent Networks as a Data Intensive Appliction (INDIA)*", International Conference on Applications of Databases, Sweden, Juni 1994.

[16] Rainer Gallersdörfer, Karin Klabunde: „*Performance and Scalability of Atomic Delayed Replication for Distributed Databases in the Intelligent Network*", Technical Report, RWTH Aachen, Informatik V and Philips Research Laboratories Aachen, 1994.

[17] H. Garcia-Molina: „*Performance of the Update Algorithms for Replicated Data in a Distributed Database*", Ph.D. Dissertation, revised, Stanford University (June '79), North Holland, 1982.

[18] H. Garcia-Molina, G. Wiederhold: „*Read-Only Transactions in a Distributed Database*", ACM TODS, Vol. 7, No. 2, June 1982, pages 209-234.

[19] N.H. Gehani, H.V. Jagadish, O. Shmueli: „*Composite Event Specification in Active Databases: Modell & Implementation*", VLDB 92, pages 347-362.

[20] Jim Gray: „*The Benchmark Handbook for Database and Transaction Processing Systems*", Morgan Kaufmann, 1993.

[21] Donald Gross, Carl M. Harris: „*Fundamentals of Queueing Theory*", John Wiley & Sons, 1985.

[22] E. Hanson: „*A Performance Analysis of View Materialization Strategies*", SIGMOD 87, pages 440-453.

[23] Peter J.B. King: „*Computer and Communication Systems Performance Modelling*", Prentice Hall, 1990.

[24] Leonard Kleinrock: „*Queueing Systems, Volume I: Theory*", John Wiley & Sons, 1975.

[25] Henry F. Korth, Abraham Silberschatz: „*Database System Concepts*", Mc Graw Hill, 1991.

[26] R. Lenz, T. Kirsche, B. Reinwald: „*ASPECT - Specifying Consistency Requirements for Replicated Data from an Applications Point of View*", Proceedings of the International Conference on Parallel and Distributed Computing Systems, Las Vegas, Oct. 1994, pages 472-477.

[27] B. Lindsay, L. Haas, C. Mohan, H. Pirahesh, P. Wilms: „*A Snapshot Differential Refresh Algorithm*", SIGMOD 86, pages 53-60.

[28] W. Mariasoosai, M Singhal: „*A Concurrency Control Algorithm for Replicated Database Systems*", Proceedings of the ISMM International Conference, New York, Oct. 1990, pages 143-147.

[29] J. Mc Dermett, R. Mukkamala: „*Performance Analysis of Transaction Management Algorithms for the SINTRA Replicated Architecture Database Systems*", IFIP Transactions (Computer Science & Technology), Vol. A-47, 1994, pages 215-234.

[30] Ralf Nellessen: „*Transaktionen in verteilten Datenbanken - Anwendung im Intelligenten Netz*", Diploma thesis (in german), RWTH Aachen, Informatik V, Juli 1993.

[31] Matthias Nicola: „*Analytische Leistungsbewertung relaxierter Kohärenz in replizierten Datenbanken*", Diploma thesis (in german), RWTH Aachen, Informatik V, April 1995.

[32] Alison Payne: „*Designing the Databases of the Intelligent Network*", 8th International Conference on Software Engineering for Telecommunication Systems and Services, 1992, pages 37-41.

[33] G. Ramanathan, V.S. Alagar: „*Specification of Real-Time Distributed Database Systems*", Proceedings of CompEuro '92: Computer Systems and Software Engineering, 1992, pages 101-106.

[34] M. Rusinciewicz, A. Sheth, G. Karabatis: „*Specifying Interdatabase Dependencies in a Multidatabase Environment*", IEEE Computer, Vol. 24, Nr. 12, Dezember 1991, pages 46-53.

[35] A. Sheth, M. Rusinciewicz : „*Management of interdependent data: Specifying depedency and consistency requirements*", Proceedings of the 1st Workshop on the Management of Replicated Data, Houston, November 1990, pages 133-136.

[36] A. Silberschatz, P.B. Galvin: „*Operating System Concepts*" , Addison Wesley, 1994.

[37] Mukesh Singhal: „*Concurrency Control Algorithms and their Performance for Replicated Database Systems*", Ph.D. Dissertation, University of Maryland, 1986.

[38] Mukesh Singhal, A.K. Agrawala: „*Performance Analysis of an Algorithm for Concurrency Control in Replicated Database Systems*", ACM SIGMETRICS '86, Mai 1986, pages 159-165.

[39] Mukesh Singhal: „*A Fully-Distributed Approach to Concurrency Control in Replicated Database Systems*", Proceedings of the 12th International Computer Software and Applications Conf., 1988.

[40] Mukesh Singhal: „*Update Transport: A New Technique for Update Synchronization in Replicated Database Systems*", TSE, Vol. 16, 1990, pages 1325-1336.

[41] S.H. Son, C.H. Chang: „*Performance Evaluation of Replication Control Algorithms for Distributed Database Systems*", Technical Report, CS-TR-9-11, University of Virginia, 1991.

[42] U. Sumita, O.R. Sheng: „*Analysis of Query Processing in Distributed Database Management Systems with fully replicated Files: A Hierarchical Approach*", Performance Evaluation, Vol. 8, 1988.

[43] Ö. Ulusoy: „*Processing Real-Time Transactions in a Replicated Database System*", Journal on Distributed and Parallel Databases, Vol. 2, No. 4, October 1994, pages 405-436.

[44] Gio Wiederhold, XiaoLei Qian: „*Modeling Asynchrony in Distributed Databases*", Proceedings of the 3rd International Conference on Data Engineering, 1987, pages 246-250.

[45] Gio Wiederhold, X. Qian: „*Consistency Control of replicated data in federated databases*", Proceedings of the 1st Workshop on the Management of Replicated Data, Houston, November 1990, pages 130-132.

Appendix

This appendix contains some hints on the calculation of the performance values mentioned in section 3.2. The probability that submitted queries (updates) can be executed at the local database is denoted as ℓ_q (ℓ_u respectively) and results in

$$\ell_q = loc + (1 - loc) \cdot r \cdot qr \quad \text{and} \quad \ell_u = loc$$

because *loc* expresses the preference of accessing original local data and the second term reflects the local read availability introduced by replication. Note that replication does not increase the write availability because of the primary copy approach.

The overall rate of queries to be executed at a local database (λ_q^{total}) not only consists of queries submitted by users but also of additional queries received from other sites:

$$\lambda_q^{total} = \ell_q \cdot \lambda_q + (n-1) \cdot (1 - \ell_q) \cdot \lambda_q \cdot \frac{1}{n-1}$$
$$= \ell_q \cdot \lambda_q + (1 - \ell_q) \cdot \lambda_q = \lambda_q$$

Considering the identical behavior of sites $\lambda_q^{total} = \lambda_q$ is not much of a surprise: every site receives just as many queries as it forwards to other sites due to a lack of appropriate local data. For updates we similarly derive

$$\lambda_u^{total} = \ell_u \cdot \lambda_u + (n-1) \cdot (1 - \ell_u) \cdot \lambda_u \cdot \frac{1}{n-1} + (n-1) \cdot r \cdot k \cdot ur \cdot \lambda_u$$
$$= (1 + (n-1) \cdot r \cdot k \cdot ur) \cdot \lambda_u$$

In addition to the rate of locally submitted updates (λ_u) the amount of propagated updates has to be included: An arbitrary update must be propagated with probability $r \cdot k \cdot ur$ at each of the n-1 remaining sites. The combined flow of arriving queries and updates at each local database system is still Poisson with rate

$$\lambda^{total} = \lambda_q^{total} + \lambda_u^{total} = \lambda_q + (1 + (n-1) \cdot r \cdot k \cdot ur) \cdot \lambda_u$$

Since the service time of the combined stream is hyperexponentially distributed and each node acts like a M/H$_2$/1/FCFS system the *average waiting time* $\overline{W}$ at a local database can be derived using the *Pollaczek-Khinchin formula* [24]:

$$\overline{W} = \frac{\lambda_q^{total} t_q^2 + \lambda_u^{total} t_u^2}{1 - \lambda_q^{total} t_q - \lambda_u^{total} t_u}$$

Using this result we can determine the *average response time for queries* which amounts to

$$\overline{R}_q = \ell_q \cdot (\overline{W} + t_q) + (1 - \ell_q) \cdot (t_n^{message} + \overline{W} + t_q + t_n^{data})$$

The first term of $\overline{R}_q$ corresponds to queries that can be answered locally and the second term covers the case that queries have to be forwarded to another site (taking $t_n^{message}$ seconds) requiring the results to be sent back (taking t_n^{data} seconds). Similarly, the *average response time for updates* happens to be

$$\overline{R}_u = \ell_u \cdot (\overline{W} + t_u) + (1 - \ell_u) \cdot (t_n^{message} + \overline{W} + t_u + t_n^{message}).$$