A Performance Analysis of K-Bound, a Consistency Protocol Supporting Multiple Isolation Levels

Raúl Salinas, Francesc D. Muñoz-Escoí

Instituto Tecnológico de Informática
Universidad Politécnica de Valencia
Campus de Vera s/n, 46022 Valencia (Spain)

{rsalinas,fmunyoz}@iti.upv.es

Technical Report ITI-ITE-08/01

A Performance Analysis of K-Bound, a Consistency Protocol Supporting Multiple Isolation Levels

Raúl Salinas, Francesc D. Muñoz-Escoí

Instituto Tecnológico de Informática
Universidad Politécnica de Valencia
Campus de Vera s/n, 46022 Valencia (Spain)

Technical Report ITI-ITE-08/01

e-mail: {rsalinas,fmunyoz}@iti.upv.es

May 7, 2008

Abstract

The construction and implementation of consistency protocols supporting multiple isolation levels in replicated databases has remained an unexplored field for a long time. Research in the last years has proved that the ability to use the most suited isolation level for every single transaction yields performance improvements and lower abort rates.

This paper takes advantage of our MADIS middleware and one of its implemented Snapshot Isolation protocols to design, implement and analyze the performance of K-Bound, a protocol that is able to concurrently execute Generalized Snapshot Isolation (GSI), optimistic Strict Snapshot Isolation (SSI), K-Bound —a non-standard SI level limiting the outdateness of transactions wanting to commit—, Serializable (using weak-voting), and Generalized Read Committed (GRC) transactions. The obtained performance results show that supporting multiple isolation levels does not introduce a significant overhead in transaction completion time. Moreover, the abortion rate of the relaxed levels is smaller than the strict ones. This allows an important abortion rate reduction if each transaction selects the most relaxed level than ensures its intended isolation.

Our paper is the first one that mixes transactions managed with a weak voting strategy with other ones using a certification-based technique. The experimental results show that weak voting management introduces performance penalties on certification-based transactions. Moreover, a first optimistic implementation of the strict snapshot isolation level is presented and its results are comparable to those obtained by fully serializable transactions, both in abortion rate and completion time.

1 Introduction

Database replication protocols have existed for years, and practically all of them have only supported a single isolation level. There are very few exceptions to this rule [2, 16]. However, if a database replication system supports more than one level, applications will be able to select the most appropriate isolation level for each transaction, as it can be done in centralized systems. In [16], we already implemented a protocol that supported *Snapshot Isolation* (SI) and *Read Committed* (RC) levels, proving that this reduces the abortion rate of those transactions that can be run using a relaxed isolation level like RC.

In this paper, we extend such an approach, implementing the ideas outlined in [2] as extensions of the protocol presented in [16]. Thus, the resulting protocol is able to support *full serializability* [1][1], an optimistic *strict snapshot* (SSI), a *generalized snapshot* (GSI) [6], and *generalized read committed* (GRC) [3], besides a k-bound [2] series of levels that are intermediate between SSI (k=0) and GSI (k=∞).

[1]The resulting level is not as strict as "full serializability", but is quite similar to it. It is based on adding the *SELECT FOR SHARE* clause on all queries.

The aims of this protocol (and, as we will see in the paper contents, its contributions) are: (a) to prove that supporting multiple isolation levels does not introduce a significant overhead on transactions' completion time, (b) to provide a first non-blocking implementation of the SSI level –although this raises its abortion rate, since such level demands to block transactions on their start in order to get an appropriate snapshot–, (c) to test the effects of a hybrid implementation of different isolation levels using different protocol families, since *full serializability* is implemented using a *weak-voting* [18] protocol whilst all other levels have been implemented with a *certification-based* [18] class, and (d) to extend tha main conclusions of [16], i.e., that relaxed isolation levels reduce abortion rates, to protocols supporting more than two levels. This leads to replication deployments that are able to compete with their centralized counterparts, since we have implemented all this support in our MADIS [9] middleware using PostgreSQL as its underlying DBMS. Note that PostgreSQL natively supports the RC and SI levels, but no other, although *full serializability*[2] could be "implemented" [7] following the same approach we have used in our protocol; i.e., extending queries with the SELECT FOR SHARE clause.

This paper is structured as follows: the system model being used in this paper is described in Section 2. In Section 3 the K-Bound protocol is presented. For building it, we took our SIRC with support for two isolation levels as the basis. An analysis of its performance in MADIS, i.e. the transaction response time and abortion rate, is shown in Section 5. Finally, conclusions end the paper.

2 System Model

The MADIS middleware [9] provides the necessary support to implement a suite of replication protocols in an interchangeable manner and serve as a testbed for them. It is developed in Java and has a JDBC interface to communicate with external applications, so that they will remain unaware of the replicated system. MADIS is also linked with the Spread [17] group communication system which provides a total order multicast [5]. We assume a fully replicated system composed of N replicas $(R_1, \ldots, R_N)$ where each replica has an underlying DBMS that stores a full physical copy of the database.; PostgreSQL [15] has been used to this end. This DBMS is a multiversion one (i.e. a new database version is generated each time a transaction is committed, we assume the version number is stored in a local replica variable called lastcommitted_tid) that locally supports the concurrent execution of RC and SI transactions. To keep consistent copies of the database a replication protocol is executed; in our case we assume that it follows the Read One Write All Available (ROWAA) approach [8]: a transaction is firstly executed at its delegate replica and at commit time its updates (denoted as *writeset*) are propagated to the rest of sites. A writeset is a list with the modified rows, having each one an associated *global object identifier* which is consistent throughout the cluster. Finally, MADIS includes a block detection mechanism [13] that greatly simplifies the application of remote writesets. It also improves the performance by earlier aborting local transactions having conflicts with validated transactions, even before the commit request.

3 Protocol Description

In addition to the levels supported by SIRC (GSI and GRC), the K-Bound protocol supports both a certification-based SI with different degrees of optimistic outdateness limitation (being $k = 0$ a particular case resulting in Strict Snapshot Isolation), and a SER using weak voting [4]. Despite the fact that all new levels have been bundled together, their implementation could have been made in a completely independent manner.

In Figure 1, the K-Bound [2] protocol is shown in pseudocode. Starting from the SIRC [16] protocol, we extended it as follows.

3.1 Supporting k-Bound SI

K-Bound SI is an isolation level defined in [2] that checks how many conflicts arise between the snapshot assigned to transaction T (that is local to its delegate node, and could be quite "old") and the writesets of

[2] We will refer to this isolation level as SER on the sequel.

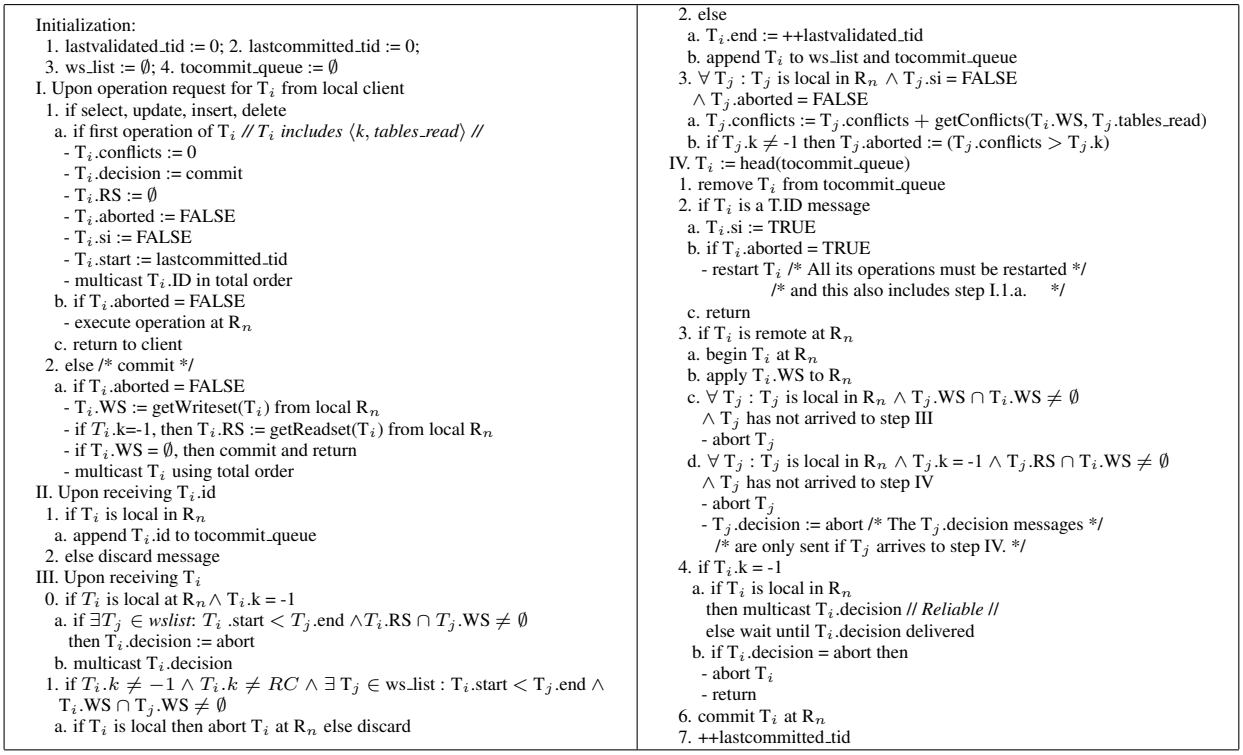

Figure 1: k-bound GSI algorithm at replica R_n

remote transactions that requested their commit in any system node prior to T 's start. So, each transaction needs to request its intended value for the k parameter; i.e., the amount of tolerated conflicting writesets. Thus, every time a new transaction is certified, the presence of conflicts between its writeset and the readset of every local transaction is checked. If conflicts are present, the conflict count for that local transaction is increased. If the resulting count exceeds its maximum acceptable k, the transaction is aborted. Note that a zero value for k means that such transaction is requesting a strict SI (SSI), whilst an infinite k value is like the GSI level defined in [6].

Different metrics can be used to measure the outdateness in K-Bound. In our case we have used the amount of committed writesets modifying the declared read tables. This implies that transactions must tell the middleware, when setting the isolation level, which tables they are about to read. A manual declaration of the readset is needed because this information has to be ready in advance, before the client starts working.

Whereas read-only transactions are never aborted in SI, this can in Kbound-SI indeed happen, if the transaction reads some data that were modified by an already validated transaction whose changes had not been applied by the time the former transaction took the snapshot at its delegate replica.

It is worth noting some implementation details of this isolation level:

- Before the actual execution of the first instruction of every k-bound transaction, a total-order broadcast (of a *T.ID* message, see last line in step I.1.a of Figure 1) is performed in order to note the version of the taken snapshot. After the broadcast, the operation is executed, and that is the moment when the snapshot is taken. The client can work without delay while this message is processed.

  When the delegate replica receives a *T.ID* message of its own, the message is enqueued into the *tocommit* list. The rest of the nodes ignore this message.

- Every time a transaction is validated, the delegate replica controls the conflicts between its writeset and the readsets of those k-bounded transactions in progress whose *T.ID* message has not been received yet, increasing the conflict count if needed. If such count reaches k, the associated local transaction is aborted.

- When processing a *T.ID* message from the *tocommit* queue for a transaction T, the transaction is marked, so that it will not be considered when evaluating conflicts with new arriving transactions.

  At this moment, the accumulated conflict count is the number of transactions T_i that satisfy that $T.start < T_i.tid$ and $T_i.writeSet.conflictsWith(T.readTables) = true$ (the number of transactions accessing our read tables that committed between the time we started and the time that pending certified transactions applied their writeset on the delegate replica). For $k = 0$ (SSI), a single conflict of this kind will result in an abortion.

  When a *T.ID* message is extracted from the *tocommit* queue, every transaction that had asked for commit before T started has now had its changes applied onto the database. The actual snapshot T is reading from is not the one it should have but the one that there was by the time T started working, and that is what the k-bounded SI bounds.

- Any updating k-bound transaction wanting to commit needs to wait for $T.si$ to become $true$. This wait could be non null in very short transactions.

3.2 Supporting SER

SER support has been built on top of SIRC's RC support, by adding a new restriction to it. Namely, checks take place in order to guarantee that no items read by a SER transaction (its *readset*, defined as a list of object identifiers) T_i have been modified by any concurrent transaction T_j (regardless of its level). This verification is performed by controlling the *recovered* items (not the *accessed* ones).

The SER level is implemented in our protocol with a weak-voting termination strategy. Since readsets are usually much bigger than writesets, weak voting saves the network and memory cost of broadcasting the readsets. Read-write conflicts are only checked at the delegate replica, using the object identifier lists.

The read-write conflict avoidance is enforced by two different mechanisms. The main one checks the readset of SER transactions against the writeset of concurrently committed transactions. In order to do that, the readset has to be extracted at commit time, along with the writeset. MADIS performs this task at the *ResultSet*. Queries are rewritten by the middleware so that every returned row includes a hidden column with the associated global object identifier. In spite of having been returned by a query, rows that have not been seen by the client code are not taken into account in order to construct the readset. This would be only important when trying to avoid predicate phenomena, which SER does not intend.

The second mechanism is actually an optimization meant to abort read-write conflicting transactions before they arrive to commit time, which is the moment when the readset is available. This is done by setting read locks on the read items. The middleware rewrites the queries so that SELECTs include the clause "FOR SHARE". This feature is implemented by most of the modern DBMSes. The semantics of PostgreSQL's "FOR SHARE" is the following: for each row returned by a query, a shared lock (read lock, compatible with other shared locks but incompatible with write, exclusive locks) is set. When a transaction T_i reads some object X, it gets a shared lock on it. If a concurrently running transaction T_j writes on this object X, at the moment that its commit time arrives at the delegate replica, the remote transaction will not be able to get a write lock on X, and therefore it will have to wait. The block detector will eventually detect a validated transaction waiting for a local transaction, and it will kill the local one. Apart from the intended consequence that the remote transaction T_j will be able to progress, some time is saved for T_i, which will have to abort anyway at commit time due to a read-write conflict.

When processing a broadcast SER transaction message, the delegate replica performs the aforementioned read-write conflict check by using the GOID lists. Then, it broadcasts the outcome reliably. Total order is not needed for this purpose.

When consuming a SER transaction from the *tocommit* list, replicas other than the delegate one will wait for the final decision from the delegate replica. The latter does not need to wait for its own broadcast message with the termination decision.

While the shared lock optimization mechanism is redundant and its absence would not affect the semantic properties of the consistency protocol, the GOID-based readset-writeset check is absolutely mandatory in a multiversioned DBMS such as PostgreSQL. Otherwise, when leaving the read-write control to the lock system, chances are that a transaction could read from an obsolete object, that was modified between the time the snapshot was taken and the time it actually accessed the object.

As always done in multilevel protocols supporting certification-based levels, the writesets of SER transactions are added to the *wslist*, so that certification-based transactions can be certified against the former ones.

4 Test conditions

Keeping the conflict rate –and thus the abortion rate– between reasonable limits is necessary to compare the performance of different replication protocols. Overloaded systems will yield unstable results, with an excessive variance to get any significative conclusions. In underloaded systems, the aborts related to conflicts will occur so seldom that most protocols will yield negligible time penalties.

In order to simplify the seek of a certain conflict rate we used the hotspot approach [11]. In this model, the entire database is split up into two sections: the hot spot and the low-conflicting area. Two parameters define the usage of the hotspot: the fraction of the total number of elements in the database and the fraction of the sentences that will access elements in this area of high concurrency access.

Before starting the tests, the database is populated. The total amount of items in the database has been fixed at 10000 rows.

At each node, a client is run, which uses the local server. Each client process launches a variable number of threads that try to satisfy a certain transaction rate (specified later for each experiment). If the queue of pending jobs grows beyond a certain threshold, the system is considered to be overloaded under the load being tested, and the experiment is aborted. The results taken are based upon a stationary regime. The time measurements of the first transactions are discarded, in order to dodge the initial glitch. PostgreSQL shows a remarkable worsening in the timings after the first transactions. In order to get minimally stable results, each experiment has run 10000 transactions in total.

The test performed was to run a series of "jobs", each job consisting of the following operations: (a) A certain number of reads on two randomly chosen tables, out of four available in total. (b) A 100 milliseconds wait. (c) A certain number of updates on two randomly chosen tables. After each write, a small wait (100 ms divided by the number of writes) is done.

Transactions are not retried when aborted.

In our tests we analyzed the following characteristics of the performance:

- Response time. Average amount of time needed for succeeding transactions to complete.
- Abort time. Average amount of time used by aborted transactions.
- Abort rate (0..1).

We kept the block detector poll interval at one second, which yields a reasonable response to deadlocks between the DBMS and the middleware while not overloading the DBMS.

5 Performance Results

We wanted to analyze the performance of this protocol when running transactions at the different isolation levels it supports. We were specially interested in finding out how this hybrid protocol, using both certification and weak-voting strategies, performs when serving a hybrid load.

We performed different experiments that are described in the following sections.

5.1 Varying the *k* Parameter in k-Bound SI

We studied the variation of the job completion time, the abort time and the abort rate as the k parameter varies. The TPS were fixed at 16, and the k varied between 0 (SSI) and 7.

In the three plots of Figure 2 we can quite clearly distinguish a threshold value of k (k_{thres}), from which GSI and k-Bound SI behave equally. This is strongly related to the average length of the tocommit queue. As that queue tends to get longer, the shift between the start moment of new transactions, and the actual snapshot they are getting becomes bigger. In an ideal, irrealistic case where local and remote transactions

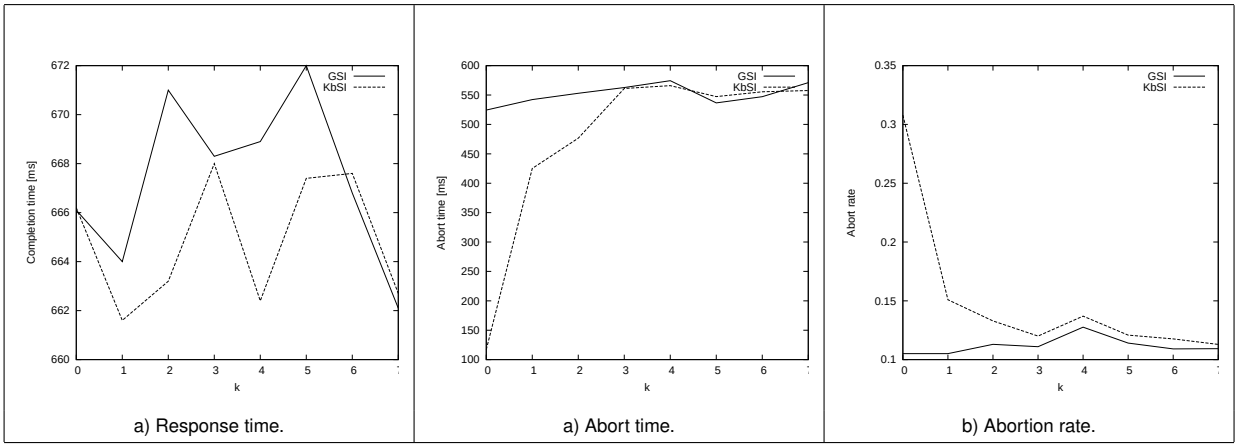

Figure 2: K-bound with different k values.

could be applied in a null period of time, the snapshot taken would always be the one wanted, and the k-bounding would make no difference whatsoever, except for the small overhead created by sending a total order broadcast, and by processing the message.

In the plot 2.a we can see that the completion time varies very little, between 662 ms and 672 ms, which is a negligible 1.51%, showing no difference between GSI and k-Bound SI. So, transactions that successfully finish do not exhibit timing differences.

Regarding plot 2.b we observe that the abort time in the k-Bound level is smaller than that of GSI transactions in the smallest values of k. Indeed, it is only 20% of GSI's one for $k = 0$; i.e., for SSI transactions. Note that in k-Bound level the abortion depends on the number of conflicting writesets that have been applied whilst a k-Bound transaction is still running (see steps III.3.a and III.3.b, in Fig. 1). Thus, with a single delivered conflicting writeset a 0-Bound SI transaction should abort, whilst in GSI such abortion is delayed until certification time; i.e., until the transaction to be aborted requests its commit and its writeset is broadcast and delivered in all replicas. Note that the MADIS block detector [13] can be tuned for aborting as soon as possible GSI transactions, but we have used it with a long interval in these tests in order to only ensure liveness, leaving abortion decisions to the replication protocol. As soon as the k values are increased, the k-Bound abort time is also increased until it gets values like those of GSI, once the k_{thres} value is reached. Such value has been 3 in the tested MADIS configuration with the load described above.

Finally, plot 2.c shows that the abortion rate decreases as the k increases. The maximum value is 30.5% for SSI whilst GSI achieved 11% with the same load and transaction sequence. However, once the k parameter exceeds its k_{thres} value, the abortion rate differences are minor than 2% between both isolation levels. Note that in this experiment we have used the same sequence of transactions for both isolation levels for each given k value, but that sequence was different between different k values.

5.2 Combining SER and GSI

In the second experiment, GSI and SER transactions were mixed. System load was set at 16 TPS again. Note that this also compares how a weak-voting (SER) and a certification-based (GSI) technique behave when they are combined.

Plot 3.a shows both completion and abortion time of transactions as the proportion of GSI and SER transactions varies. To begin with, we should look at the extreme values in the horizontal axis; i.e., using only GSI transactions (value 0) or only SER transactions (value 100). Thus, if the load is composed of only GSI transactions, their average completion time is 681 ms, whilst their average abortion time is 568 ms. On the other hand, with only SER transactions both times are quite bigger: 1171 ms as their completion time and 702 ms as their abortion time. This confirms the trend shown in [18], where certification-based protocols were able to provide the best completion time. However, when both kinds of transactions are mixed, the results are not so clear. Surprisingly, GSI transactions need more time than SER in both cases,

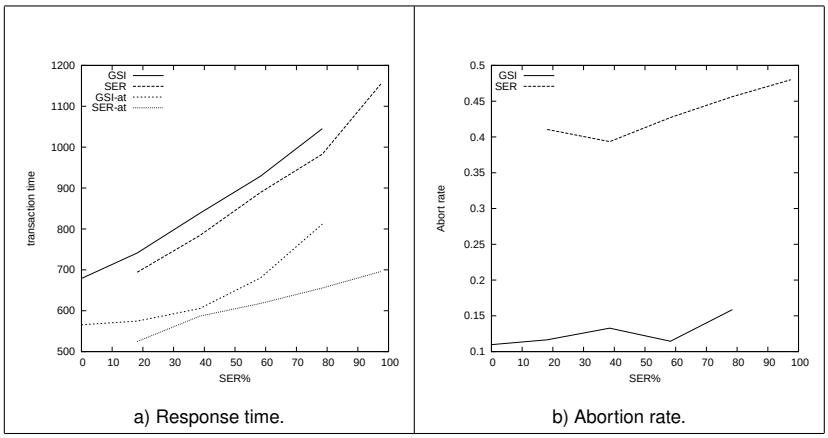

Figure 3: Combining SER and GSI levels.

abortion and successful completion. In general, weak-voting management (SER) needs to hold transactions for a longer time than a certification-based protocol, since they need to wait for a decision message that is broadcast and processed once the delegate replica has determined the fate of such transactions. In case of mixing transactions of both kinds, certification-based ones (in this case, GSI ones) need to wait until their preceding weak-voting ones have been completely managed. This implies that the protocol should know whether a SER transaction has committed or not before certifying any of its successors in the *tocommit* queue. This is a first factor to explain the SER-GSI behavior regarding completion time. Besides this, we should also consider that the SER level has an additional condition under which its transactions are aborted, that is namely read-write conflicts. So, the longer a SER transaction life is, the more probable it is that it will have a conflict with other transactions, being SER a level with more chances for conflict, the probability that a SER transaction will survive decreases as the transaction lifetime increases, so average completion times are shorter for SER. This constitutes a second factor for explaining the SER-GSI trends.

Regarding abortion rate, in plot 3.b we can see that the abort rate is higher for SER. This is what we expected to find, since although both types of transactions mixed in these experiments perform the same amount of random reads at the beginning, these reads are a source of conflict only in SER (this is done by means of adding the clause "FOR SHARE" to the SELECTs).

Considering again completion time, in order to discern which of the two mentioned factors is more important, we have repeated the previous experiment using write-only transactions; i.e., eliminating the effects of the second factor presented above. The results can be found in Figure 4. As we can see, when transactions do not read any item, the abort rate for SER transactions is similar to that of GSI ones, and as a result, their completion time gets also smaller than in Figure 3. As a result, the gap between GSI and SER in a given mix is very small (or even negative) in Figure 4 whilst it was close to 40 ms in all cases considered in Figure 3. Note, however, that the extreme values follow the same trend shown above; i.e., in the case of all SER transactions, their completion time (568 ms) is still bigger than that of the all-GSI-transactions one (542 ms), and this is explained by the first factor introduced above.

To sum up, running SER transactions –that use weak voting– introduces time penalties into the system, not only in SER transactions, but in every other transaction concurrently run. The execution time of the reliable broadcast notifying the transaction fate does not defer the commitment or abortion at the delegate replica. However, every other replica will have to wait for the message with the decision. This makes the transactions in the *tocommit* list to experience an accumulative delay that grows significatively as the proportion of serializable transactions increases.

5.3 Combining All Isolation Levels

In this experiment, transactions at every level offered by this protocol were concurrently run, in the same proportion. The transaction rate varied from 8 to 22 TPS. For the k-Bound SI transactions we chose the

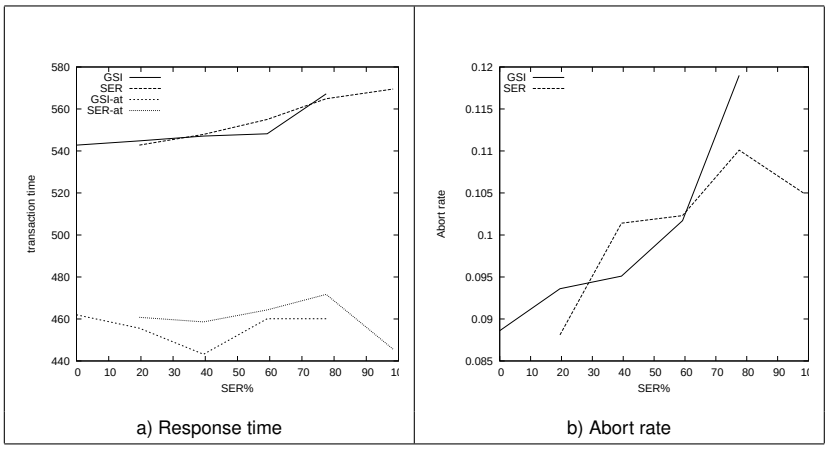

Figure 4: Combining SER and GSI (write-only transactions).

levels SSI ($k = 0$), $k = 1$ and $k = 2$. Because of the reasons explained about the k_{thres}, using a higher k would provide little useful information on the behavior of this protocol; i.e., its results would have been equal to those obtained with GSI.

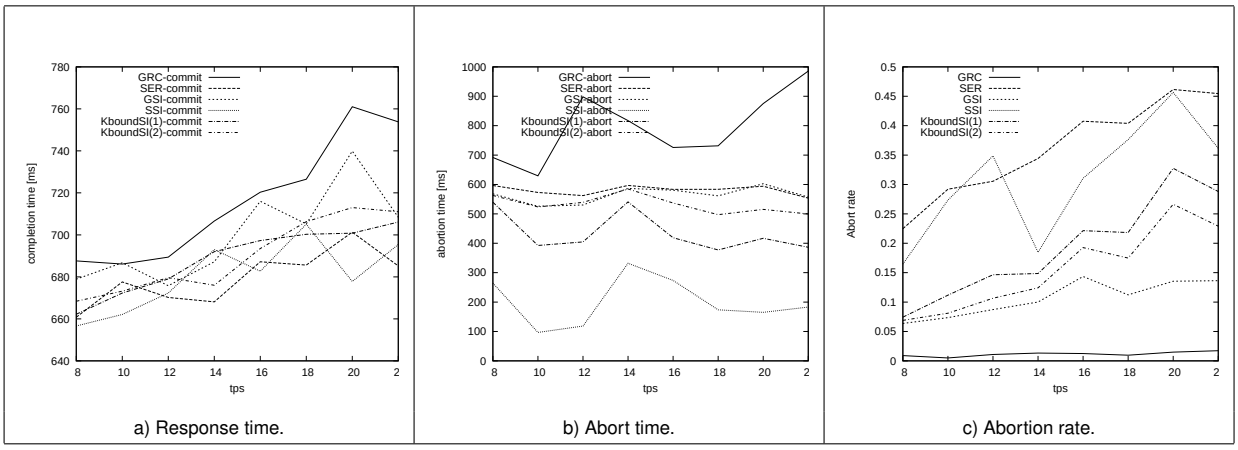

Figure 5: Combining transactions at all levels.

On the plot 5.a (completion time) we see that there are no great gaps between different levels.

In general, the trend is that SER and SSI provide the best results regarding completion time, and their differences are minimal. On a second group, we can find 1-Bound and 2-Bound SI with the GSI levels. Finally, GRC provides the worst results, since it demands local abortions and self-writeset application in order to ensure liveness, as discussed in [16]. The explanation for this behavior is also provided by the first factor described in Section 5.2 in order to explain the completion time in a SER-GSI mix: weak-voting levels (SER, in this paper) introduce a penalization in all certification-based levels (all the other). Additionally, the SER level is not affected by such penalization and artificially gets better results than all other levels, since it gets its results in its delegate replica without waiting for the decision message delivery, that penalizes all remaining transactions already present in its local *tocommit* queue. Differences among all other levels are not significant and are very dispersed, since their completion times mainly depend on its sequence number being assigned by the atomic broadcast protocol in the resulting total order. Some times, they are able to get the best places, some other, they will receive the worst ones, and their completion time always depends on their position in such total order.

Regarding abortion times (Figure 5.b), the shortest times appear in SSI (i.e., 0-Bound SI) transactions.

K-Bound SI transactions abort as soon as their *T.ID* message is taken from the tocommit queue and consumed, if conflicting writesets were committed in the meanwhile. Then comes 1-Bound SI and 2-Bound SI, confirming the trend already shown in Figure 2.b. Later GSI, and SER providing the expected mix between the results of Figures 2.b and 3.b. Finally, GRC provides the worst times, due to the same reason discussed for its completion time: transactions need to be locally aborted first, being certified later on; when such certification fails, the client is notified about their abortion, but this can not be done before.

Considering abortion rates (Figure 5.c), the experimental results confirm what was already presented in previous figures about SER, k-Bound SI, and GSI and in [16] regarding GRC. Moreover, the results show an increasing trend that directly depends on the load being supported, although such effect is bigger in some levels (e.g., in 2-Bound SI it reaches a 371% relative increase) than in other ones (e.g., in GRC). So, the strictest the isolation level is, the bigger its abortion level will be. As a result, SER generates the biggest abortion rate, ranging from 22% to 46%. SSI is close to it and more dispersed, ranging from 16% to 45.5%. In a second bunch, we can find 1-Bound SI (from 7.5% to 32.5%), 2-Bound SI (from 7% to 26%) and GSI (from 6.5% to 14%). Finally, the abortion rates of GRC range from 0.8% to 2%.

5.4 Combining All Levels without SER

Let us now show the effects of removing the SER level in the experiments described in Section 5.3. To this end, the same loads used in such experiments are re-used here, but distributing them among five isolation levels instead of six.

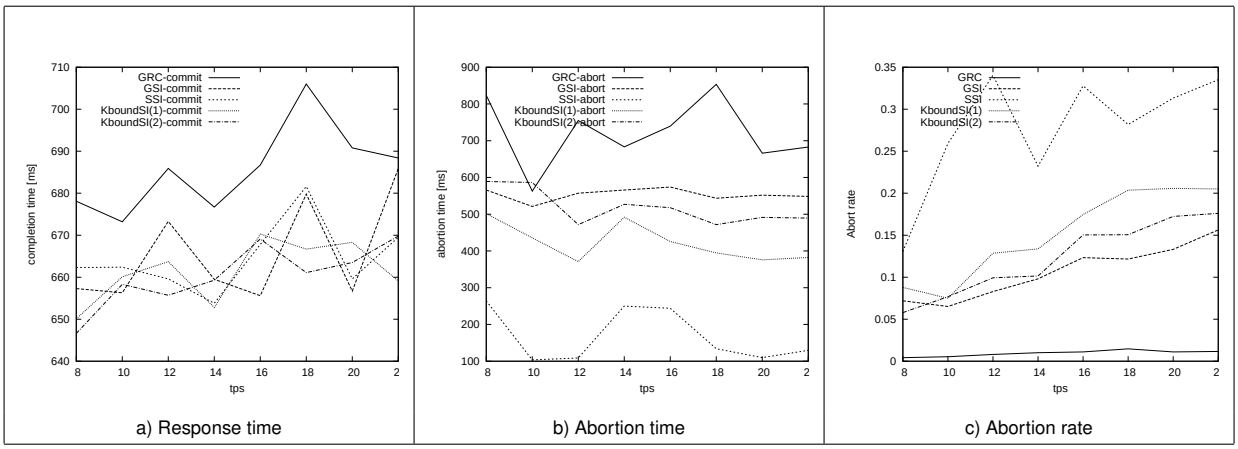

Figure 6: Combining transactions at all levels, without SER.

Figure 6.a plots the commit time for such isolation levels. The trend is almost equivalent to that shown in Figure 5.a; i.e., GRC obtains the worst times, but all other levels can be bunched together since no clear winner can be found. Recall that such variation depends on the ordering being set by the group communication service, and its results are random. Note however that an important conclusion can be extracted from these results once they are compared with those of Figure 5.a: in that figure the times for non-GRC levels ranged from 656 ms to 740 ms, whilst in this figure the values are comprised in the range 646 ms through 686 ms; i.e., they are a bit better. This shows again that weak voting techniques penalize the results of certification-based ones, as it already happened in Fig. 3.a.

Regarding abortion time (Figure 6.b), the lowest times (i.e., best results) correspond to the most restrictive isolation level (SSI) and the worst ones to the most relaxed one (GRC). This can be easily explained due to the evaluation conditions being used in each level, as already detailed above. No significant gain can be observed in this case, comparing its results with those of Figure 5.b. Note that abortion times depend only on the arrival rate for conflicting transactions, and this does not depend on the way each protocol accepts transactions for committing; i.e., the criteria and steps being used for abortion is the same in all levels, whilst the criteria and algorithm for accepting transactions was different between SER and all other levels.

Finally, the abortion rates shown in Figure 6.c follow the same trends already observed in Figure 5.c. All isolation levels can be ordered in the same way: the most relaxed the isolation level is, the lowest abortion rate can be obtained. A single difference can be found: all abortion rates are slightly lower in this case.

5.5 Recapitulation

We have implemented and tested a consistency protocol that provides different isolation levels, allowing each single transaction to request the required isolation guarantees, depending on the intended kind of work. Whereas certification-based transactions running at different isolation levels can coexist without interference (different certification rules are used for each level), the introduction of weak-voting transactions (offering the SER level) strongly hinders the overall performance. Manual application of the techniques to prevent serialization anomalies [7], or using automatic procedures [10] will certainly yield a better performance.

The snapshot outdateness limitation given in the k-Bound SI levels makes the abort rate much higher, especially in highly loaded systems. Being able to declare the readset with a finer resolution would reduce such abort rates. The extreme case consists in using a 0-Bound SI level; i.e., a strict SI level that provides, up to our knowledge, the first implementation of a practical strict SI replication protocol in a middleware system, providing similar performance results to those of a *fully serializable* [1] (or SER) level. Such SER level differs to those reported in other papers (e.g., [12, 14]) in checking also for write-write conflicts, instead of only for read-write ones.

6 Conclusions

In this paper we have analyzed the performance of k-Bound, implemented as an extension of the SIRC protocol, originally suited for executing GSI and GRC transactions. By supporting a variant of Snapshot Isolation called k-Bound SI –which establishes a limit on the outdateness on the snapshot taken by transactions wanting to commit– and a weak-voting serializable, k-Bound shows that it is possible to design a replication protocol supporting multiple isolation levels using different transaction termination approaches (both certification and weak voting).

The performance analysis has shown that SER transactions (using weak-voting and therefore needing a final message broadcast from the delegate replica with the transaction outcome) slow down the whole database system remarkably, affecting not only weak-voting transactions but also certification-based ones. Our experiments conclude that weak voting scales worse than certification.

The k-Bound protocol offers Strict Snapshot Isolation. Despite its convenience for applications sensitive to their read updateness, care must be taken when using k-Bound SI, since the coarse granularity worked with (table level) severely hinders the abort rate. In addition to that, k-Bound SI requires explicit declaration of the tables to be read.

The overall results show that multiple isolation levels can be supported at once in a single replication protocol. This does not compromise the overall performance. Additionally, application programmers can select the most appropriate isolation level for each transaction, reducing the abortion rate of those transactions that can now be managed with relaxed levels.

References

[1] A. Adya. *Weak Consistency: A Generalized Theory and Optimistic Implementations of Distributed Transactions*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, March 1999.

[2] J. E. Armendáriz-Íñigo, J. R. Juárez-Rodríguez, J. R. González de Mendívil, H. Decker, and F. D. Muñoz-Escoí. k-Bound GSI: A flexible database replication protocol. In *SAC*. ACM, 2007.

[3] J. M. Bernabé-Gisbert, J. E. Armendáriz-Íñigo, R. de Juan-Marín, and F. D. Muñoz-Escoí. Providing read committed isolation level in non-blocking ROWA database replication protocols. In *JCSD*, 2007.

[4] J. M. Bernabé-Gisbert, R. Salinas-Monteagudo, L. Irún-Briz, and F. D. Muñoz-Escoí. Managing multiple isolation levels in middleware database replication protocols. In *ISPA*, LNCS. Springer, 2006.

[5] Gregory Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[6] S. Elnikety, F. Pedone, and W. Zwaenepoel. Database replication providing generalized snapshot isolation. In *SRDS*, 2005.

[7] Alan Fekete, Dimitrios Liarokapis, Elizabeth O'Neil, Patrick O'Neil, and Dennis Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2):492–528, 2005.

[8] Jim Gray, Pat Helland, Patrick E. O'Neil, and Dennis Shasha. The dangers of replication and a solution. In *SIGMOD*, 1996.

[9] L. Irún, H. Decker, R. de Juan, F. Castro, J. E. Armendáriz, and F. D. Muñoz. MADIS: a slim middleware for database replication. In *Euro-Par.*, LNCS. Springer, 2005.

[10] Sudhir Jorwekar, Alan Fekete, Krithi Ramamritham, and S. Sudarshan. Automating the detection of snapshot isolation anomalies. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 1263–1274. VLDB Endowment, 2007.

[11] B. Kemme. *Database Replication for Clusters of Workstations*. PhD thesis, Swiss Federal Institute of Technology, Zürich, Switzerland, August 2000.

[12] Bettina Kemme and Gustavo Alonso. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In *VLDB*, pages 134–143, 2000.

[13] F. D. Muñoz-Escoí, J. Pla-Civera, M. I. Ruiz-Fuertes, L. Irún-Briz, H. Decker, J. E. Armendáriz-Íñigo, and J. R. González de Mendívil. Managing transaction conflicts in middleware-based database replication architectures. In *SRDS*, 2006.

[14] F. Pedone. *The Database State Machine and Group Communication Issues*. PhD thesis, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 1999.

[15] PostgreSQL. PostgreSQL, the world's most advanced open source database. `http://www.postgresql.org`, 2008.

[16] R. Salinas, J.M. Bernabé-Gisbert, and F.D. Muñoz-Escoí. SIRC, a multiple isolation level protocol for middleware-based data replication. In *22nd International Symposium on Computer Information Sciences*, Ankara, Turkey, November 2007. IEEE-CS Press.

[17] Spread. The Spread toolkit. `http://www.spread.org`, 2007.

[18] Matthias Wiesmann and André Schiper. Comparison of database replication techniques based on total order broadcast. *IEEE TKDE*, 17(4):551–566, 2005.