iJRASET
International Journal For Research in Applied Science and Engineering Technology

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 6 **Issue:** I **Month of publication:** January 2018

**DOI:** http://doi.org/10.22214/ijraset.2018.1434

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor :6.887*
*Volume 6 Issue I, January 2018- Available at www.ijraset.com*

# Analysis of Emerging Field Related to Distributed Concurrency control

Rajdeep Singh solanki[1]
[1]*Research scholar computer science department swami Vivekananda University, sagar, M.P., India*

***Abstract:*** ***In this chapter, we explore the emerging field related to distribute concurrency control. We discuss advance transaction process called nested transaction and issues of nested transaction. In this chapter we also describe nested transactions where the transactions from one system interact with the transactions from another system. We also show other issues cause by distributed concurrency control. We discuss that recovery considerations do not affect the correctness of our results.***

***Keywords: Distributed data base, concurrency control, Deadlock, Recovery***

## I. INTRODUCTION

The concept of nested transactions offers more decomposable execution units and finer grained control over concurrency and recovery than flat transactions. Furthermore, it supports the decomposition of a unit of work into subtasks and their appropriate distribution in a computer system as a prerequisite of intra-transaction parallelism. However, to exploit its full potential, suitable granules of concurrency control as well as access modes for shared data are necessary.

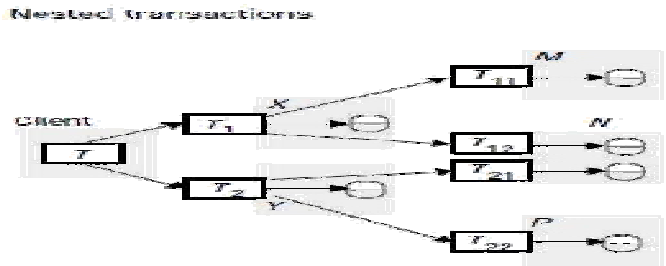
Deadlocks and recovery are a fundamental problem in distributed systems. Distributed systems are subject to different types of failure, and a transaction processing system needs a recovery algorithm in order to ensure correct execution. Two phase commit (2PC) is the most popular recovery protocol. Another problem is deadlock. Distributed deadlocks can occur in distributed systems when distributed transactions or concurrency control is being used.

Deadlock detection is more difficult in systems where there is no such central agent and processes may communicate directly with one another. Deadlock detection and resolution is one among the major challenges faced by a Distributed System.

## II. NESTED TRANSACTION CONCURRENCY CONTROL

Using transactions as we have to this point does not always allow applications the granularity of error isolation that may be desired. If the transaction aborts, all changes are rolled back. However, for more complicated transactions, we may want a finer granularity in error isolation. We need a way to isolate any errors that occur in the interaction with the local database, preventing such errors from aborting the entire transaction. The solution is to check and decrement the local database from within a nested transaction. A nested transaction is a new transaction begun from within the scope of another transaction.

The term nested refers to the fact that a transaction can be (recursively) decomposed into subtransactions, parts that form a logically related subtask. In this way, a parent transaction can have multiple children, each child being a subtransaction. A key point is that a successful sub transaction only becomes permanent (i.e., committed) if all its ancestors succeeded as well, whereas the inverse does not hold: if a child fails, the parent is free to try an alternative task, thereby rescuing the global work.




Nested Transaction

A transaction may contain any number of subtransactions, which again may be composed of any number of subtransactions - conceivably resulting in an arbitrarily deep hierarchy of nested transactions. The root transaction, which is not enclosed in any transaction, is called the top-level transaction (TL-transaction). Transactions having subtransactions are called parents, and their subtransactions are their children. We will also speak of ancestors and descendants. The ancestor (descendant) relation is the reflexive transitive closure of the parent (child) relation. We will use the term superior (inferior) for the non-reflexive version of the ancestor (descendant). The set of descendants of a transaction together with their parent/child relationships is called the transaction‘s hierarchy.

## III. BASIC LOCKING RULES FOR NESTED TRANSACTIONS

Locking as the standard method of concurrency control in DBMS has been used successfully for a variety of applications over the past decade and longer. Therefore, it is reasonable to choose conventional locking protocols as our starting point of investigation for nested transactions. Locking protocols offer three modes of synchronization - read, which permits multiple transactions to Share an object at a time, and write, which gives the right to a single transaction for exclusively accessing an object. Read Only, which again permits multiple transactions to share a lock but does not allow locks to be upgraded. Basic locking rules for nested transaction are:

We have four possible lock modes: NL, RO, S, and X. The Not Locked mode (NL) represents the absence of a lock request or a lock on the object. A transaction can acquire a lock on an object in some mode; then it holds the lock in the same mode until its termination (commit or abort) or until it upgrades the lock mode explicitly. Also, besides holding a lock, a transaction can retain a lock. Retention concept is required for modeling the correctness of nested transaction execution. When a subtransaction commits, its (retained and held) locks are inherited by its parent transaction, which in turn retains these locks. A retained lock is a place holder, i.e. unlike a lock 'held' by a transaction, for which a transaction has all the rights to access the locked object (in corresponding mode), a transaction retaining a lock does not have any right to access that object. A retained X-lock denoted by r:X ( as opposed to h:X for an X- lock held), indicates that the transactions outside the sphere of the retainer cannot acquire the lock, but the descendants of the retainer potentially can - subject to the locking rules given below. That is, if a transaction retains an X-lock, then all non- descendants of that transaction cannot hold the lock either in X- or in S-mode. If a transaction is a retainer of S-lock, it is guaranteed that a non-descendant of that transaction cannot hold the lock in X-mode, but potentially in S-mode. As soon as a transaction becomes the retainer of a lock, it remains retainer for that lock during its lifetime (i.e., till it comits).

We now describe the basic locking rules, including the extensions related to read-only mode. The discussions are with respect to an object O.

*A.* Transaction may acquire a lock in X-mode ifNo other transaction holds the lock in X- or S-mode, andAll transactions that retain the lock in X- or S-mode are ancestors of the requesting transaction.
*B.* Tranaction may acquire a lock in S-mode if
*C.* No other transaction holds the lock in X-mode, and All transactions that retain the lock in X-mode are ancestors of the requesting transaction
*D.* Transaction may acquire a lock in RO-mode No other transaction holds the lock in X- or S-mode, andAll transactions that retain the lock in X- or S-mode are ancestors of the requesting transaction.
*E.* When a transaction aborts, it releases all locks it holds or retains. If any of its superiors holds or retains any of these locks they continue to do so.
*F.* When a subtrans action commits, its parent inherits the locks it held or retained.
*G.* After that the parent retains the locks in the same mode (X or S) as held or retained by the child earlier. If the parent already retains the lock, it keeps the more restrictive mode (multiple retainment rule).

## IV. DISTRIBUTED COMMITMENT

In the transaction process, all the processes that participate in a distributed transaction on whether to commit or abort (roll back) the transaction. A distributed commit protocol known as the two-phase commit protocol (2PC) is used to ensure data consistency. The recovery subsystem is used to restore the database to a consistent state on the restoration of the failed communication links. In a normal execution of any single distributed transaction, i.e., when no failure occurs, which is typically the most frequent situation, the protocol comprises two phases:

*A. Voting Phase*

The commit-request phase (or voting phase), in which a coordinator process attempts to prepare all the transaction's participating processes (named participants, cohorts, or workers) to take the necessary steps for either committing or aborting the transaction and to vote, either "Yes": commit (if the transaction participant's local portion execution has ended properly), or "No": abort (if a problem has been detected with the local portion), and

*B. Decision Phase*

The commit phase (Decision Phase) , in which, based on voting of the cohorts, the coordinator decides whether to commit (only if all have voted "Yes") or abort the transaction (otherwise), and notifies the result to all the cohorts. The cohorts then follow with the needed actions (commit or abort) with their local transactional resources (also called recoverable resources; e.g., database data) and their respective portions in the transaction's other output (if applicable).The two-phase commit (2PC) protocol should not be confused with the two-phase locking (2PL) protocol, a concurrency control protocol.

The greatest disadvantage of the two-phase commit protocol is that it is a blocking protocol. If the coordinator fails permanently, some cohorts will never resolve their transactions: After a cohort has sent an agreement message to the coordinator, it will block until a commit or rollback is received.

## V. RECOVERY

All systems are prone to failures, and handling recovery from failure is a must. The properties of the generated schedules, which are dictated by the concurrency control mechanism, may have an impact on the effectiveness and efficiency of recovery. For example, the Strictness property (mentioned in the section Recoverability above) is often desirable for an efficient recovery. Process of restoring database to a correct state in the event of a failure is called database recovery. Recovery is always caused by a failure. A typical classification of failures distinguishes between transaction failures, system (site) failures, media failures and communication failures.

## VI. FAILURES IN A DISTRIBUTED SYSTEM

A distributed system consists of two kinds of components: sites, which process information, and communication links, which transmit information from site to site. A distributed system is commonly depicted as a graph where nodes are sites and undirected edges are bi-directional communication links.

*A. Site Failures*

When a site experiences a system failure, processing stops suddenly and the contents of volatile storage are destroyed. In this case, we‘ll say the site has failed. When the site recovers from a failure it first executes a recovery procedure, which brings the site to a consistent state so it can resume normal processing.

*B. Communication link Failures*

Communication links are also subject to failures. Such failures may processes at different sites from communicating. A variety of prevent communication failures are possible: A message may be corrupted due to noise in a link; a link may malfunction temporarily, causing a message to be completely lost; or a link may be broken for a while, causing all messages sent through it to be lost.

Message corruption can be effectively handled by using error-detecting codes, and by retransmitting a message in which the receiver detects an error. Loss of messages due to transient link failures can be handled by retransmitting lost messages. Also, rerouting can reduce the probability of losing messages due to broken links.

*C. Recovery with Two-Phase Commit*

Distributed systems are subject to different types of failure, and a transaction processing system needs a recovery algorithm in order to ensure correct execution. Two phase commit (2PC) is the most popular recovery protocol.

The algorithm works as follows:

The coordinator sends a vote_request to all the participants.

receiving a vote_request, a participant responds with its vote, which can be a Yes or a No. The participant aborts if its vote is a No.

If all the votes were Yes, the coordinator sends COMMITs to all the participants that voted Yes. Otherwise, the coordinator sends ABORTs to all the participants that voted Yes.

A participant waits for a COMMIT or an ABORT and decides accordingly.

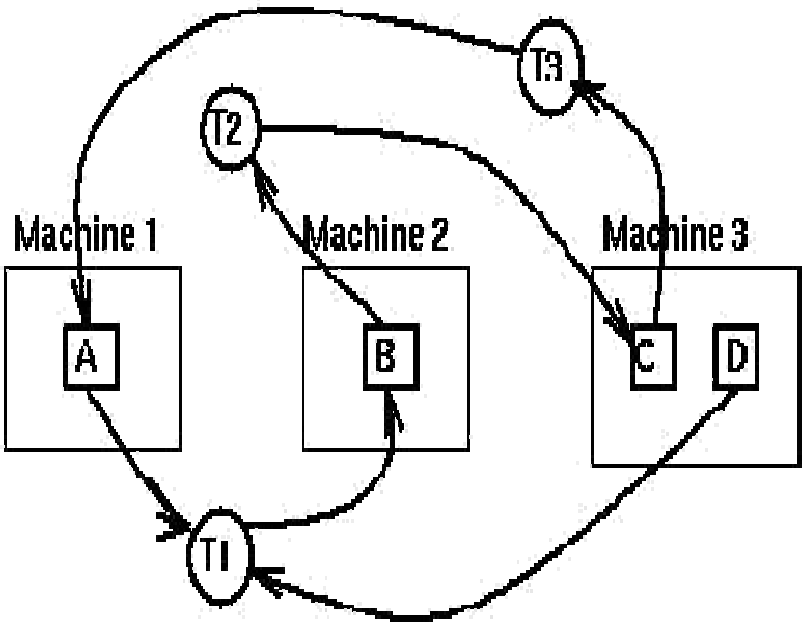
A participant‘s uncertainty period starts after step 2, and ends in step 4 when it receives a COMMIT or ABORT. If the participant times out while waiting for the vote_request, the participant unilaterally decides ABORT. If the coordinator times out while waiting for votes, the coordinator decides ABORT, and sends ABORT to all the participants that voted Yes. If the participant times out while waiting for the COMMIT or ABORT, the participant executes a *termination protocol*. The simplest termination protocol is to block until communication with the coordinator is restored, and then retrieves the COMMIT or ABORT decision. A cooperative termination protocol can result in less blocking than the simple termination protocol, and works as follows. Assume that the coordinator appends the list of participants to its vote_request. The participant sends a decision_request to every other participant. A participant sends a COMMIT or ABORT if it knows the decision, does nothing if it is uncertain, and sends an ABORT if it has not voted yet. The cooperative termination protocol can still block if all the participants are uncertain and the coordinator has failed.

In order to be able to recover correctly from a failure, the coordinator and the participants must record every Yes, No, COMMIT or ABORT message on stable storage before sending it. Site failures as well as communication failures can be handled by, using the 2PC recovery protocol.

## VII. DISTRIBUTED DEADLOCK

A Distributed system consists of a collection of sites that are interconnected through a communication network each maintaining a local database system. The same conditions for deadlocks in uniprocessors apply to distributed systems. Unfortunately, as in many other aspects of distributed systems, they are harder to detect, avoid, and prevent. Distributed deadlocks can occur in distributed systems when distributed transactions or concurrency control is being used.

We have the wait-for graph (WFG) of distributed deadlock.




In the above WFG that has a directed cycle, thus we have a distributed deadlock. A deadlock is a fundamental problem in distributed systems. A process may request resources in any order, which may not be known a priori and a process can request resource while holding others. If the sequence of the allocations of resources to the processes is not controlled, deadlocks can occur. There are four strategies for dealing with distributed deadlocks:

1) Ignorance: ignore the problem (this is the most common approach).
2) Detection: let deadlocks occur, detect them, and then deal with them.
3) Prevention: make deadlocks impossible.
4) Avoidance: choose resource allocation carefully so that deadlocks will not occur.

## VIII. APPROACHES FOR DEADLOCK DETECTION FOR DISTRIBUTED SYSTEMS:

Deadlock detection requires examination of the status of process-resource interactions for presence of cyclic wait. Deadlock detection in distributed systems seems to be the best approach to handle deadlocks in distributed systems. The basic algorithm for distributed deadlock detection is as follows:

1) Step 1: Create the local wait for graph (WFG).
2) Step 2: Add possible edges obtained from other sites that may cause deadlock.
3) Step 3: The local WFG now also contains locks on remote objects and the sub transaction holding those lock.
4) Step 4: determine the cycles, if it is to be found. There is a deadlock.

## IX. ALGORITHM FOR DEADLOCK

### A. *Path-Pushing Algorithms*

The basic idea underlying this class of algorithms is to build some simplified form of global WFG at each site. For this purpose each site sends its local WFG to a number of neighboring sites every time a deadlock computation is performed. After the local data structure of each site is updated, this updated WFG is then passed along, and the procedure is repeated until some site has sufficiently complete picture of the global situation to announce deadlock or to establish that no deadlocks are present. The main features of this scheme, namely, to send around paths of the global WFG, have led to the term *path-pushing algorithms.*

### B. *Edge-Chasing Algorithms*

The presence of a cycle in a distributed graph structure can be verified by propagating special messages called *probes* along the edges of the graph. Probes are assumed to be distinct from resource request and grant messages. When the initiator of such a probe computation receives a matching probe, it knows that it is in cycle in the graph. A nice feature of this approach is that executing processes can simply discard any probes they receive. Blocked processes propagate the probe along their outgoing edges.

## X. DISTRIBUTED DEADLOCK PREVENTION

Deadlock prevention protocols ensure that the system will never enter into a deadlock state. The basic prevention strategies are:
The strategies require that each transaction lock all its data item before it begins execution.
They impose partial ordering of all data item and require that a transaction can lock data item only in the order specified by the partial order.
An alternative to detecting deadlocks is to design a system so that deadlock is impossible. One way of accomplishing this is to obtain a global timestamp for every transaction (so that no two transactions get the same timestamp). When one process is about to block waiting for a resource that another process is using, check which of the two processes has a younger timestamp and give priority to the older process.
If a younger process is using the resource, then the older process (that wants the resource) waits. If an older process is holding the resource, the younger process (that wants the resource) kills itself. This forces the resource utilization graph to be directed from older to younger processes, making cycles impossible.
This algorithm is known as the wait-die algorithm.
An alternative method by which resource request cycles may be avoided is to have an old process preempt (kill) the younger process that holds a resource. If a younger process wants a resource that an older one is using, then it waits until the old process is done. In this case, the graph flows from young to old and cycles are again impossible. This variant is called the wound-wait algorithm.

## XI. CONCLUSION

In this paper, we explore the emerging field related to distribute concurrency control. We discuss advance transaction process called nested transaction and issues of nested transaction.

Also in this paper we discussed the techniques for recovery from transaction failures. The main goal of recovery is to ensure the atomicity property of a transaction. If a transaction fails before completing its execution, the recovery mechanism has to make sure that the transaction has no lasting effects on the database. We first gave an informal outline for a recovery process and then discussed system concepts for recovery.

A distributed system consists of two kinds of components: sites, which process information, and Communication links, which transmit information from site to site.

A Distributed system consists of a collection of sites that are interconnected through a communication network each maintaining a local database system. The same conditions for deadlocks in uniprocessors apply to distributed systems.

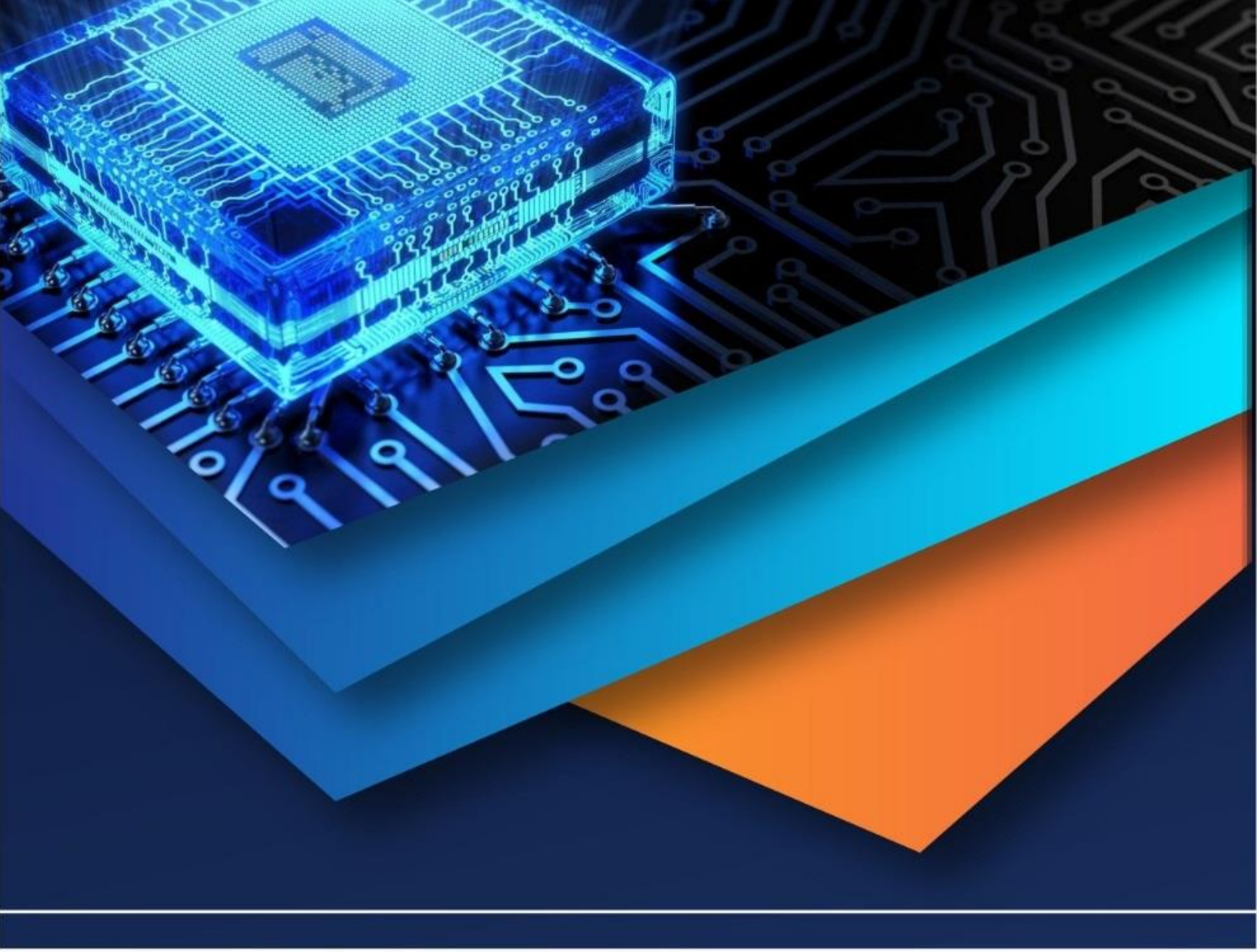
Deadlock prevention protocols ensure that the system will never enter into a deadlock state.

Deadlock detection in distributed systems seems to be the best approach to handle deadlocks in distributed systems. In this chapter two deadlock algorithm are discussed *Path-pushing algorithms* and *Edge-chasing algorithms*

In the transaction process, all the processes that participate in a distributed transaction on whether to commit or abort (roll back) the transaction. A distributed commit protocol known as the two-phase commit protocol (2PC) is used to ensure data consistency.

## REFERENCES

[1] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman.Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.
[2] Lance Hammond, Brian D. Carlstrom, Vicky Wong, Ben Hertzberg, Mike Chen, Christos Kozyrakis, and Kunle Olukotun.Programming withtransactiona coherence and consistency (tcc). In ASPLOS-XI:Proceedings of the 11th international conference on Architectural support for programming languages and operating systems, pages 1{13. ACM Press, Oct 2004
[3] Christos Kotselidis, Mohammad Ansari, Kim Jarvis, Mikel Luj_an, Chris Kirkham, and Ian Watson. Designing a distributed softwaretransactional memory system. In ACACES '07: 3rd InternationalSummer School on Advanced Computer Architecture and Compilation for Embedded Systems, July 2007.
[4] Database Systems,Second Edition. Prentice-Hall, 1999.
[5] Concurrency Control in Distributed Database Systems By PHILIP A.
[6] Badal D. Z., Correctness of Concurrency Control and Implications inDistributed Databases, Proceedings of COMPSAC 79 Conference, Nov1979.
[7] Badal D. Z., Correctness of Concurrency Control and Implications inDistributed Databases, Proceedings of COMPSAC 79 Conference, Nov1979
[8] “Distributed Transaction Processing on an Ordering Network” ByRashmiSrinivasa, Craig Williams, Paul F. Reynolds
[9] Transaction Processing in Distributed Databases.By ManpreetKaur
[10] Transaction Processing in Distributed Databases.By ManpreetKau
[11] “Distributed databases Principals & Systems”,Stefano Ceri, GinseppePelagatti, McGrawHill Book Company, 1984.
[12] Denning, Dorothy E. et al., ―Views for Multilevel Database Security,‖ In IEEE Transactions on Software Engineering, vSE-13 n2, pp. 129-139, February 1987.
[13] Denning, Dorothy E. et al., ―Views for Multilevel Database Security,‖ In IEEE Transactions on Software Engineering, vSE-13 n2, pp. 129-139, February 1987.
[14] Denning, Dorothy E. et al., ―Views for Multilevel Database Security,‖ In IEEE Transactions on Software
[15] Herbert, Andrew, ―Distributing Objects,‖ In Distributed Open Systems, F.M.T. Brazier and D. Johansen eds., pp. 123-132, Los Alamitos: IEEE Computer Press, 1994
[16] ―Illustrate Object Relational Database Management System,‖ Informix white paper from the Illustra Document Database, 1996
[17] Jajodia, Sushil and Ravi Sandhu, ―Polyinstantiation Integrity inMultilevel Relations,‖ In Proceedings IEEE Symposium on Research in
[18] Jajodia, Sushil and Ravi Sandhu, ―Polyinstantiation Integrity inMultilevel Relations,‖ In Proceedings IEEE Symposium on Research in
[19] Concurrency Control in Distributed Transaction Process‖ by JitendraSheetlani and manojjangde
[20] Agrawal D., El Abbadi A. and Lang A. E., The Performance ofProtocols Based on Locks with Ordered Sharing, IEEE Transactionson Knowledge and Data Engineering 6/5, Oct 1994
[21] Adya A., Gruber R., Liskov B. and Maheshwari U., Efficient OptimisticConcurrency Control Using Loosely Synchronized Clocks,Proceedings of the ACM SIGMOD International Conference on the Management of Data, May 1995
[22] Alsberg P. A. and Day J. D., A Principle for Resilient Sharing ofDistributed Resources, Proceedings of the 2nd International Conferenceon Software Engineering, Oct 1976.
[23] ANSI X3.135-1992, American National Standard for InformationSystems Database Language —SQL, Nov 1992.
[24] Bayer R., Heller H. and Reiser A., Parallelism and Recovery inDatabase Systems, ACM Transactions on Database Systems 5/2, Jun1980.
[25] ANSI X3.135-1992, American National Standard for InformationSystems Database Language —SQL, Nov 1992.
[26] Sharing, IEEE Transactionson Knowledge and Data Engineering 6/5, Oct 1994.
[27] Adya A., Gruber R., Liskov B. and Maheshwari U., Efficient OptimisticConcurrency Control Using Loosely Synchronized Clocks,Proceedings of the ACM SIGMOD International Conference on the Management of Data, May 1995.
[28] Alsberg P. A. and Day J. D., A Principle for Resilient Sharing ofDistributed Resources, Proceedings of the 2nd International Conferenceon Software Engineering, Oct 1976.
[29] ANSI X3.135-1992, American National Standard for InformationSystems Database Language —SQL, Nov 1992.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY