[Pap86] C. Papadimitriou. *The Theory of Database Concurrency Control*. Computer Science Pr Rockville, Maryland, 1986.

[PL91] C. Pu and A. Leff. Replica control in distributed systems: An asynchronous appro In *Proceedings of ACM-SIGMOD 1991 International Conference on Management of D Denver, Colorado*, pages 377–386, May 1991.

[PRR91] W. Perrizo, J. Rajkumar, and P. Ram. Hydro: A heterogeneous distributed datal system. In *Proceedings of ACM-SIGMOD 1991 International Conference on Managem of Data, Denver, Colorado*, pages 32–39, May 1991.

[Pu88] C. Pu. Superdatabases for composition of heterogeneous databases. In *Proceedings of Fourth International Conference on Data Engineering, Los Angeles*, 1988.

[Raz92] Y. Raz. The principle of atomic commitment ordering, or guaranteeing serializab in a heterogeneous environment of multiple autonomous resource managers using ato commitment. In *Proceedings of the eighteenth International Conference on Very L Databases, Vancouver*, 1992.

[RKS92] R. Rastogi, H. F. Korth, and A. Silberschatz. Relaxed concurrency control in multidatal systems. Technical Report TR-92-45, Department of Computer Science, University of Te at Austin, 1992.

[SKS91] N. R. Soparkar, H.F. Korth, and A. Silberschatz. Failure-resilient transaction managem in multidatabases. *IEEE Computer*, December 1991.

[WA92] M. H. Wong and D. Agrawal. Tolerating bounded inconsistency for increasing concurre in database systems. In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIG Symposium on Principles of Database Systems, San Diego*, pages 236–245, June 1992.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recover Database Systems*. Addison-Wesley, Reading, MA, 1987.

[BS88] Y. Breitbart and A. Silberschatz. Multidatabase update issues. In *Proceedings of A SIGMOD 1988 International Conference on Management of Data, Chicago*, pages 135– 1988.

[BST90] Y. Breitbart, A. Silberschatz, and G. R. Thompson. Reliable transaction managemen a multidatabase system. In *Proceedings of ACM-SIGMOD 1990 International Confere on Management of Data, Atlantic City, New Jersey*, pages 215–224, 1990.

[DE89] W. Du and A. K. Elmagarmid. Quasi serializability: a correctness criterion for gl concurrency control in InterBase. In *Proceedings of the Fifteenth International Confere on Very Large Databases, Amsterdam*, pages 347–355, 1989.

[ED90] A.K. Elmagarmid and W. Du. A paradigm for concurrency control in heterogeneous tributed database systems. In *Proceedings of the Sixth International Conference on I Engineering*, 1990.

[FO89] A. A. Farrag and M. T. Ozsu. Using semantic knowledge of transactions to increase currency. *ACM Transactions on Database Systems*, 14(4):503–525, December 1989.

[GLPT75] J. N. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger. Granularity of locks and deg of consistency in a shared data base. In *IFIP Working Conference on Modeling of I Base Management Systems*, pages 1–29, 1975.

[GM83] H. Garcia-Molina. Using semantic knowledge for transaction processing in a distribu database. *ACM Transactions on Database Systems*, 8(2):186–213, June 1983.

[GMS87] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of ACM-SIGMOD 1987 Inte tional Conference on Management of Data, San Francisco*, pages 249–259, 1987.

[GRS91] D. Georgakopolous, M. Rusinkiewicz, and A. Sheth. On serializability of multidata transactions through forced local conflicts. In *Proceedings of the Seventh Internati Conference on Data Engineering, Kobe, Japan*, 1991.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and C putation*. Addison-Wesley, Reading, MA, 1979.

[KB91] N. Krishnakumar and A. J. Bernstein. Bounded ignorance in replicated systems. In *Proc ings of the tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Data Systems, Denver*, pages 63–74, May 1991.

[KKB88] H. F. Korth, W. Kim, and F. Bancilhon. On long duration CAD transactions. *Informa Sciences*, 46:73–107, October 1988.

[MRB+92] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz. The concurre control problem in multidatabases: Characteristics and solutions. In *Proceedings of A SIGMOD 1992 International Conference on Management of Data, San Diego, Califor* 1992.

[MRKS91] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz. Non-serializable executi

Definition 8: Let $RT = e_0 : reg\_exp$ be a regular term containing only elements with arity 2 let $(G_0, s_{i_0})(s_{i_0}, G_1) \cdots (G_n, s_{i_{n-1}})(s_{i_{n-1}}, G_0)$, $n > 1$, be a strong-cycle in a TSGD. The strong-c $(G_0, s_{i_0}) \cdots (s_{i_{n-1}}, G_0)$ is said to be consistent with respect to RT iff

- $e_0 = type(G_0 : G_{0i_{n-1}}, G_{0i_0})$, and
- $type(G_1 : G_{1i_0}, G_{1i_1}) \cdots type(G_{n-1} : G_{(n-1)i_{n-2}}, G_{(n-1)i_{n-1}})$ is a string in $L(reg\_exp)$.

A TSGD is said to be strongly-acyclic with respect to a regular specification R iff for every $RT \in$ it does not contain a strong-cycle consistent with respect to RT. □

The problem of determining if the invariant holds can be shown to be NP-complete as a conseque of the following NP-completeness result.

Theorem 8: The following problem is NP-complete: Given a TSGD (V, E, D, L), such tha is consistent, a regular specification R containing only elements with arity 2, does there exist a se dependencies Δ such that $D \cup \Delta$ is consistent, and the TSGD $(V, E, D \cup \Delta, L)$ is strongly-acyclic w respect to R?

Proof: See Appendix E. □

Note that, in an execution, at any instant, the invariant holds if and only if at that instant, in TSGD (V, E, D, L), there exists a set of dependencies Δ such that $(V, E, D \cup \Delta, L)$ is strongly acy with respect to R (since every element in R has arity 2, the TSGD is strongly acyclic with respec R if and only if no instantiations of regular terms in R can result in S). Thus, from Theorem 8 follows that determining if the invariant holds is NP-complete.

9 Conclusion

In an MDBS environment, based on the semantics of transactions, certain non-serializable executi are acceptable. In this paper, we proposed a simple and powerful mechanism for specifying, in MDBS environment, the set of non-serializable executions that are unacceptable. The undesir interleavings among global subtransactions are specified using regular expressions over the type global subtransactions. We showed that using our mechanism, it is possible to characterize interle ings that cannot be captured by existing mechanisms for specifying interleavings. Also, unlike exis approaches, our approach scales well to the addition of new global applications in the system. We de oped efficient graph-based schemes (optimistic and conservative) in order to ensure that the concur execution of transactions meet specifications. In MDBS environments in which certain non-serializ executions are acceptable, we expect our schemes to outperform existing schemes for ensuring glo serializability. We showed that although none of the conservative schemes proposed by us permit timal concurrency, the problem of optimally scheduling operations for execution is NP-complete. are currently investigating recovery algorithms that can be used with our schemes to deal with failures and transaction aborts, alternative mechanisms for specifying interleavings, and distribu concurrency control schemes for preventing unacceptable interleavings. The results in this paper also applicable to homogeneous distributed database systems.

References

[BGRS91] Y. Breitbart, D. Georgakopolous, M. Rusinkiewicz, and A. Silberschatz. On rigorous tr

Definition 6: Consider a TSGD containing the sequence of edges $(G_0, s_{i_0})(s_{i_0}, G_1)\cdots(G_{n-1}, s_{i_{n-1}})$ $(s_{i_{n-1}}, G_0)$, $n > 1$. This sequence of edges form a *strong-cycle* if

- for all $j = 0, 1, \ldots, n-1$, $G_j \neq G_{(j+1)\bmod n}$, $s_{i_j} \neq s_{i_{(j+1)\bmod n}}$ and dependency (G_j, s_{i_j}) $(s_{i_j}, G_{(j+1)\bmod n}) \notin D$.
- $D \cup \{(G_{(j+1)\bmod n}, s_{i_j}) \rightarrow (s_{i_j}, G_j) : 0 \leq j \leq n-1\}$ is consistent.

A TSGD is said to be *strongly-acyclic* if it does not contain any strong-cycles. □

Strong-minimality is defined in terms of strong-cycles as follows.

Definition 7: A set of dependencies Δ is *strongly-minimal* with respect to the TSGD an transaction $G_i \in V$ iff

- $(V, E, D \cup \Delta, L)$ does not contain any strong-cycles involving G_i, and
- for all $d \in \Delta$, $(V, E, D \cup \Delta - d, L)$ contains a strong-cycle involving G_i. □

The computation of a minimal Δ can be shown to be NP-hard as a consequence of the follov NP-hardness result.

Theorem 7: Given a TSGD (V, E, D, L) such that D is consistent, and a transaction n $G_i \in V$ such that for all transactions $G_j \in V$, for all sites s_k, dependency $(G_i, s_k) \rightarrow (s_k, G_j) \notin D$. A TSGD (V', E', D', L') resulting due to the deletion of G_i, its edges and dependencies from $(V, E, D$ is strongly-acyclic. The problem of computing a set of dependencies, Δ, such that $D \cup \Delta$ is consist and Δ is strongly-minimal with respect to the TSGD and transaction G_i is NP-hard.

Proof: See Appendix E. □

If the regular specification were to contain the single regular term $RT = (A : a, a) : (A : a, a$ every global transaction were to have type A, and one or more subtransactions of type a, then a mini Δ would also be strongly-minimal with respect to the TSGD and G_i (since an instantiation of R S involving G_i could result if and only if there is a strong-cycle in the TSGD containing G_i). T from Theorem 7, it follows that the computation of a minimal Δ is NP-hard.

Also, in the TSGD scheme presented in the previous section, the set of dependencies in orde prevent instantiations of regular terms, are computed when an $init_i$ operation is processed. Howe this approach that involves restricting the execution of $ser_k(G_i)$ operations *a priori* (when *ini* processed) is inflexible and could result in a low degree of concurrency. An alternative conserva scheme would be one that does not impose restrictions on the processing of $ser_k(G_i)$ operations w an $init_i$ operation is processed, but instead, ensures that at any instant the following invariant ho there exists a total order on unexecuted $ser_k(G_i)$ operations such that executing them in the o consistent with the total order does not result in instantiations of regular terms in S. The invariant sures that all the unexecuted $ser_k(G_i)$ operations can be executed without jeopardizing the correct of S and without aborting any global transactions. Furthermore, a $ser_k(G_i)$ operation is permitte execute if and only if its execution preserves the invariant (note that the processing of $init_i$ operat trivially preserve the invariant). Thus, the alternative approach would provide the maximum de of concurrency that can be provided by a conservative scheme. However, it can be shown that, at instant, determining if the invariant holds is an NP-complete problem. We begin by showing a rel

for every pair of nodes (w, u) such that (w, v') and (u, v') are both edges in the TSGD, (w, u) be
appended to $anc(v')$ when v' is visited in state st.

The dependencies added to the TSGD during the processing of an $init_i$ operation ensure that instantiations of regular terms involving global transaction G_i are possible. Thus, it can be shown b simple induction argument on the number of $init_i$ operations processed that there are no instantiat of any of the regular terms in global schedule S.

Theorem 5: Let R be a complete regular specification. The TSGD scheme ensures that correct with respect to R.

Proof: See Appendix D. □

The complexity of the TSGD scheme is dominated by the number of steps it takes to process an i operation. Detect_Ins_TSGD1 can be shown to terminate in $O(n_S n_G^2 m)$ steps and Detect_Ins_TS(can be shown to terminate in $O(n_S n_G^3 m)$ steps. Since, in the worst case, Detect_Ins_TSGD? is invo for every regular term in R and for every subtransaction of G_i, the complexity of the TSGD schem as stated in the following theorem.

Theorem 6: The worst-case complexity of the TSGD scheme is $O(n_S n_G^2 m n_R v_S)$ if Detect_Ins_T is Detect_Ins_TSGD1 and is $O(n_S n_G^3 m n_R v_S)$ if Detect_Ins_TSGD? is Detect_Ins_TSGD2. □

Among the conservative schemes presented in the last two sections, the TSGD scheme with tect_Ins_TSGD2 provides the highest degree of concurrency, but also has the highest complexity. TSG scheme with Detect_Ins_TSG1 has the lowest complexity among all the schemes, but also perr the lowest degree of concurrency. The TSGD scheme with Detect_Ins_TSGD1 and the TSG sch with Detect_Ins_TSG2 have identical complexities, but the degree of concurrency provided by the schemes is incomparable.

8 Intractability Results

In the TSGD scheme in the previous section, the set of dependencies Δ computed during the proces of an $init_i$ operation ensures that there will be no instantiations of regular terms in S involving gl transaction G_i. However, a number of the restrictions imposed on the processing of $ser_k(G_i)$ op tions due to the dependencies in Δ may be unnecessary; that is, there may exist a set of dependen $\Delta' \subset \Delta$ such that adding Δ' to the TSGD prevents instantiations of regular terms in S involving Let us refer to a set of dependencies Δ as *minimal* if Δ ensures that there will be no instantiat of regular terms in S involving G_i, while for any $\Delta' \subset \Delta$, adding Δ' to the TSGD does not prev such instantiations. Thus, ideally, in order to impose minimal restrictions on the execution of $ser_k($ operations, the set of dependencies Δ computed when an $init_i$ operation is processed must be mini However, the computation of such a minimal Δ is NP-hard. In order to show this, we need to define *strong-minimality* for which we need the following additional definitions.

Definition 5: A set of dependencies D is consistent, if there do not exist nodes $v_0, v_1, \ldots, v$ $n > 2$, in the TSGD such that $(v_1, v_0) \rightarrow (v_0, v_2) \in D$, $(v_2, v_0) \rightarrow (v_0, v_3) \in D$, ..., $(v_{n-1}, v_0) \rightarrow (v_0, v_1) \in$
□

In addition, we need to define the notion of a *strong-cycle*.

G_i and its edges are inserted into the TSGD. For every operation $ser_k(G_i) \in G_i$, for all transactior $G_j \in V$ such that $ser_k(G_j) \in G_j$ and $act(ser_k(G_j))$ has executed, dependencies $(G_j, s_k) \rightarrow (s_k, G$ are added to D.

```
Δ := ∅;
for every regular term RT = e_0 : reg_exp in R such that type(G_i) = hdr(e_0) do
    for every subtransaction G_ik such that type(G_ik) = last(e_0) do
        begin
        if arity(e_0) = 1 then  set_1 := {s_k}
                        else set_1 := {s_l : (s_l ≠ s_k) ∧ (type(G_il) = first(e_0))};
         Δ := Δ∪ Detect_Ins_TSGD?((V, E, D ∪ Δ, L), G_i, s_k, set_1, RT)
        end;
D := D ∪ Δ
```

Figure 9: Pseudocode for $act(init_i)$

- ***act***$(\boldsymbol{ser}_k(\boldsymbol{G}_i))$: For every transaction $G_j \in V$ such that $ser_k(G_j) \in G_j$ and $act(ser_k(G_j))$ not yet been executed, dependencies $(G_i, s_k) \rightarrow (s_k, G_j)$ are added to D. Operation $ser_k(G$ submitted to the local DBMSs (through the servers) for execution.

- ***cond***$(\boldsymbol{ack}(\boldsymbol{ser}_k(\boldsymbol{G}_i)))$: *true*.

- ***act***$(\boldsymbol{ack}(\boldsymbol{ser}_k(\boldsymbol{G}_i)))$: Operation $ack(ser_k(G_i))$ is sent to GTM_1.

- ***cond***$(\boldsymbol{fin}_i)$: *true*.

- ***act***$(\boldsymbol{fin}_i)$: For every transaction $G_j \in V$ such that val_j has been processed, if for every tran tion $G_k \in V$ serialized before G_j, val_k has been processed, then G_j along with all its edges dependencies is deleted from the TSGD.

Procedures Detect_Ins_TSGD1 and Detect_Ins_TSGD2 traverse edges in the TSGD in order to tect potential instantiations, and are very similar to procedures Detect_Ins_TSG1 and Detect_Ins_TS respectively. Detect_Ins_TSGD1 and Detect_Ins_TSGD2 are similar to Detect_Ins_TSG1 and tect_Ins_TSG2, in that they may detect false instantiations (Detect_Ins_TSGD1 detects more f instantiations than Detect_Ins_TSGD2). However, instead of returning a set of site nodes, they ret a set of dependencies that if added to the TSGD, restricts the execution of the appropriate $ser_k($ operations so that there are no instantiations involving G_i.

The updates to $anc(v')$ and $V\_set(v')$ when a node v' is visited are the same in both Detect_Ins_TS and Detect_Ins_TSGD1. One of the main differences between the two schemes is that for the seque of edges $(v_0, u_0), (v_1, u_1), \ldots, (v_p, u_p)$ traversed as mentioned earlier, Detect_Ins_TSGD1 ensures for all $i = 1, 2, \ldots, p$, (v_i, u_i) is distinct from (v_0, u_0) (unlike Detect_Ins_TSG1, which only ensures (v_0, u_0) and (v_p, u_p) are distinct). Furthermore, due to the presence of dependencies in the TSGD, due to conditions in steps 3(a) and 3(b), for any state st of F, every node v' in the TSGD may nee be visited in state st once for each node w such that (v', w) is an edge in the TSGD, w being appen to $anc(v')$ when v' is visited in state st.

Detect_Ins_TSGD2, too, updates $anc(v')$ and $V\_set(v')$, when a node v' is visited, in a m ner identical to Detect_Ins_TSG2, and like Detect_Ins_TSG2, ensures that for the sequence of ed $(v_0, u_0), (v_1, u_1), \ldots, (v_p, u_p)$, for all $i = 1, 2, \ldots, p$, (v_i, u_i) is distinct from both (v_{i-1}, u_{i-1}) and $(v_0,$ Also, due to the presence of dependencies in the TSGD, and due to conditions in steps 3(a), 3(b),

in the same state st multiple times, each time the same node u being appended to $anc(v')$, $(st, u$
added to $V\_set(v')$ when v' is visited in state st and u is appended to $anc(v')$.

Similarly, in Detect_Ins_TSG2, since the TSG does not contain any dependencies, and due to first two conditions in Step 3, in order to detect instantiations, for any state st of F, every node v the TSG may need to be visited in state st twice for each node w such that (v', w) is an edge in TSG, the pairs (w, u_1) and (w, u_2) appended to $anc(v')$ the two times v' is visited in state st be distinct. Also, in order to prevent a node v' from being visited in the same state st multiple tim each time the same pair (w, u) being appended to $anc(v')$, $(st, (w, u))$ is added to $V\_set(v')$ when visited in state st and (w, u) is appended to $anc(v')$.

When an $init_i$ operation is processed, in order to prevent instantiations involving transaction the TSG scheme restricts the execution of certain $ser_k(G_i)$ operations by marking them (processin marked operations is delayed until all the operations ahead of it in the queue have been process Thus, by a simple induction argument on the number of $init_i$ operations processed, it can be sho that there are no instantiations of any of the regular terms in global schedule S involving any of global transactions.

Theorem 3: Let R be a complete regular specification. The TSG scheme ensures that S is cor with respect to R.

Proof: See Appendix C. □

The complexity of the TSG scheme is dominated by the number of steps it takes to process an i operation. Procedures Detect_Ins_TSG1 and Detect_Ins_TSG2 can be shown to terminate in $O(n_S n_G$ and $O(n_S n_G^2 m)$ steps respectively. Since, in the worst case, when $init_i$ is processed, Detect_Ins_TS is invoked for every regular term in R and for every subtransaction of G_i, the complexity of the T scheme is as stated in the following theorem.

Theorem 4: The worst-case complexity of the TSG scheme is $O(n_S n_G m n_R v_S)$ if Detect_Ins_TS is Detect_Ins_TSG1 and is $O(n_S n_G^2 m n_R v_S)$ if Detect_Ins_TSG? is Detect_Ins_TSG2. □

7 Conservative Schemes with Dependencies

The conservative schemes described in the previous section do not exploit the knowledge of the ser ization order of global subtransactions since they utilize the TSG as a data structure. In this sect we present conservative schemes that employ the TSGD as a data structure to store the execution o of $ser_k(G_i)$ operations and thus, provide a higher degree of concurrency than the schemes descri in the previous section. In the schemes, if potential instantiations of regular terms involving G_i detected when the edges of the TSGD are traversed during the processing of an $init_i$ operation, t appropriate dependencies are added to the TSGD in order to restrict the processing of certain $ser_k($ operations. We now specify, for every operation o_j in QUEUE, $cond(o_j)$ and $act(o_j)$ (no actions performed when a val_i operation is processed, and $cond(val_i) = true$). Initially, for the TSGD, V $E = \emptyset$, $D = \emptyset$.

- ***cond*(*init*$_i$)**: $true$.
- ***act*(*init*$_i$)**: The pseudocode in Figure 9 is executed. Procedure Detect_Ins_TSGD? can be ther Detect_Ins_TSGD1 (see Figure 13 in Appendix A) or Detect_Ins_TSGD2 (see Figure 1 Appendix A).

G_i and its edges are inserted into the TSG. Also, for every operation $ser_k(G_i) \in G_i$, $ser_k(G_i)$
inserted at the end of the queue for site s_k.

$set_2 := \emptyset$;
for every regular term $RT = e_0 : reg\_exp$ in R such that $type(G_i) = hdr(e_0)$ **do**
  for every subtransaction G_{ik} such that $type(G_{ik}) = last(e_0)$ **do**
    begin
    if $arity(e_0) = 1$ **then** $set_1 := \{s_k\}$
        else $set_1 := \{s_l : (s_l \neq s_k) \wedge (type(G_{il}) = first(e_0))\}$;
    $set_2 := set_2 \cup$ Detect_Ins_TSG?$((V, E, L), G_i, s_k, set_1, set_2, RT)$
    end
For every site s_l in set_2, $ser_l(G_i)$ is marked in the queue for site s_l.

Figure 8: Pseudocode for $act(init_i)$

- ***act*($\boldsymbol{fin}_i$)**: For every transaction $G_j \in V$ such that val_j has been processed, if for every tran tion $G_k \in V$ such that there is a path from G_j to G_k in the TSG, val_k has been processed, t G_j, along with all its edges, is deleted from the TSG.

Procedures Detect_Ins_TSG1 and Detect_Ins_TSG2 traverse edges in the TSG in order to de potential instantiations in a similar fashion as Detect_Ins_Opt. However, unlike Detect_Ins_Opt wh returns commit/abort, they return a set of site nodes that identify $ser_k(G_i)$ operations whose e cution, if restricted, would prevent instantiations of regular terms. Both Detect_Ins_TSG1 and tect_Ins_TSG2 may detect false instantiations and thus require the execution of more operations to restricted than are actually required to prevent instantiations (Detect_Ins_TSG1 detects a larger n ber of false instantiations than Detect_Ins_TSG2, but has a lower complexity than Detect_Ins_TSG Also, in Detect_Ins_TSG1 and Detect_Ins_TSG2, since the TSG contains no dependencies, when a n v' is visited, the nodes appended to $anc(v')$ are different from those appended in Detect_Ins_Opt.

As mentioned earlier, for an instantiation $t_0 : t_1 \cdots t_{n-1}$, if for some j, k, $j = 0, 1, 2, \ldots, n$ $j < k < j+n$, it is the case that for all l, $j < l < k$, $arity(t_{l \bmod n}) = 1$, then $hdr(t_j) \neq hdr(t_{(j+1) \bmod n}$ $\cdots \neq hdr(t_{k \bmod n})$. Thus, ideally, an algorithm for precisely detecting instantiations must ensure t if it does a 2-arity traversal of an edge (v_0, u_0) followed by a sequence of 1-arity traversals of ed $(v_1, u_1), \ldots, (v_{p-1}, u_{p-1})$ and finally a 2-arity traversal of edge (v_p, u_p), then all the edges traver $(v_0, u_0), (v_1, u_1), \ldots, (v_p, u_p)$, are distinct. Detect_Ins_Opt ensures that the edges $(v_0, u_0), \ldots, (v_p,$ are distinct since there are dependencies between any two edges of the TSGD that is passed as argument to Detect_Ins_Opt. However, in case there are no dependencies between certain edge the TSGD, the computational complexity of such an ideal algorithm that ensures $(v_0, u_0), \ldots, (v_p,$ are distinct would be prohibitive (the problem of precisely detecting instantiations by traversing ed in the TSG is intractable). Thus, procedure Detect_Ins_TSG1 only ensures that edges (v_0, u_0) (v_p, u_p) are distinct, while procedure Detect_Ins_TSG2 goes one step further and ensures that for $i = 1, \ldots, p$, (v_i, u_i) is distinct from both (v_0, u_0) and (v_{i-1}, u_{i-1}). For this purpose, Detect_Ins_TS appends to $anc(v')$, when v' is visited, the node u such that (u, v') is the most recent 2-arity traver edge by Detect_Ins_TSG1; while Detect_Ins_TSG2 appends to $anc(v')$, when v' is visited, the orde pair of nodes (u, w) such that (u, v') is the most recent 2-arity traversed edge and (w, v') is the m recently traversed edge.

In Detect_Ins_TSG1, since the TSG does not contain any dependencies, and due to the conditio Step 3(a), in order to detect instantiations, for any state st of F, every node v' in the TSG may n to be visited in state st twice during the execution of Detect_Ins_TSG1, the nodes appended to anc

Theorem 1: Let R be a complete regular specification. The optimistic scheme ensures that correct with respect to R.

Proof: See Appendix B. □

The complexity of the optimistic scheme is dominated by the number of steps it takes to pro a val_i operation. Procedure Detect_Ins_Opt can be shown to terminate in $O(n_S n_G^2 m)$ steps. Since the worst case, Detect_Ins_Opt is invoked for every regular term in R and for every subtransactio global transaction G_i, the complexity of the optimistic scheme is as stated in the following theore

Theorem 2: The worst-case complexity of the optimistic scheme is $O(n_S n_G^2 m n_R v_S)$. □

Note that n_R, n_S and v_S can be expected to be small in comparison to the number of gl transactions n_G and the number of sites m in the MDBS environment. Also, in our complexity anal of the optimistic scheme, we assume that Detect_Ins_Opt can be implemented such that each of three conditions in Step 2 can be checked in constant time.

6 Conservative Schemes

In this section, we present conservative schemes for ensuring that global schedule S is correct. schemes utilize a data structure referred to as the *transaction-site graph* (TSG). A TSG is similar to TSGD, except that it contains no dependencies. Thus, a TSG is a 3-tuple (V, E, L). Also, associa with every site s_k, is a queue. Initially, all queues are empty, and for the TSG, both $V = \emptyset$ and E When $init_i$ is processed, edges in the TSG are traversed in order to detect potential instantiat of regular terms involving G_i. In case potential instantiations are detected, the processing of cer $ser_k(G_i)$ operations in the queues is constrained by "marking" them. For an operation o_j in QUE $cond(o_j)$ and $act(o_j)$ are defined as follows (no actions are performed when a val_i operation is proces and $cond(val_i)$ is $true$):

- $\boldsymbol{cond(init_i)}$: $true$.
- $\boldsymbol{act(init_i)}$: The pseudocode in Figure 8 is executed. Procedure Detect_Ins_TSG? in the p docode can be either Detect_Ins_TSG1 (see Figure 11 in Appendix A) or Detect_Ins_TSG2 Figure 12 in Appendix A) (the two procedures differ in the degree of concurrency they pe and their complexities).
- $\boldsymbol{cond(ser_k(G_i))}$: For every transaction $G_j \in V$ such that $ser_k(G_j) \in G_j$, if $act(ser_k($ has executed, then $act(ack(ser_k(G_j)))$ has also completed execution. In addition, if $ser_k(G$ marked, then it is the first element in the queue for site s_k.
- $\boldsymbol{act(ser_k(G_i))}$: Operation $ser_k(G_i)$ is submitted to the local DBMSs (through the servers) execution.
- $\boldsymbol{cond(ack(ser_k(G_i)))}$: $true$.
- $\boldsymbol{act(ack(ser_k(G_i)))}$: Operation $ser_k(G_i)$ is deleted from the queue for site s_k (note that $ser_k($ may not be at the front of the queue for site s_k). Operation $ack(ser_k(G_i))$ is sent to GTM_1.
- $\boldsymbol{cond(fin_i)}$: $true$.

Let (V', E', D', L') be the TSGD obtained as a result of deleting from (V, E, D, L), the edges an
dependencies incident on transactions $G_k \in V$, $G_k \neq G_i$, such that val_k has not yet been processe

for every regular term $RT = e_0 : reg\_exp$ in R such that $type(G_i) = hdr(e_0)$ **do**
    for every subtransaction G_{ik} such that $type(G_{ik}) = last(e_0)$ **do**
        begin
        if $arity(e_0) = 1$ **then** $set_1 := \{s_k\}$
                    else $set_1 := \{s_l : (s_l \neq s_k) \wedge (type(G_{il}) = first(e_0))\}$;
        if Detect_Ins_Opt$((V', E', D', L'), G_i, s_k, set_1, RT)$ = abort **then**
            begin
            Delete G_i along with all its edges and dependencies from the TSGD;
            Inform GTM_1 to abort G_i;
            exit()
            end
        end
Inform GTM_1 to commit G_i

Figure 7: Pseudocode for $act(val_i)$

and element l_0, $head(l)$ returns l_1, $tail(l)$ returns $[l_2, \ldots, l_p]$ and $l_0 \circ l$ returns $[l_0, l_1, l_2, \ldots, l_p]$. Also an ordered pair $o = (o_1, o_2)$, $o[1] = o_1$, while $o[2] = o_2$.

Detect_Ins_Opt utilizes the finite automaton $F = FA(RT)$ to ensure that the sequence of ed traversed by it corresponds to a string in $L(reg\_exp)$. Every time Detect_Ins_Opt traverses an ed the current state of F is updated and a node is visited (the node is said to be visited in a state equa the current state of F. A node may be visited multiple times during the execution of Detect_Ins_O The current node being visited is stored in variable v, while the current state of F is stored in varia cur_st. Since instantiations may contain elements with arity 1, edge traversals do not always re in a new node being visited. If, for an edge (v, u), $st' = st_F(cur\_st, \overline{L(v, u)})$ is defined, and if traversal of edge (v, u), cur_st is set to st', then the node visited is v itself (since the traversed e (u, v) corresponds to an element with arity 1 in the instantiation). We refer to the edge traversal a 1-arity traversal. However, if for edge (v, u), $st = st_F(cur\_st, L(v, u))$ is defined, and if on traversa edge (v, u), cur_st is set to st, then the node visited is u (thus, edge (v, u) is traversed normally). refer to the edge traversal as a 2-arity traversal.

Since for any two consecutive elements t_i and $t_{(i+1) \bmod n}$ in an instantiation $t_0 : t_1 \cdots t_{n-1}$, $hdr(t_i$ $hdr(t_{(i+1) \bmod n})$, and for an element t_j with arity 2, $first(t_j) \neq last(t_j)$, consecutive edges traversed Detect_Ins_Opt must be distinct. This is ensured by appending to the list $anc(v')$, when v' is visi the node u such that (u, v') is the most recently traversed edge. Furthermore, an edge is traver only if it satisfies the condition in Step 3(a). Since the TSGD contains dependencies, and due to conditions in steps 3(a) and 3(b), in order to detect instantiations, for any state st of F, every n v' in the TSGD is visited in state st at least once for every edge (v', u) whose traversal could re in v' being visited in state st. However, in order to prevent a node v' from being visited in the sa state st due to the traversal of the same edge (v', u) multiple times, the ordered pair (st, u) is ad to $V\_set(v')$ when v' is visited in state st due to edge (v', u) being traversed. Also, an edge m satisfy the condition in Step 3(c) before it can be traversed. Finally, every time a node v' is visi the current node and the current state of F just before v' is visited is appended to $F\_list(v')$ to ena backtracking from v' to take place (Step 4 of procedure Detect_Ins_Opt).

When a val_i operation is processed, G_i is aborted if there is an instantiation of a regular term inv ing G_i and other transactions G_j that have already been committed in S. Thus, by a simple induc sites (site nodes) and global transactions in S (transaction nodes), E is a set of edges, D is a se dependencies (denoted by $\rightarrow$) between edges incident on a common site node, and L is a function t maps every edge to an ordered pair (τ_1, τ_2) where $\tau_1 \in g\tau$ and $\tau_2 \in l\tau$. Site and transaction no are labeled by the corresponding sites and transactions, respectively. Edges in the TSGD may present only between transaction nodes and site nodes. An edge between a transaction node G_i a site node s_k is present only if operation $ser_k(G_i) \in G_i$, and is denoted by either (s_k, G_i) or $(G_i,$ The set of edges $\{(G_i, s_k) : ser_k(G_i) \in G_i\}$ are referred to as G_i's edges. Dependencies specify relative order in which operations are processed and are sometimes also used to restrict the process of $ser_k(G_i)$ operations. If (G_i, s_k) and (s_k, G_j) are edges in the TSGD, then a dependency of the fo $(G_i, s_k) \rightarrow (s_k, G_j)$ denotes that $ser_k(G_i)$ is processed before $ser_k(G_j)$. In addition, L maps the e (G_i, s_k) to the pair $(type(G_i), type(G_{ik}))$.

The optimistic scheme uses the TSGD to store information relating to the serialization orde global subtransactions. Edges and dependencies are added to the TSGD when $ser_k(G_i)$ operations submitted for execution to the local DBMSs, and the TSGD is traversed in order to detect instantiati of regular terms when a val_i operation is processed. We now specify, for every operation o_j in QUE $cond(o_j)$ and $act(o_j)$ (no actions are performed when an $init_i$ operation is processed, and $cond(in$ is $true$). Initially, for the TSGD, $V = \emptyset$, $E = \emptyset$, $D = \emptyset$.

- $\boldsymbol{cond(ser_k(G_i))}$: For every transaction G_j such that $ser_k(G_j) \in G_j$, if $act(ser_k(G_j))$ has cuted, then $act(ack(ser_k(G_j)))$ has also completed execution.

- $\boldsymbol{act(ser_k(G_i))}$: Edge (G_i, s_k) is inserted into the TSGD. For every transaction $G_j \in V$ such t edge $(G_j, s_k) \in E$ and $act(ser_k(G_j))$ has executed, dependency $(G_j, s_k) \rightarrow (s_k, G_i)$ is addec D. For every transaction $G_j \in V$ such that $(G_j, s_k) \in E$ and $act(ser_k(G_j))$ has not yet b executed, dependency $(G_i, s_k) \rightarrow (s_k, G_j)$ is added to D. Operation $ser_k(G_i)$ is submitted to local DBMSs (through the servers) for execution.

- $\boldsymbol{cond(ack(ser_k(G_i)))}$: $true$.

- $\boldsymbol{act(ack(ser_k(G_i)))}$: Operation $ack(ser_k(G_i))$ is sent to GTM_1.

- $\boldsymbol{cond(val_i)}$: $true$.

- $\boldsymbol{act(val_i)}$: The pseudocode in Figure 7 is executed. Procedure Detect_Ins_Opt in the pseudoc traverses the TSGD in order to detect instantiations of regular terms, and is as shown in Figur in Appendix A

- $\boldsymbol{cond(fin_i)}$: $true$.

- $\boldsymbol{act(fin_i)}$: For every transaction $G_j \in V$ such that val_j has been processed, if for every tran tion $G_k \in V$ serialized before G_j, val_k has been processed, then G_j, along with all its edges dependencies, is deleted from the TSGD.

When a val_i operation is processed, for certain regular terms in R, Detect_Ins_Opt is invoke order to determine if there is an instantiation of the regular term in R involving transaction G_i (r that there is a dependency between every pair of edges in (V', E', D', L')). Procedure Detect_Ins_ traverses edges in the TSGD, beginning with node G_i in a direction against that of dependencies in TSGD in order to detect instantiations of RT. If it detects an instantiation of RT involving transac G_i, Detect_Ins_Opt returns abort. Since regular specifications are complete, any instantiation o regular term RT involving transaction G_i can be detected by traversing the TSGD beginning at n

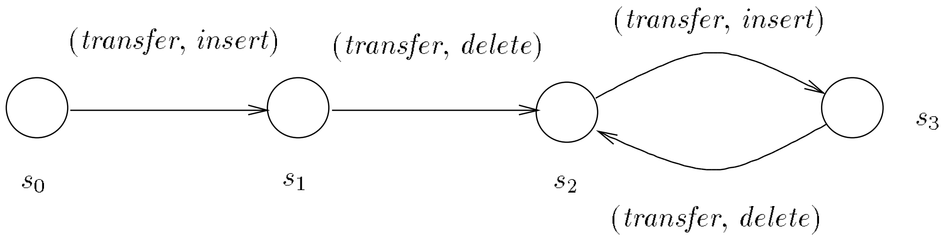

Figure 6: Finite Automaton for (*approx_listing* : *list, list*): (*transfer* : *insert, delete*)

Thus, $\underline{reg\_exp'}$ is a regular expression over the alphabet $\Sigma_F = \{(\tau_1, \tau_2) : (\tau_1 \in g\tau) \wedge (\tau$
$l\tau)\} \cup \{\overline{(\tau_1, \tau_2)} : (\tau_1 \in g\tau) \wedge (\tau_2 \in l\tau)\}$. $FA(RT)$ is a deterministic finite automaton that acc
exactly the strings in $L(reg\_exp')$. □

Note that transitions between states in $FA(RT)$ are due to elements in Σ_F. For a finite automa
F, we denote the initial state by $init\_st_F$ and the state transition function by st_F. We disting
between elements with arity 1 and those with arity 2 in the construction of $FA(RT)$ since the gra
need to be traversed differently in the two cases. For the regular term $RT = (approx\_listing : list,$
: (*transfer* : *insert, delete*)+, the finite automaton $FA(RT)$ is as shown in Figure 6 (note that s_2 i
accept state).

We define the *complexity* of a concurrency control scheme to be the average number of steps it t
the scheme to schedule all the operations associated with global transaction G_i. For the purpos
analyzing the complexity of the various schemes, we assume the following.

- The average number of sites at which a global transaction executes is v_S.
- At no point during the execution of a scheme does the difference between the number of i
and fin_i operations processed by the scheme exceed n_G (we assume that $v_S \ll n_G$).
- The number of regular terms in the regular specification R is n_R.
- Let RT be the regular term in R such that $FA(RT)$ has the maximum number of states.
denote by n_S, the number of states in $FA(RT)$.

In the following sections, we present an optimistic scheme based on the certifier approach,
conservative schemes for ensuring that S is correct. The optimistic scheme we present provid
higher degree of concurrency than the conservative schemes but could result in global transac
aborts that could hurt performance. (A concurrency control scheme, say CC_1, is said to provi
higher degree of concurrency than another concurrency control scheme CC_2 if, for any given orde
insertion of operations into QUEUE by GTM_1, CC_2 does not cause a fewer number of operation
be added to WAIT than CC_1). We specify the concurrency control schemes by specifying the
structures maintained by the scheme, and $cond(o_j)$, $act(o_j)$ for the various operations. We also s
the complexity of each of the schemes, and compare the degree of concurrency provided by the var
schemes. An analysis of the complexity of the schemes and proofs of their correctness can be foun
the appendices.

5 An Optimistic Scheme

The optimistic scheme presented in this section utilizes a data structure, referred to as the *transact*
site graph with dependencies (TSGD), introduced in [MRB+92]. A TSGD is an undirected bi-pa

procedure Basic_Scheme():
Initialize data structures;
while (true)
begin
Select operation o_j from the front of QUEUE;
if $cond(o_j)$ **then begin**
    $act(o_j)$;
    while (there exists an operation $o_l \in$ WAIT such that $cond(o_l)$ is true)
        begin
        $act(o_l)$;
        WAIT := WAIT $-\{o_l\}$
        end
    end
  else WAIT := WAIT $\cup \{o_j\}$
end

Figure 5: Basic Structure of Concurrency Control Schemes

The regular specification for the car rental example presented in Section 3 is not complete. Howe the regular specification containing the following additional regular terms (in addition to those li in Section 3) is complete. In the following regular terms, R_1 is the regular expression ((*accurate_lis* : *list, list*) + (*approx_listing* : *list, list*) + (*transfer* : *delete, insert*) + (*transfer* : *insert, delete* (*booking* : *reserve, reserve*)).

- (*transfer* : *insert, delete*) : (*transfer* : *insert, delete*)* (*approx_listing* : *list, list*) (*transfer* : *ins delete*)*
- (*transfer* : *insert, delete*) : R_1* ((*accurate_listing* : *list, list*) + (*accurate_listing* : *list*)) R_1*
- (*transfer* : *delete, insert*) : R_1* ((*accurate_listing* : *list, list*) + (*accurate_listing* : *list*)) R_1*
- (*booking* : *reserve, reserve*) : R_1* ((*accurate_listing* : *list, list*) + (*accurate_listing* : *list*)) R_1*
- (*approx_listing* : *list, list*) : R_1* ((*accurate_listing* : *list, list*) + (*accurate_listing* : *list*)) R_1*

Given a regular specification, it is possible to automate the process of determining the regular te that need to be added to it such that it becomes complete. Note that the addition of regular te to a regular specification so that it is complete needs to be performed only once (when the syste configured) and can be handled off-line.

Also, in order to detect instantiations of a regular term $RT = e_0 : reg\_exp$, the algorithms traversing graphs need to determine if there exist global transactions and subtransactions such t the sequence of their types is a string in $L(reg\_exp)$. For this purpose, the schemes employ a *fi automaton* [HU79], denoted by $FA(RT)$, which is defined as follows.

Definition 4: Let $RT = e_0 : reg\_exp$ be a regular term. Let reg_exp' be the regular expres obtained from reg_exp by performing the following two steps (in the order mentioned).

1. Replace each occurrence of $(\tau_1 : \tau_2)$ in reg_exp by $\overline{(\tau_1, \tau_2)}$.

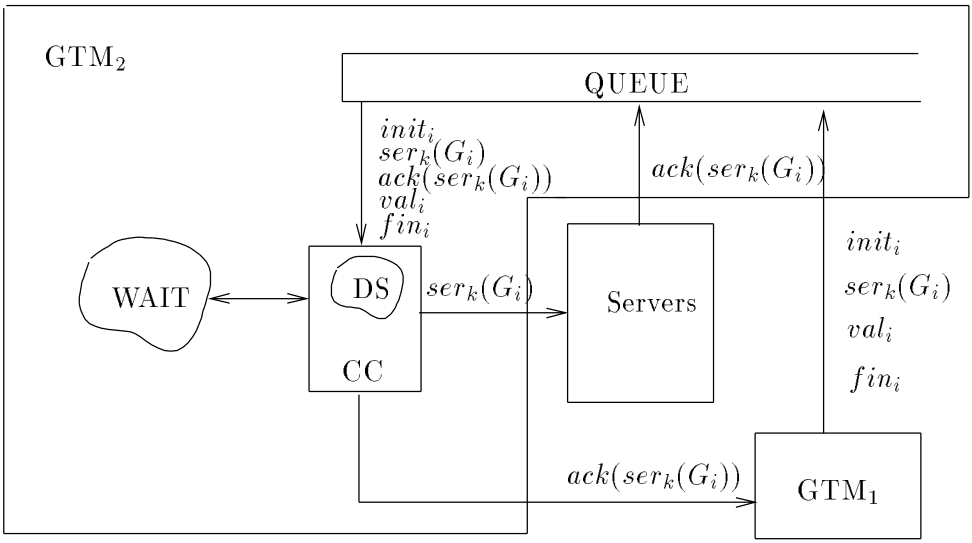

Figure 4: Basic Structure of GTM$_2$

- $\boldsymbol{ack(ser_k(G_i))}$: Operation $ack(ser_k(G_i))$ is forwarded to GTM$_1$.
- $\boldsymbol{val_i}$: GTM$_2$ determines if global transaction G_i can be committed without causing S to con instantiations of regular terms. If it can, GTM$_2$ informs GTM$_1$ to commit G_i, else it info GTM$_1$ to abort G_i.
- $\boldsymbol{fin_i}$: Information relating to G_i is deleted from DS.

We denote by $act(o_j)$, the actions performed by CC when it processes an operation o_j in QUE Also, associated with every operation o_j in QUEUE is a condition, $cond(o_j)$, that is defined over and that must hold if o_j is to be processed by CC. If $cond(o_j)$ does not hold when operation o selected from QUEUE by CC, then o_j is added to a set of waiting operations, WAIT, to be proces at a later time when $cond(o_j)$ becomes true. Thus, every scheme for ensuring the correctness o has the same basic structure as shown in Figure 5. However, different schemes differ in the values $act(o_j)$ and $cond(o_j)$ for the various operations, and the data structures associated with the scheme. instance, since *conservative* schemes do not abort transactions [BHG87], the set of sites a transac G_i executes at must be known *a priori*, and is added to DS when $init_i$ is processed; however, actions are performed by conservative schemes when a val_i operation is processed. On the other ha in *optimistic* schemes based on the certifier approach [BHG87], no actions are performed when an i operation is processed (the set of sites a transaction G_i executes at are not required to be know *priori* when $init_i$ is processed). A concurrency control scheme can be specified by specifying $cond($ $act(o_j)$ for the various operations, and the data structures maintained by the scheme.

Concurrency control schemes for GTM$_2$ presented in this paper are graph-based schemes. schemes involve traversal of graphs in order to detect instantiations of regular terms in the glo schedule S. In order to enable instantiations to be detected efficiently, our schemes require reg specifications to be complete, defined below.

Definition 3: A regular specification R is said to be complete if for every regular term RT $e_0 : reg\_exp_1$ in R, the following is true: let $e_1 e_2 \cdots e_{n-1}$, $n > 1$, be any string in $L(reg\_exp_1)$ s that for all $i = 1, 2, \ldots, n-1$, $e_i \in \Sigma$. For every i, $i = 1, 2, \ldots, n-1$, there exists a regular t $RT_2 = e_i : reg\_exp_2$ in R, such that the string $e_{(i+1) \bmod n} \cdots e_{(i+n-1) \bmod n}$ is an element of $L(reg\_ex$

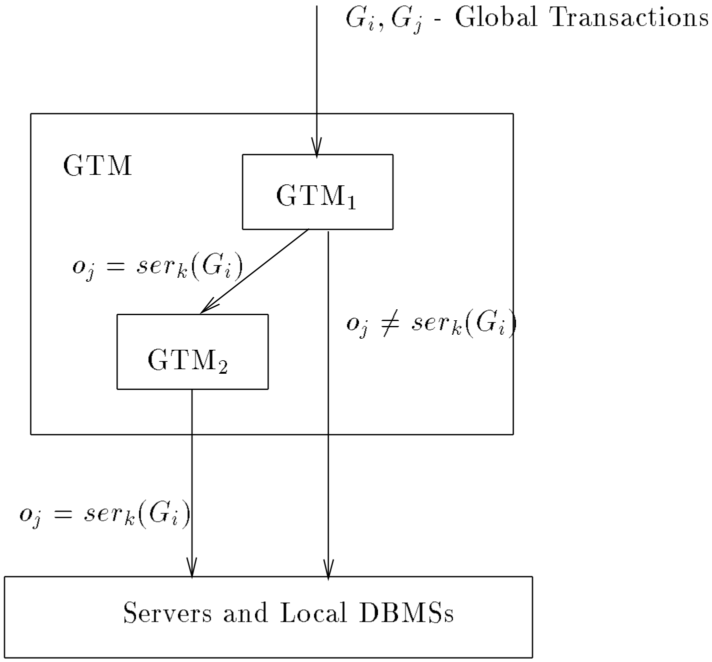

Figure 3: The GTM Components

utility is discussed below). We now briefly describe the operations in QUEUE for an arbitrary glo
transaction G_i and site s_k.

- ***init***$_i$: This operation is inserted into QUEUE by GTM_1 before any other operation belong
  to G_i is inserted into QUEUE.

- $\boldsymbol{ser_k(G_i)}$: This operation is inserted into QUEUE by GTM_1 in order to request the execu
  of operation $ser_k(G_i)$.

- $\boldsymbol{ack(ser_k(G_i))}$: This operation is inserted into QUEUE by the servers when the local DBI
  complete executing operation $ser_k(G_i)$.

- ***val***$_i$: This operation is inserted into QUEUE by GTM_1 before the global transaction G
  committed (global subtransactions may have been committed, however) and after $ack(ser_k(G$
  for all $ser_k(G_i)$ operations belonging to G_i have been received by GTM_1.

- ***fin***$_i$: This operation is inserted into QUEUE by GTM_1 after val_i is inserted into QUEUE.

Note that the $init_i$, val_i and fin_i operations do not belong to global transaction G_i.

Figure 4 illustrates the basic structure of GTM_2. CC is any concurrency control scheme for ensu
the correctness of S. CC selects operations from the front of QUEUE, in order to *process* th
Associated with CC are certain data structures (DS) that are manipulated every time an opera
selected from QUEUE is processed by it. In addition, the following actions are performed by CC w
it processes an operation o_j in QUEUE.

- ***init***$_i$: Operation $init_i$ contains information relating to global transaction G_i (e.g., the t
  of G_i and its subtransactions, the set of sites, if known *a priori*, at which G_i executes). T
  information is added to DS and is utilized by CC to detect instantiations of regular terms in

- $\boldsymbol{ser_k(G_i)}$: Operation $ser_k(G_i)$ is submitted to the local DBMSs for execution (the ord

Note that in contrast to existing approaches [GM83, FO89], our approach scales well to the addi of new global applications and local DBMS procedures in the MDBS. For example, in [GM83], addition of a new application may require modifications to existing compatibility sets in orde permit more interleavings and maximize the degree of concurrency. However, since regular specificati specify unacceptable interleavings using regular expressions, in case new global applications are ad to the MDBS, no modifications are required to previously existing regular terms; only additional reg terms that specify the unacceptable interleavings involving the newly added global applications n to be added to the regular specification. Any resulting redundancy among the regular terms car detected and eliminated using well known techniques for determining equivalence of regular expressi [HU79].

3.3 Serialization Functions

In order to develop concurrency control schemes for ensuring correctness, we utilize the notior *serialization functions* introduced in [MRB+92], which is similar to the notion of serialization ev [ED90]. Let σ_k be the set of all global subtransactions in schedule S_k. A serialization function for *ser*, is a function that maps every subtransaction in σ_k to one of its operations such that for any of subtransactions $G_{ik}, G_{jk} \in \sigma_k$, if G_i is serialized before G_j in S_k, then $ser(G_{ik}) \prec_{S_k} ser(G_{jk})$.

For example, if the *timestamp ordering* (TO) concurrency control protocol is used at site s_k, the local DBMS at site s_k assigns timestamps to transactions when they begin execution, then function that maps every subtransaction $G_{ik} \in \sigma_k$ to G_{ik}'s begin operation is a serialization func for s_k. For a site s_k, there may be multiple serialization functions. For example, if the local DBM s_k follows the *two-phase locking* (2PL) protocol, then a possible serialization function for s_k maps ev subtransaction $G_{ik} \in \sigma_k$ to the operation that results in G_{ik} obtaining its last lock. Alternatively, function that maps every subtransaction $G_{ik} \in \sigma_k$ to the operation that results in G_{ik} releasing its lock is also a serialization function for s_k[2]. We denote by ser_k, any one of the possible serializa functions for site s_k.

By controlling the execution of $ser_k(G_i)$ operations, the GTM can control the serialization o of global transactions at the various, and the correctness of global schedule S can be ensured. T we split the GTM into two components, GTM_1 and GTM_2 (see Figure 3). GTM_1 utilizes the in mation on serialization functions for various sites in order to determine for every global transac G_i, operations $ser_k(G_i)$, and submits them to GTM_2 for processing. The remaining global transac operations (that are not $ser_k(G_i)$) are directly submitted to the local DBMSs (through the server

GTM_2 is responsible for ensuring that the operations submitted to it by GTM_1 execute at local DBMSs in such a manner that S contains no instantiations of regular terms. Our concern, the remainder of the paper, shall be the development of concurrency control schemes for GTM_2 t ensure S is correct. Our schemes require only GTM_2 to be centrally located; GTM_1 can be distribu across the various sites in order to reduce the overhead involved in global transaction processing.

4 Structure of Concurrency Control Schemes

In this section, we describe the structure and complexity of concurrency control schemes employe GTM_2 for ensuring the correctness of S (the concurrency control model we adopt is similar to presented in [MRB+92]). As mentioned earlier, GTM_1 submits the $ser_k(G_i)$ operations belongin each global transaction G_i to GTM_2. GTM_1 inserts these operations into a queue, QUEUE. In addit for every global transaction G_i, GTM_1 inserts into QUEUE, the operations $init_i$, val_i and fin_i (wh

[2] Actually, any function that maps every subtransaction $G_{ik} \in \sigma_k$ to one of its operations that executes betweer

In the execution in Figure 1(a),

$$(G_0 : G_{00}, G_{0(n-1)}) : (G_{n-1} : G_{(n-1)(n-1)}, G_{(n-1)(n-2)}) \cdots (G_1 : G_{11}, G_{10})$$

is an instantiation of the regular term

$$(\mathit{approx\_listing} : \mathit{list}, \mathit{list}) : (\mathit{transfer} : \mathit{insert}, \mathit{delete})+$$

since $type(G_0 : G_{00}, G_{0(n-1)}) = (\mathit{approx\_listing} : \mathit{list}, \mathit{list})$, and for $n > 1$,
$type(G_{n-1} : G_{(n-1)(n-1)}, G_{(n-1)(n-2)}) \cdots type(G_1 : G_{11}, G_{10}) = (\mathit{transfer} : \mathit{insert}, \mathit{delete}) \cdots (\mathit{trans}$
insert, delete),
which is a string in L((*transfer* : *insert, delete*)+).

Note that, in an instantiation, there can be multiple occurrences of a global transaction. Howe since local schedules are serializable and global transactions have a single subtransaction at each in the instantiation $t_0 : t_1 \cdots t_{n-1}$, any two adjacent global transactions $hdr(t_j)$ and $hdr(t_{(j+1)\text{mo}}$ $j = 0, 1, \ldots, n-1$, are distinct. Actually, instantiations posess the following stronger property.

Property 1: In an instantiation $t_0 : t_1, \ldots, t_{n-1}$, if for some j, k, $j = 0, 1, 2, \ldots, n-1$, $j < k < j$ it is the case that for all l, $j < l < k$, $arity(t_{l \bmod n}) = 1$, then $hdr(t_j) \neq hdr(t_{(j+1) \bmod n}) \neq \cdot$ $hdr(t_{k \bmod n})$. Also, for all r, s, $j \leq r < s \leq k$, $last(t_{r \bmod n})$ is serialized after $first(t_{s \bmod n})$ at the s site. □

The set of all cycles in the serialization orders of global transactions characterized by a reg term in S is the set of all instantiations of the regular term in S. Since we use regular terms to spe unacceptable interleavings among global subtransactions, none of the cycles characterized by reg terms in S must be contained in S if it is to be correct.

Definition 2: Let R be a regular specification. Global schedule S is correct with respect to for every regular term RT in R, there are no instantiations of RT in S. □

Since regular specifications are based on regular expressions, they are a powerful, yet simple too capturing the set of unacceptable interleavings. We expect that in most MDBS applications, reg specifications will be adequate to specify the set of non-serializable executions that are unaccepta The regular specification for the car rental example in Section 3 contains the following three reg terms.

1. (*approx_listing* : *list, list*) : (*transfer* : *insert, delete*)+
2. (*accurate_listing* : *list, list*) : ((*accurate_listing* : *list, list*) + (*approx_listing* : *list, list*) + (*tran* : *delete, insert*) + (*transfer* : *insert, delete*) + (*booking* : *reserve, reserve*))+
3. (*accurate_listing* : *list*) : ((*accurate_listing* : *list, list*) + (*approx_listing* : *list, list*) + (*trans* *delete, insert*) + (*transfer* : *insert, delete*) + (*booking* : *reserve, reserve*))+

Term 1 characterizes the set containing only the unacceptable interleavings involving *approx_lis* and *transfer* transactions (the unacceptable interleaving in Figure 1(a) is in the set of interleav characterized by regular term 1, while the acceptable interleaving in Figure 1(b) is not). Note it is not possible to characterize the set containing only the unacceptable interleavings involving *prox_listing* and *transfer* transactions using any of the mechanisms for specifying interleavings prop in [GM83, FO89]. The terms 2 and 3 characterize the set of unacceptable interleavings that involv characterizes a set of cycles in the serialization order of global transactions. Every string in L specifies the types and serialization orders of global transactions in cycles belonging to the set. example, a cycle in the serialization orders of global transactions $G_0, \ldots, G_{n-1}$ such that for all $0, \ldots, n-1$, G_k is serialized before $G_{(k+1) \bmod n}$ is in the set of cycles characterized by R if there ex a string $\tau_0 \cdots \tau_{n-1}$ in $L(R)$ (each τ_i is a global transaction type) such that for all $k = 0, \ldots, n$ $type(G_k) = \tau_k$. Global schedule S is correct if it does not contain any of the cycles in the serializa order of global transactions characterized by regular terms in the regular specification.

We now formally define the syntax and semantics of regular specifications, and the correctnes global schedules. Let Σ denote the set

$$\{(\tau_1 : \tau_2, \tau_3) : (\tau_1 \in g\tau) \wedge (\tau_2, \tau_3 \in l\tau)\} \cup \{(\tau_1 : \tau_2) : (\tau_1 \in g\tau) \wedge (\tau_2 \in l\tau)\}.$$

In the car rental example, (*approx_listing* : *list, list*) is an element of Σ; however, (*approx_listi* *transfer*) does not belong to Σ since *transfer* $\notin l\tau$. A regular specification R is a finite set of ter referred to as *regular terms*, each having the following syntax:

$$e_0 : reg\_exp$$

where e_0 is an element of Σ and reg_exp is a regular expression over the alphabet Σ.

In the car rental example, (*approx_listing* : *list, list*) : (*transfer* : *insert, delete*)+ is a regular te Note that a regular term contains only global transaction and global subtransaction types. The rea we do not include local transaction information in regular terms is that the GTM has no control the execution order of local transactions since they execute outside the control of the GTM.

Each regular term characterizes a set of cycles in the serialization orders of global transacti (and thus, a set of interleavings among global subtransactions). Every cycle in the set contai global transaction and global subtransactions with types mentioned in e_0, and strings in $L(reg\_e$ capture the types and serialization orders of the remaining global transactions and subtransaction the cycles.

In order to describe the set of cycles characterized by a regular term, we need to introduce following additional notation. For the global schedule S, Σ_S denotes the set
$\{(G_i : G_{ij}, G_{ik}) : G_{ij}$ and G_{ik} are distinct subtransactions of a committed transaction G_i in S $\} \cup$
$\{(G_i : G_{ij}) :$ G_{ij} is a subtransaction of a committed transaction G_i in $S\}$.
For an element $e \in \Sigma_S$, if $e = (e_1 : e_2, e_3)$, then $type(e) = (type(e_1) : type(e_2), type(e_3))$, whi $e = (e_1 : e_2)$, then $type(e) = (type(e_1) : type(e_2))$. Also, for an element $e \in \Sigma \cup \Sigma_S$, if $e = (e_1 : e_2,$ then $hdr(e) = e_1$, $first(e) = e_2$, $last(e) = e_3$ and $arity(e)$ is 2, while if $e = (e_1 : e_2)$, then $hdr(e) =$ $first(e)$ and $last(e)$ are both e_2, and $arity(e)$ is 1.

In the following definition, we introduce the notion of an instantiation of a regular term in o to identify the set of cycles in the serialization orders of global transactions characterized by a reg term in the global schedule S.

Definition 1: Let $RT = e_0 :$ reg_exp be a regular term and let S be a schedule. $t_0 : t_1 t_2 \cdots t$ $n > 1$, is an instantiation of RT in S if

- for all j, $j = 0, 1, \ldots, n-1$,

  1. $t_j \in \Sigma_S$, and
  2. $last(t_j)$ and $first(t_{(j+1) \bmod n})$ execute at the same site, and $last(t_j)$ is serialized a $first(t_{(j+1) \bmod n})$ at the site, and

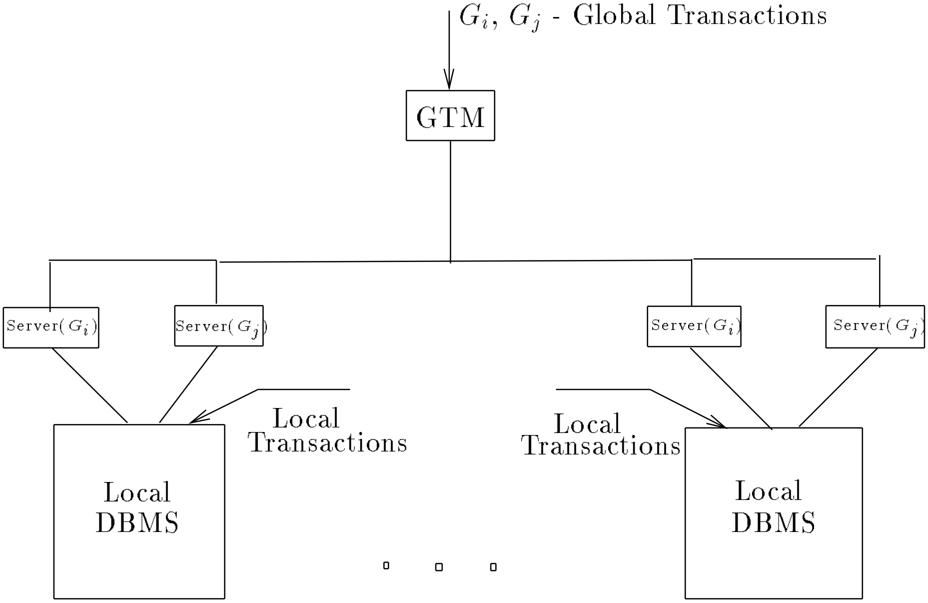

Figure 2: The MDBS Model

a totally ordered set of **read**, **write**, **begin** and **commit** operations. The *local schedule* at a site
denoted by S_k, is the set of all operations (belonging to local transactions and global subtransacti
that execute at s_k with a total order $\prec_{S_k}$ on them. The *global schedule* S is the set of all the l
schedules S_k.

We assume that the GTM is centrally located, and controls the execution of global transacti
(our schemes require only one operation belonging to every global subtransaction to be submitte
the centrally located GTM; the remaining global subtransaction operations can be submitted dire
to the local DBMSs at the sites outside the control of the centrally located GTM. As a result, a num
of problems with having a centrally located GTM are alleviated since most of the global transac
processing can be done locally at the sites). It communicates with the various local DBMSs by me
of *server* processes (one per transaction per site) that execute at each site on top of the local DB
(see Figure 2). We assume that the interface between the servers and the local DBMSs provides
operations to be submitted by the servers to the local DBMSs, and the local DBMSs to acknowledge
completion of operations to the servers. The local DBMSs do not distinguish between local transacti
and global subtransactions executing at its site. In addition, each of the local DBMSs ensures t
local schedules are serializable.

3.2 Regular Specifications

Before we develop schemes that permit certain acceptable interleavings in an MDBS environment
need to develop a mechanism for specifying the set of unacceptable interleavings among transacti
Since local schedules are serializable and local transactions execute at a single site, any unaccepta
interleaving among transactions must be due to an unacceptable cycle in the serialization order
global transactions. The set of unacceptable cycles in the serialization orders of global transactions
be characterized using *regular specifications* which are defined using regular expressions (we assume
reader is familiar with the syntax and semantics of regular expressions as described in [HU79]). F
regular expression R, $L(R)$ is the set of all strings denoted by R.

We begin by giving the intuition that underlies our definition of regular specifications, follow
which, we present a formal treatment. A regular specification consists of regular terms, each of wh

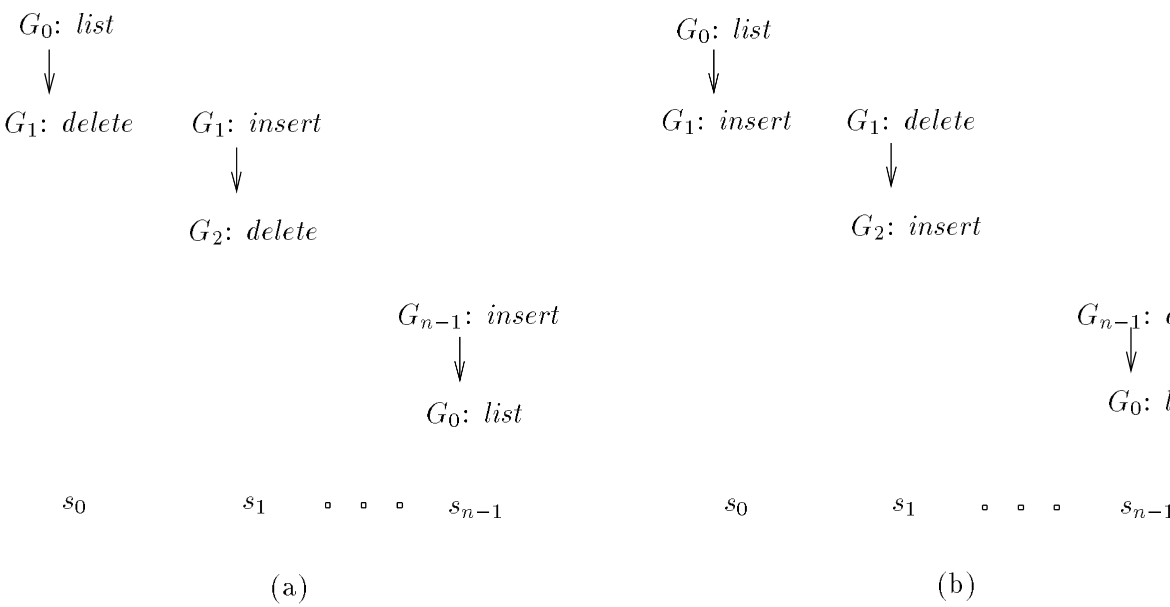

Figure 1: Unacceptable and Acceptable Executions

containing an *accurate_listing* transaction G_0 and any other transaction G_1 such that G_1 is seriali both before and after G_0 is unacceptable.

Note that an *accurate_listing* transaction returns a precise listing of car records, but cannot in leave with other transactions at all; on the other hand, although an *approx_listing* transaction retu an approximate listing of car records, its subtransactions can interleave freely with subtransacti of *booking* transactions and most interleavings involving *approx_listing* and *transfer* transactions acceptable. Thus, based on the semantics of transactions, certain interleavings are acceptable. In m cases, since accurate listings may not be required, *approx_listing* transactions can be employed inst of *accurate_listing* transactions, thus enhancing the degree of concurrency. In the next section, develop a mechanism for specifying the set of undesirable interleavings between global subtransacti

3 Correctness of Schedules

In this section, we define the correctness of schedules in terms of undesirable interleavings among glo transactions. However, before defining the correctness of schedules, we first describe the MDBS mo that we adopt.

3.1 The MDBS Model

An MDBS is a collection of pre-existing and autonomous local DBMSs located at sites $s_0, s_1, \ldots, s_n$ Each local DBMS exports an interface consisting of procedures that can be invoked by the glo transactions in order to access and manipulate data at the local DBMSs. We denote by $l\tau$, the se types of all the procedures exported by local DBMSs. Every global transaction G_i has a type deno by *type*(G_i) and consists of one or more subtransactions, at most one per site, each resulting from invocation of a single local DBMS procedure. The set of global transaction types is denoted by and the subtransaction of G_i at site s_k is denoted by G_{ik}. The type of subtransaction G_{ik} is deno by *type*(G_{ik}) and is the same as the type of the local DBMS procedure whose execution result G_{ik}. In the car rental example in the previous section, $l\tau$ = {*reserve, delete, insert, list*} and g

- **list**. This procedure lists the information contained in records for a subset of cars in the datab Arguments to the *list* procedure include a predicate that identifies the subset of cars whose reco are to be listed (e.g., all the cars classified as compact, or all the cars not reserved on a partic date).

Since a single car cannot be associated with more than one branch (site) at any given time, following simple global integrity constraint is introduced due to the integration of the local DBMS

For every car, at most one local DBMS contains the record for the car.

The above constraint ensures that a single car is not rented out to multiple clients on the same date. integration facilitates the execution of global transactions that reserve cars at multiple local DBM list car records at multiple local DBMSs, and transfer car records from one local DBMS to another case the number of available cars at a site fall below a threshold value, cars are transferred to the from nearby sites). Global transactions invoke the exported local DBMS procedures mentioned ab (every invocation results in a global subtransaction), and are one of the following:

- **booking**. A transaction that reserves cars at multiple local DBMSs and consists of one or m *reserve* subtransactions.

- **transfer**. A transaction that transfers car records from one local DBMS to another, and cons of a *delete* subtransaction and an *insert* subtransaction.

- **approx_listing**. A transaction that is used to provide an approximate, conservative listin car records at multiple local DBMSs and consists of one or more *list* subtransactions. E *approx_listing* transaction is only required to see a consistent database state, that is, a s which satisfies the global integrity constraint. As a result, it is unacceptable for an *approx_lis* transaction to list the same car record as contained in two or more distinct local DBMSs. Howe it is acceptable for an *approx_listing* transaction to not list certain car records contained in local DBMSs.

- **accurate_listing**. A transaction that is used to return a global snapshot of the state of MDBS, and consists of one or more *list* subtransactions. Thus, every *accurate_listing* tran tion requires that every other transaction in the MDBS be either serialized before it or afte (it is unacceptable for a transaction to be serialized both before and after an *accurate_lis* transaction).

Since each local DBMS ensures serializability, global subtransactions execute in isolation. T if every local transaction in the MDBS environment preserves the global integrity constraint, t every non-serializable execution consisting of only local transactions, *booking* and *transfer* transacti preserves the global integrity constraint, and is acceptable. However, certain non-serializable executi containing *approx_listing* or *accurate_listing* transactions are unacceptable.

Consider an execution involving a global transaction G_0 of type *approx_listing* that lists all car records at sites s_0 and s_{n-1}, and a set of global transactions $G_1, \ldots, G_{n-1}$ of type *transfer*, w each G_i transfers, among other records, a particular car record r_0 from site s_{i-1} to s_i. Suppose for all $i = 0, 1, \ldots, n-1$, G_i is serialized before $G_{(i+1) \bmod n}$ at site s_i. This execution sequenc depicted in Figure 1(a), where the arrows represent the serialized before relationship among gl subtransactions, and the text following ":" are global subtransaction types. In the above execut the *approx_listing* transaction G_0 lists the car record r_0 as contained in databases at both site and s_{n-1}, and thus, sees an inconsistent database state. As a result, the execution in Figure 1(a on *regular expressions* over transaction types, and many of the interleavings that can be specified u our approach cannot be specified using the approaches in [GM83, FO89]. The proposed mechan assumes that the global transactions access data at local DBMSs via well-defined interfaces (in o to ensure that global transactions access their databases in a controlled and restricted fashion, l DBMSs export a fixed set of procedures that can be invoked by global transactions). The mechanis flexible, and can even be used to specify that global schedules resulting from the concurrent execu of transactions must be serializable. Unlike existing mechanisms, it scales well to the additio new global applications and local DBMS procedures in the system. Furthermore, the mechanism facilitates the development of efficient graph-based schemes (*optimistic* and *conservative*) for ensu that the concurrent execution of transactions in an MDBS environment meet the specifications. MDBS environments in which certain non-serializable executions are permissible, our schemes pro a higher degree of concurrency than schemes to ensure global serializability. We examine the tr off between the complexities and the degree of concurrency permitted by the various schemes. also show that although none of the conservative schemes proposed by us are optimal, the proble scheduling the operations of the various concurrently executing transactions in order to permit opti concurrency is NP-complete. The results in this paper are also applicable to homogeneous distrib database systems. Recovery algorithms in case of site failures, transaction aborts, etc. are outside scope of this paper and are not discussed.

The remainder of the paper is organized as follows. In Section 2, we present an MDBS applica in which the semantics of transactions can be exploited in order to relax the serializability req ment. In Section 3, we present our mechanism for specifying unacceptable interleavings in an MI environment and define correctness of global schedules. The concurrency control model we adop discussed in Section 4. In Section 5, we develop an optimistic scheme that ensures global schedules correct. Conservative schemes presented in sections 6 and 7 prevent unacceptable interleavings am global subtransactions. In Section 8, we show that the problem of optimally scheduling operations execution is NP-complete. Concluding remarks are offered in Section 9.

2 A Motivating Example

Consider a car rental company with branches at n geographically distributed sites $s_0, s_1, \ldots, s_{n-1}$. E branch has a database that contains information related to cars at the branch (one record per c The information stored in a record for a car comprises of a unique car identifier for the car, the m and model of the car, its classification (compact, luxury, etc.), the dates the car has been reserved, name and credit card number of the client that made the reservations etc.

Consider an MDBS environment in which the databases belonging to the various branches integrated. We assume that every local DBMS ensures the serializability of schedules at its site, each local DBMS exports the following procedures:

- **reserve**. This procedure reserves a car. Arguments to the *reserve* procedure include the clas cation of the car to be reserved, the name and credit card number of the person reserving the and the dates the car is to be reserved. The *reserve* procedure only utilizes information st in the car record in the local DBMS in order to ensure that the car is not reserved by mult clients on the same date.

- **delete**. This procedure deletes records corresponding to one or more cars from the datab Arguments to the *delete* procedure include a list of car identifiers of the cars whose records to be deleted.

1 Introduction

A *multidatabase system* (MDBS) consists of a number of pre-existing and autonomous local *data* *management systems* (DBMSs) that are integrated to enable users of a local DBMS to access d residing at remote systems. The local DBMSs may follow different concurrency control protocols. integration is transparent and must be accomplished without any modifications to the local DB software and to the pre-existing local applications. This is achieved by building on top of the lo DBMSs, a software module, referred to as the *global transaction manager* (GTM), that is respons for co-ordinating the execution of *global transactions* – transactions whose execution span multiple lo DBMSs. Those transactions that execute at a single local DBMS are referred to as *local transacti* Since the GTM is built on top of local DBMSs, and no modifications are made to local DBMS softw and to pre-existing local applications, local transactions execute outside the control of the G As a result, the GTM has no knowledge of the indirect conflicts between global transactions du the execution of local transactions, and the only way the GTM can enforce a particular orderin global transactions at a local DBMS is by controlling the execution of certain operations belongin global subtransactions (referred to as *serialization events* [ED90]). In such an environment, adop *serializability*[1] as the notion of correctness, as is done in [Pu88, BS88, ED90, BST90, BGRS91, PRI SKS91, GRS91, MRB+92, Raz92], could result in a low degree of concurrency and adversely affect performance of the system. Furthermore, in most MDBS environments, due to the autonomous na of of local DBMSs, integrity constraints between data items belonging to different local DBMSs be expected to be weak and few in number. As a result, serializability may not be required in or to preserve database consistency, and it would therefore be advantageous to come up with a wea notion of correctness for such systems.

For MDBS environments, two correctness criteria that are weaker than serializability have b proposed – *quasi-serializability* (QSR) [DE89] and *two-level serializability* (2LSR) [MRKS91]. B criteria require each local DBMS to generate serializable schedules. In addition, 2LSR requires t the restriction of global schedules to only global transactions to be serializable, while QSR impo the stronger condition that global schedules be equivalent to a schedule in which global transacti execute serially. QSR and 2LSR schedules preserve database consistency if transactions and integ constraints are of a restricted nature.

The degree of concurrency permitted by the correctness criteria proposed in [DE89, MRKS91 limited since they restrict themselves to only read and write operations. In [GM83, GMS87, FO the authors adopt a different approach in which a transaction is assumed to consist of steps, e of which could be semantically richer than read and write operations. In [GM83, FO89], types associated with transactions, and mechanisms that use the type information for specifying accepta interleavings between steps are developed. The authors also develop protocols to ensure that o the specified interleavings are permitted. In [GM83], transaction types are grouped into *compatib* *sets* which are used to specify interleavings. All the steps of transactions whose types belong t single compatibility set can interleave freely, while steps of transactions whose types belong to dist compatibility sets cannot interleave at all. In [FO89], acceptable interleavings are specified by specify the set of types of transactions that can interleave between any two consecutive steps of a transact A schedule is correct if it is equivalent to a schedule in which steps are executed serially, and betw any two consecutive steps of a transaction, only steps of those transactions that are permitted interleave appear in the schedule. In [PL91, KB91, WA92] the authors propose schemes that exp the semantics of applications in order to bound the inconsistency due to non-serializable executio

The approach we adopt in this paper enables the semantics of transactions to be exploited, an similar to the one used in [GM83, FO89]. However, our mechanism for specifying interleavings is ba

Exploiting Transaction Semantics in Multidatabase Systems

Rajeev Rastogi[1*]
Henry F. Korth[2]
Avi Silberschatz[1*]

[1]Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188 USA

[2]Matsushita Information Technology Laboratory
182 Nassau Street, third floor
Princeton, NJ 08542-7072

Abstract

Serializability is the traditionally accepted notion of correctness in most database systems. However, in a *multidatabase system* (MDBS) environment consisting of pre-existing and autonomous database systems, requiring schedules to be serializable could severely hurt performance. Besides, in a number of instances, the semantics of transactions can be exploited in order to permit certain non-serializable executions. In this paper, we propose a powerful, yet simple mechanism for specifying, in an MDBS environment, the set of non-serializable executions that are unacceptable. The undesirable interleavings among transactions are specified using *regular expressions* over transaction types. The mechanism facilitates the development of efficient graph-based schemes for ensuring that the concurrent execution of transactions in the MDBS environment meet the specifications. We examine the trade-off between the complexities and the degree of concurrency permitted by the various schemes. Finally, we show that the problem of scheduling operations for execution so as to permit optimal concurrency is NP-complete.

EXPLOITING TRANSACTION SEMANTICS IN MULTIDATABASE SYSTEMS

Rajeev Rastogi
Henry F. Korth
Avi Silberschatz

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

TR-92-45 December 1992

DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN

AUSTIN, TEXAS 78712