VGS Algorithm: An Efficient Deadlock Prevention Mechanism for Distributed Transactions using Pipeline Method

Menka Goswami
Deptt. of Comp. Sc. & Engg.
BT KIT, Dwarahat-263656
Dist – Almora (UK), INDIA

Kunwar Singh Vaisla
Deptt. of Comp. Sc. & Engg.
BT KIT, Dwarahat-263656
Dist – Almora (UK), INDIA

Ajit Singh
Deptt. of Comp. Sc. & Engg.
BT KIT, Dwarahat-263656
Dist – Almora (UK), INDIA

ABSTRACT
Deadlock is one of the most serious problems in database system. The deadlock problem becomes further complicated if the underlying system is distributed. Distributed deadlock prevention has been studied to some extent in distributed database systems. This paper introduces brief overview of the most recent algorithm for deadlock prevention. The main objective of this paper is to provide an improvement over other deadlock prevention algorithms. Executing the transactions requesting for same resources in pipeline fashion has been discussed which efficiently prevents deadlocks and mechanism for reducing the waiting time of the requesting transactions has also been discussed.

Keywords
VGS, Deadlock, WFG, Transactions, Resources

1. INTRODUCTION
Deadlock refers to the coordination and concurrency problem where two or more processes are waiting indefinitely for the release of a shared resource [1], [2].

The deadlock problem involves a circular waiting where one or more transactions are waiting for resources to become available and those resources are held by some other transactions that are in turn blocked until resources held by the first transaction are released. [3, 4].Deadlock processes never terminate their executions and the resources held by them are not available to any other process.

Deadlock is an undesirable situation; some of the consequences of deadlock have been listed:

1. Throughput of the system is affected.
2. Performance of system suffers.
3. Deadlock in real time applications is not appreciable at any cost.
4. Entire or partial system is crippled.
5. Utilization of the involved resources decreases to zero.
6. Deadlock increases with deadlock persistence time.
7. Deadlock cycles do not terminate by themselves until properly detected and resolved.
8. No progress

These are the consequences of deadlock; let's have a look at some reasons why deadlock occurs in a resource allocation system:

1. Deficiency of system resources.
2. An inappropriate execution order of processes.
3. Improper resource allocation logic [5]

Deadlock is a highly unfavorable situation, in deadlock situations the whole system or a part of it remains indefinitely blocked and cannot terminate its task. Therefore it is highly important to develop efficient control and scheduling algorithm to optimize the system performance while preventing deadlock situations [6]. Deadlock control mechanisms prevent the system from entering a state that may lead to catastrophic failure.

At present, deadlock is a well known problem in many contemporary technological systems such as automated manufacturing systems, distributed systems etc... In all of these systems deadlock is a highly undesirable state which causes the entire or partial system to cripple also the utilization of the involved resource decreases to zero [7]. An effective way to prevent the occurrence of deadlock can be appropriately restricting the allocation of system resources to various requesting processes. Since deadlock state is the one that cannot evolve anymore, the first step to prevent deadlock should be to identify deadlock states and prevent them from occurring [8]. Basically the necessary four conditions for deadlock to occur which are known as Coffman conditions [9], are:

Mutual exclusion condition: A resource cannot be used by more than process at a time.

Hold and wait condition: Processes that already hold resource may request new resource.

Non preemption condition: No resource can be forcibly removed from a process that holds it, and resources can be released only by the explicit action of the process.

Circular wait condition: Two or more processes form circular chain where each process waits for a resource that the next process in the chain holds.

Basically the first three conditions are decided by the physical characteristics of a system and its resources. In other words, for a system with a given set of resources, the first three conditions are either true or false. However, we can work on the fourth condition and vary it depending on request, allocation, and release of the resources in the system. A deadlock occurs, if all of the four conditions are true. On the contrary, a deadlock will never occur if one of these conditions is not satisfied [5].

To maintain database consistency the condition of mutual exclusion is important.

The hold and wait condition can be removed if all of the required resources are allocated to the processes for the execution. Although the mechanism is conservative and may lead to low resource utilization [5] but is helpful in preventing deadlocks.

The pre emption of a resource in database applications is not preferable.

Circular wait prevention allows processes to wait for resources but ensures that the waiting is not circular.

The occurrence of deadlock degrades system performance. Although all of the four conditions are necessary for deadlock to occur but if one of them is negated then the occurrence of deadlock can be prevented in distributed transactions system. The first three deadlock condition always holds and the only possible way to prevent deadlock is to falsify the condition of the circular wait condition [5].

The algorithm proposed in the paper successfully eliminates the circular wait condition of processes over the same set of shared resources. The best part of the algorithm is that it also prevents hold and waits condition without causing any low resource utilization. In fact it allows the resources to utilize the same set of shared resources in an effective way.

2. RELATED WORK

Deadlock has been a very serious problem in all contemporary technological sectors distributed systems, automated manufacturing systems, operating systems etc and has been extensively studied. Basically there are four strategies to handle deadlocks: deadlock prevention, deadlock avoidance, deadlock detection, deadlock resolution.

Deadlock detection, resolution techniques and deadlock avoidance technology typically require pre empting resources, aborting processes or centralized resource management so they are inappropriate for many distributed real time systems that require the execution time of processes be predictable [10]. Therefore it would be more efficient if we look for an efficient deadlock prevention mechanism for distributed real time systems.

The deadlock avoidance strategies make system performance to suffer [1,11], although deadlock detection may be effective, but it costs a lot in business transaction services [12] and detection of false deadlocks leads to wastage of resources. However deadlock resolution mechanisms are there but many real time applications are not well suited for run time deadlock resolution [13].
Although the deadlock prevention strategies are considered to be conservative and less feasible but in safety critical systems in which deadlocks may lead to serious results and [5] economic losses only deadlock prevention mechanisms can prove to be miraculous, for example in a highly automated semiconductor manufacturing system, the amount of time that wafers stay at a chamber is absolutely crucial. The occurrence of a deadlock extends the sojourn of wafers at chambers, possibly making all the wafers in such chambers scrapped. [5] As a result, it is essential to ensure that deadlock will never occur.

Now a system cannot be made completely deadlock free, if resource sharing is high, but the occurrence of deadlocks can be prevented using deadlock prevention mechanisms.

A major advantage of deadlock-prevention algorithms is that they require no runtime cost since problems are solved in system design and planning stages [5].

In [14] describes certain theorems which propose sequence structures for processes and the resource allocation methods. It emphasizes on sequential allocation to prevent deadlocks in tool sharing systems.

In [15] a service interaction model has been designed and the deadlock problem related with shared internet resources has been analyzed. It has proposed 2 solutions for deadlock prevention: 1) to arrange an order for all shared web resources.2) if one or more requested resource is busy transaction releases the resources it got before and restarts.

Work of [10] suggests a deadlock prevention technique that involves requiring that managers in the RTC runtime system meet the AND-OR request conditions. This prevents deadlock in systems where processes can make AND-OR requests.

[16] Describes a deadlock prevention mechanism which is based on atomic transaction approach to co allocate grid resources.

It is well known that the deadlock can be prevented if a global linear ordering of all the resources are defined and each application are required to secure its resources one by one in an increasing according to this ordering. This approaches based on resource order are called ODP2 (Order-based deadlock prevention protocol) [17]. ODP2 is not efficient enough though correct. Applications with ODP2 are susceptible to long waits and congestion. [16] Therefore design of an algorithm is required which reduces the waiting time of transactions and the congestion at sites in distributed system.

As far as our knowledge goes little work has been done on deadlock prevention in distributed transactions.

The paper [12] describes deadlock prevention algorithm for both local and global deadlocks.

For local deadlock it has proposed replica based mechanism and for global deadlock it has proposed an improvement over 2 phase commit protocol. It introduced pre check phase to the 2 phase commit protocol.

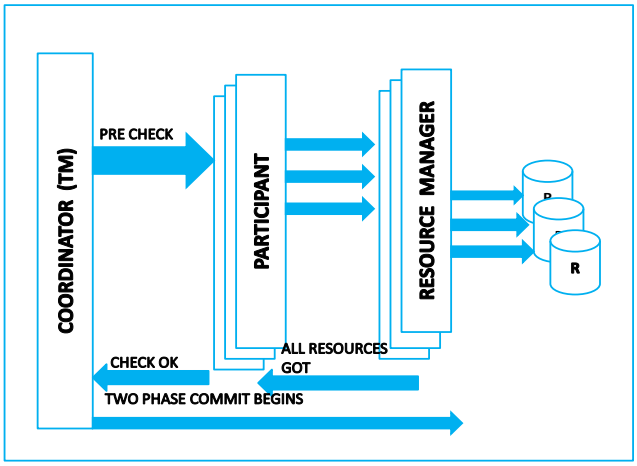

Fig 1: Pre check has been added to 2PC protocol.

It can be seen from the figure that at the first phase the coordinator delivers all the user's requests to the participants/sub transactions, then these participants communicate with resource manager to check the resource if they are available.

If the resources are available then the participants hold them at the same time and return ok to the coordinator. It can be

seen as a try lock phase that means a transactions can go on if the resource lock is obtained, otherwise it should return false. On receiving positive feedback from participants the coordinator enters in the standard 2 phase commit protocol process.

The deadlock prevention steps of [12] have been described as follows:

Step 1.Transaction manager will receive user's task and produce a global unique root transaction id. This id can be a function of current time to distinguish which transaction starts earlier. TM divides the task into sub-transactions and distributes them onto different sites which host specified transaction service.

Step 2. Each participant will receive pre-check instruction and it begins to try to get all the needed resources from resource manager. Suppose T1A from root transaction T1 requests R1 from resource manager RM1 and T2A from T2 requests R2 from RM2. Resource managers cache their root transaction ids separately. They both successfully obtain the locks of required resources. Then, sub-transaction T1B starts to acquire the lock of R2 by sending request to RM2. RM2 checks its cache that if any transaction has the same root id with T1B. If not, it then notifies T1B that it cannot access resource R2 because it's locked. However, T1B will not be blocked but returns pre-check failure information to T1's manager.

Step 3. If the coordinator receives positive checked messages from all participants, it decides to send a request to-prepare message to each of them, and the two-phase commit begins. Otherwise, it should be regarded as not meeting its prerequisite to continue. So the T1 decides to give up, and T1A stops to go ahead by releasing its lock of R1. It's free of deadlock that if some sub-transaction T2B of T2 starts to request the resource R1, it can acquire the lock successfully. This ensures that T2 can continue its work without global deadlock.

3. PROBLEM SPECIFICATION

The deadlock prevention algorithm [12] has two limitations which need to be considered.

Firstly, consider the case of transactions T1, T2.

T1A requests R1 from resource manager RM1. T1 locks R1.

T2A requests R2 from resource manager RM2. T2 locks R2.

Now when T1B requests R2 from RM2, RM notifies that it cannot access R2. Now T1B returns pre check failure information to the T1's manager. As such T1 is regarded as not meeting its pre requisite to continue and T1 gives up.

If same transaction happens to release resources whenever resource conflict occurs it may cause live lock, to solve this problem a time stamp based mechanism has been used in [12].

Now, when a transaction is at its pre check phase only the resource availability is checked at sites and the transaction is initiated, it is not checked whether the required resources are free or not. After entering the two phase commit protocol if any of the sub transaction is not being able to attain lock on any of the resource the transactions gives up, as it is using 2PC protocol. A live lock may happen if the transaction restarts again but still cannot acquire needed resources. To solve this problem, time-stamp based mechanism has been used to choose which transaction should quit when resource competing occurs [12].

Although some mechanism is being used to choose the transaction that will give up but the problem is that someone (transaction) will give up. All the work done up till now will be a waste. And it makes the execution time of processes unpredictable which is not acceptable.

Secondly, the transactions competing for the same resources have to wait till the earlier resource holding transactions commit successfully. It increases their waiting time and it increases the congestion at sites as in the pre check of the transaction only the resource availability at the sites was checked. It was not checked whether the resources are free or not. As such the number of transactions that are requesting other transactions to release the resource will increase. In such cases some of them may be in hold and wait and may cause deadlock!

Our algorithm proposes a different approach to handle the limitations. Firstly it does not allow the transactions to lock the resources in pre check phase. The resources will be held by the resource manager of respective sites till the 2nd phase. As such for every transaction the resources will appear free in their pre check phase and they will start. And when they will reach the 2nd phase of the 2 PC they will hold the resources as requested by them. Secondly it allows the sub transactions to execute in pipeline method if the transactions are requesting for the same set of resources. It reduces the waiting time of transactions and the congestion at sites.

Working of the algorithm has been described in the upcoming fourth section.

Some idea has been taken from the paper [12], our work on the paper presents an improvement over the algorithm and the two phase commit protocol. It works not only over the prevention of deadlocks but also reduces the waiting time of the transactions who are requesting for the same resources. Our algorithm uses the concept of pipelining so that the transactions requesting for the same resources would execute in pipeline fashion cooperating each other.

4. VGS ALGORITHM FOR DEADLOCK PREVENTION

The proposed algorithm requires that the prior knowledge of the resources is necessary. Although most of the researchers do not appreciate the concept of prior knowledge of resources but in distributed applications where the transactions follow two phase commit protocol the prior knowledge of resources would be very beneficial. The prior knowledge of resources would let the transaction to be sub divided into sub transactions and distribute them at different sites as per the availability of resources as such the hold and wait condition would not occur.

Most of the distributed transaction applications use two phase commit protocol to coordinate sub transactions [12]. The sub transactions need to lock the required resources before updating data to ensure that the transactions will commit at all sites or at none of them.

The proposed algorithm prevents the deadlock in following ways:

1. The pre check phase, in which sub transactions are divided only when all of the required resources are available and free. As such none of any transaction will be waiting for resource. When they will be initiated at sites they will not be waiting for any

resource. Hold and wait condition has been eliminated therefore no deadlock.

2. The transaction will be divided into sub transactions only when resources are available and will commit and the sub transactions (not of same transaction) requesting for same resource will execute in pipeline fashion. As such all of the sub transactions are active they are not in waiting state. Sites will be less populated by those transactions which are in waiting or requesting mode there by reducing the chances of formation of deadlock cycles. All of the transactions will be in some phase out of the four phases (pre check, coordinator, phase 1, and phase 2) and executing.

And the algorithm effectively reduces the waiting time of the transactions requesting for the same resource.

The proposed algorithm has been explained as follows:

Suppose there is a transaction Ti initiated at site Sr, the coordinator of Sr is Cr and the resource manager is RMr.

And R {} be the set of required resources by Ti, now Ti is going to follow 2PC protocol. So it will be divided into sub transactions according to the resource availability.

PRE CHECK PHASE: The Ti will sub divide into sub transactions only when the resources are available at the sites and if they are available the sub transactions will initiate at the sites.

Suppose the sub transactions are $T[i_m][j]$……$T[i_n][k]$. After initiating suppose sub transaction $T[i_m][j]$ initiates at site Sj requesting for resource Rm. Now $T[i_m][j]$ will ask RMj to hold the resources for it. $T[i_m][j]$ will not lock the resource. In fact $T[i_m]$ [j] does not lock resource until it enters phase 2.

Here Ti refers to transaction i^{th}.

$T[i_m][j]$ refers to the sub transaction m of transaction i initiated at site j.

COORDINATOR PHASE: RMj is holding the resources, till that time the $T[i_m]$ [j] informs its coordinator Cj that the resources are available. Just like $T[i_m][j]$ all other sub transactions of Ti will inform their respective site coordinators about the availability of resources.

PHASE 1: Cr will send a request to prepare message to all the participating site coordinators {sites at which sub transactions of T[i] have initiated}.

//from here now the term participating site coordinators refers to sites where the sub transactions of the respective transaction initiated.

$T[i_m]$ [j] enters in phase 1 and site coordinator Cj responds <ready T[i]> message to the coordinator Cr. Like $T[i_m][j]$ all other sub transactions' of Ti will send <ready T[i]> message to coordinator Cr. On receiving positive response to its prepared message Cr decides that it should commit.

PHASE 2: As soon as $T[i_m]$ [j] enters phase 2 it locks the resource and executes successfully. Like $T[i_m]$ [j] all other sub transactions of Ti finally commit.

Now suppose when $T[i_m]$ [j] initiated at Sj a sub transaction $T [i+1_s]$ [j] of transaction T [i+1] also initiated at Sj.

Now the resource manager will check if $T[i+1_s][j]$ is also requesting for the same resource requested by $T[i_m][j]$ then $T[i+1_s][j]$ will execute in pipeline fashion with $T[i_m][j]$ cooperating each other to commit successfully. It means $T [i+1_s]$ [j] is also requesting for the same resource which RMj is holding for $T[i_m]$ [j]. Now since $T [i+1_s]$ [j] and $T [i_m]$ [j] are requesting for the same resource they will execute in pipeline fashion. When $T [i+1_s]$ [j] will inform its coordinator till then $T[i_m]$ [j] would have entered in phase 1. As soon as $T[i_m][j]$ enters its phase 1 it sends ready<T[i]> message in response of request to prepare message to coordinator Cr, then it enters phase 2 and till then $T[i+1_s][j]$ enters phase 1. In phase 2, on receiving positive response to its request to prepare message Cr sends <commit T[i] > message to all the participating coordinators. Therefore when $T[i_m]$ [j] enters phase 2, it commits successfully and resource is now given to $T [I+1_s]$ [j] because it requested. After sending the <ready T > message to coordinator $T [i+1_s]$ [j] enters phase 2 and transaction T [i+1] commits.

And if the resources requested are not the same the T[i+1] [j] will execute normally.

The algorithm has been divided into two parts, part 1 describes that how a sub transaction will execute in phases and the second part describes that how sub transactions will execute when they want to access the same set of resources. Handling of two sub transactions requesting for same resources of different transactions has been described in 2nd part of algorithm.

VGS ALGORITHM FOR DEADLOCK PREVENTION

TRANSACTION (Ti, Ti+1….Tn),

SITES Sj…….Sk,

RESOURCES Rm…….Rs,

COORDINATORS Cj…….Ck,

RESOURCE MANAGERS RMj…….RMk

BEGIN,

Suppose a transaction Ti is initiated at site Sr where site coordinator is Cr and resource manager is RMr

Let R {} be the set of required resources by Ti to execute successfully

Do,

PART 1:

// the first part of the algorithm describes that how a sub transaction will execute in phases.

Pre check phase: In the pre check phase Ti is subdivided in sub transactions on the basis of resource availability at different sites i.e. resources should be available.

Subtransaction $T[i_m][j]$……$T[i_n][k]$ will initiate at site S[j]……S[k] if all resource in R{} are available at S[j]…….S[k].

Here **i** represent the transaction number i.e. 1, 2, 3……

j…k represents the site at which sub transaction of T[i] initiated.

R represents the resource which $T [i_m]$ requires and is available at site [j].

Suppose $T [i_m][j]$ initiates at site Sj requesting resource R

End of pre check phase.

Coordinator phase:

{

T [i_m][j] requests RMj to hold the resource R, it doesn't lock it. RMj will hold the resources till that time T [i_m] [j] informs its coordinator that the resources are available.

T [i_m][j]will not lock the resource until it enters phase 2,till then RMj will hold the resource.

Like T [i_m][j] all other sub transactions of Ti will inform their respective site coordinators.

End of coordinator phase:

Phase 1:Sending of the request to prepare message by Cr to all the participating site coordinators (where sub transactions of Ti initiated) marks the beginning of the phase 1 of sub transactions of T [i].The site coordinator Cj responds <READY T[i]> message to coordinator Cr. T[i_m] is in phase 1.

Like T [i_m][j] all other sub transactions of Ti will send the <READY T[i]> message to their coordinators.

End of phase 1.

Phase 2: Beginning of phase 2 is marked by the receipt of positive responses to the prepare message of the coordinator Cr. It decides to commit and sends a <commit T[i]> message to all the participating site coordinators (sites where sub transactions of T[i] are executing). T [i_m][j] locks the resource R and executes successfully.

End of phase 2.

Like T [i_m][j] all other sub transactions execute and Ti finally commits

}

PART 2:

//the 2nd part of algorithm describes the deadlock prevention method when a same set of resource is being requested by two transactions

Now suppose transaction T [i+1] initiated at site Cp and let R' {} be the required set of resources.

Pre check phase of T[i+1]:

Suppose T [$i+1_s$] ……..T [$i+1_n$] be the sub transactions in which T [i+1] is sub divided as per the availability of resources in their pre check phase at different sites.

Now suppose when T [i_m][j](a sub transaction of T[i]) requested RMj to hold the resource at site Sj, a sub transaction T[$i+1_s$][j] also initiated at site Sj requesting resource X .

End of pre check phase of T [i+1].

{

Coordinator phase of T[i+1]:

T [$i+1_s$] will request RMj to hold resource X.

The resource manager will check

IF (resources requested by T [i_m][j]) **INTERSECTION** (resources requested by T[$i+1_s$][j])==**null**

Then resource manager will hold the resources requested by T[$i+1_s$][j]and T[$i+1_s$][j] can proceed normally.

ELSE

{

It means T [$i+1_s$][j] is also requesting R.

Resource manager is holding resource R for T [i_m][j] as it requested earlier so it can not grant resource to T[$i+1_s$][j].

T[$i+1_s$][j] will neither be blocked nor it will be sent back instead RMj will ask T[$i+1_s$][j] to cooperate with T[i_m][j] and begin the transaction in **pipeline fashion.**

As the T [i_m][j] begins phase 1 ,T[$i+1_s$][j] will inform its coordinator that resources are available.

End of coordinator phase of T [i+1].

Like T [$i+1_s$][j] all other sub transactions of Ti+1 will inform their site coordinators.

Phase 1 of T [$i+1_s$]:

Cp (the site coordinator where Ti+1 initiated) will send a request to prepare message to all the participating site coordinators (sites where sub transactions of T [i+1] initiated).

As the T [i_m][j] enters phase 2, T[$i+1_s$][j] enters phase 1 and the site coordinator Cj responds <READY T[i+1]> message to coordinator Cp. T[$i+1_s$] is in phase 1.

Like T [$i+1_s$][j] all other sub transactions of Ti+1 will send the <READY T[i+1]> message to their coordinators.

End of phase 1 of T[$i+1_s$].

Phase 2 of T [$i+1_s$]:

On receiving positive response to its prepare message the coordinator Cp decides that it should commit and sends commit<T [i+1]> message to all the participating site coordinators (sites where sub transactions of T [i+1] are executing).

As now T [i_m][j] has successfully executed and is now committed the resource R is free and now T[$i+1_s$][j] will lock it and complete its phase 2.

End of phase 2 of T[$i+1_s$].

Since the entire sub transactions of Ti+1 have successfully executed Ti+1 commits.

}

}

The flowchart describes the phases of the transaction T[i_m] and T [$i+1_s$] as already been discussed in the algorithm. T[i] will initiate at a site and sub divide into sub transactions only when the required resources will be free and available at sites. Although it is not possible for all the resources to be free simultaneously. Therefore in our approach when a transaction wishes to access the resource it asks RM to hold the resource, it (transaction) itself does not lock the resource as such the resources will appear free to the other transactions, although they will acquire and lock the resources in their second phase. This mechanism not only prevents deadlock but also reduces the waiting time of other waiting transactions by making them to execute in pipeline fashion.

The flowchart shows that how a sub transaction will execute after initiating at a site (i.e. after accomplishing its pre check phase). The sub transaction enters its coordinator phase where it asks resource manger RM to hold the resources here T[i_m]

initiates at site Sj and asks RMj to hold the resources. The resources have not been locked by transaction $T[i_m]$ instead RMj is holding them for $T[i_m]$. $T[i_m]$ will acquire resources and lock them in its second phase of 2PC.

As $T[i_m]$ initiates at site Sj and enters in its coordinator phase requesting RMj to hold the required resource R suppose a sub transaction $T[i+1_s]$ of transaction T[i+1] initiates at site Sj. It also requests RMj to hold required resource. Now RMj will check if the requested resource by $T[i_m]$ and $T[i+1_s]$ are the same then they should execute in pipeline fashion. Executing in pipeline fashion will not affect consistency of database as when $T[i_m]$ will acquire lock on resource it will be in its 2nd phase and $T[i+1_s]$ will be in 1st phase, and when $T[i+1_s]$ will enter 2nd phase the resource R will be already freed by $T[i_m]$.

If the requested resource by $T[i+1_s]$ is not R then it will execute normally. The requested resource will be held by RMj so that any other transaction requesting for same resource could also initiate.

Transactions requesting for same resources have to wait for the resource holding transaction to release resource as they will use the value updated by the earlier transaction which held the resource. So if we are able to eliminate the waiting time then the deadlocks can be prevented to a much larger extent. If the resource managers of site hold the resources requested by a transaction than any other transaction requiring the same resource will also be able to initiate since transaction has not locked them (resources) instead RM is holding the requested resources for respective transactions which requested them. So they will find resources free and will initiate and in distributed applications which use 2PC protocol. This concept of VGS algorithm can be implied as it will help not only in efficient execution of transactions requesting for same resources, will also help in reducing waiting time, less population of transactions which are in hold and wait condition and hence less formation of deadlock cycles.

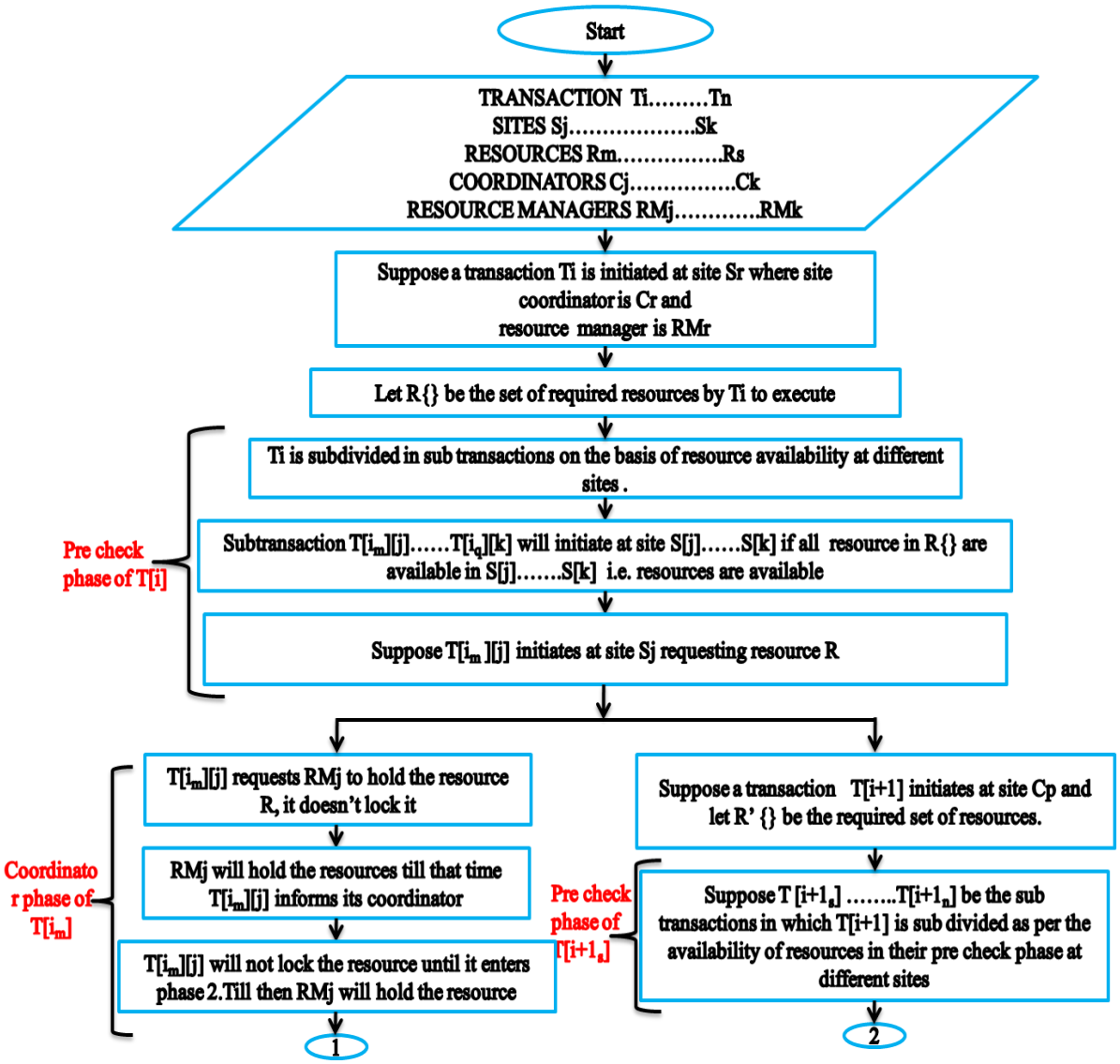

Fig 2(i): flow chart of the working of the VGS algorithm.

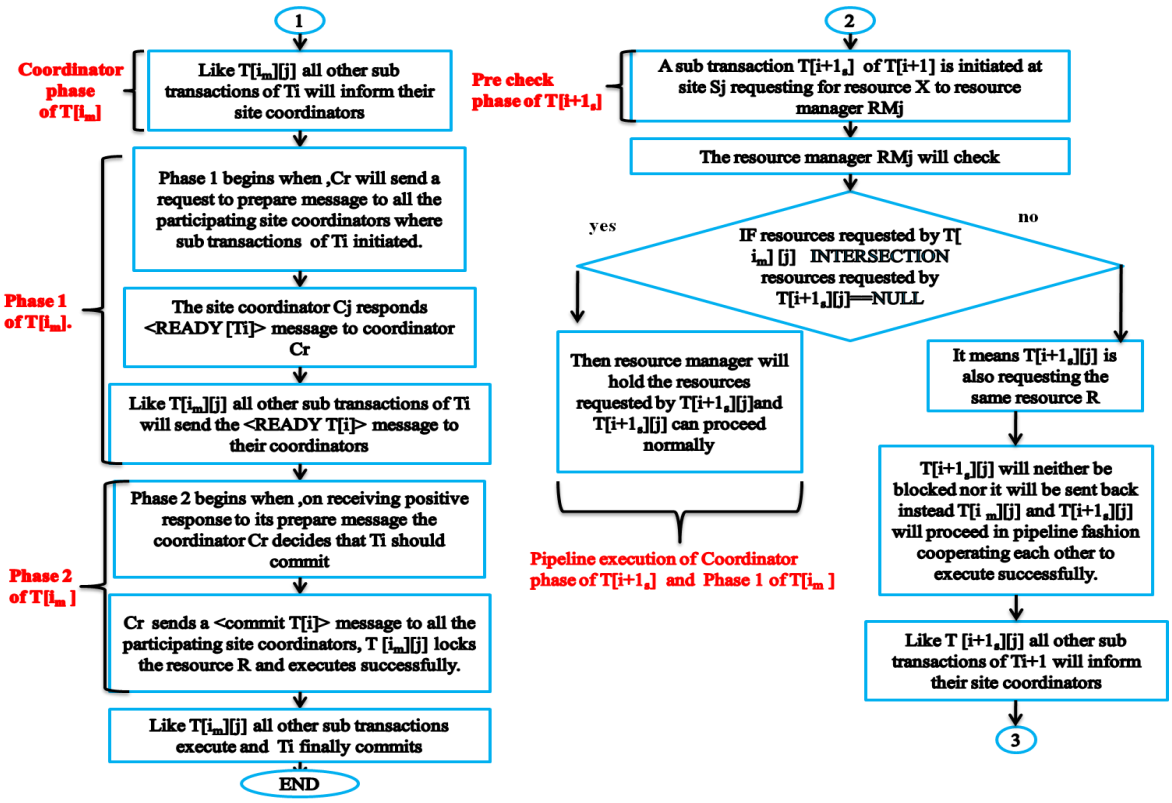

Fig 2(ii): flow chart of the working of the VGS algorithm.

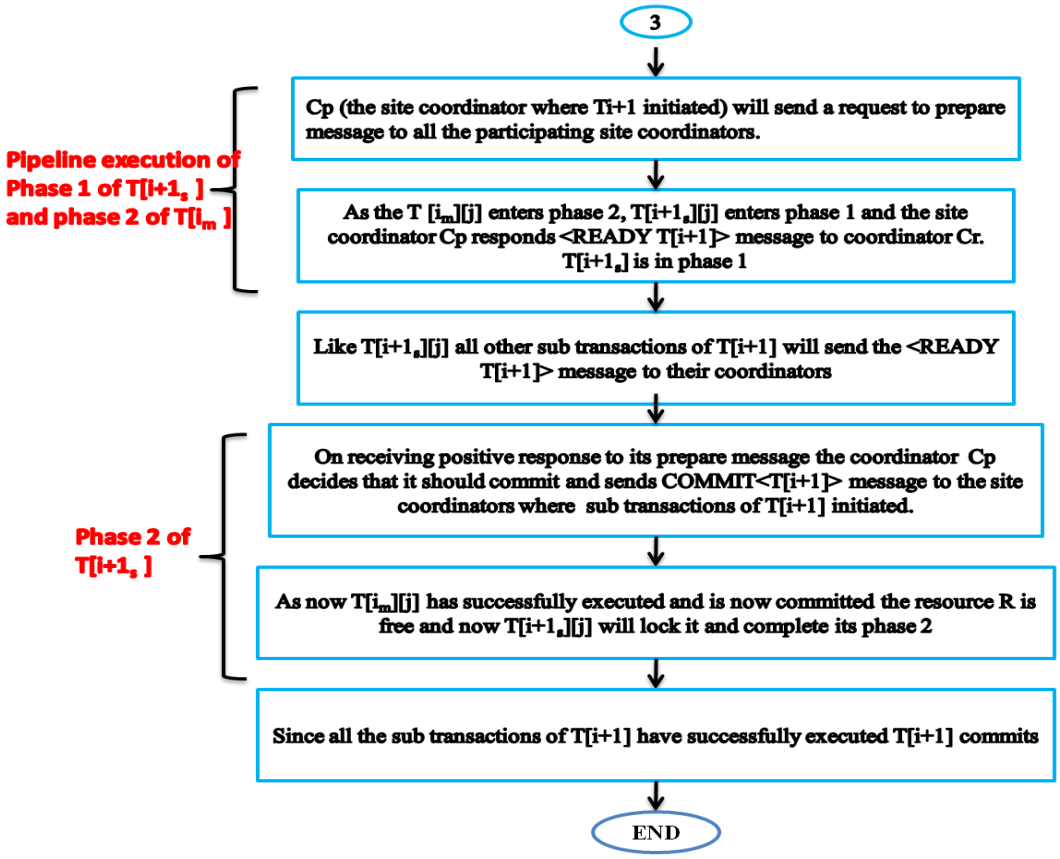

Fig 2(iii): flow chart of the working of the VGS algorithm.

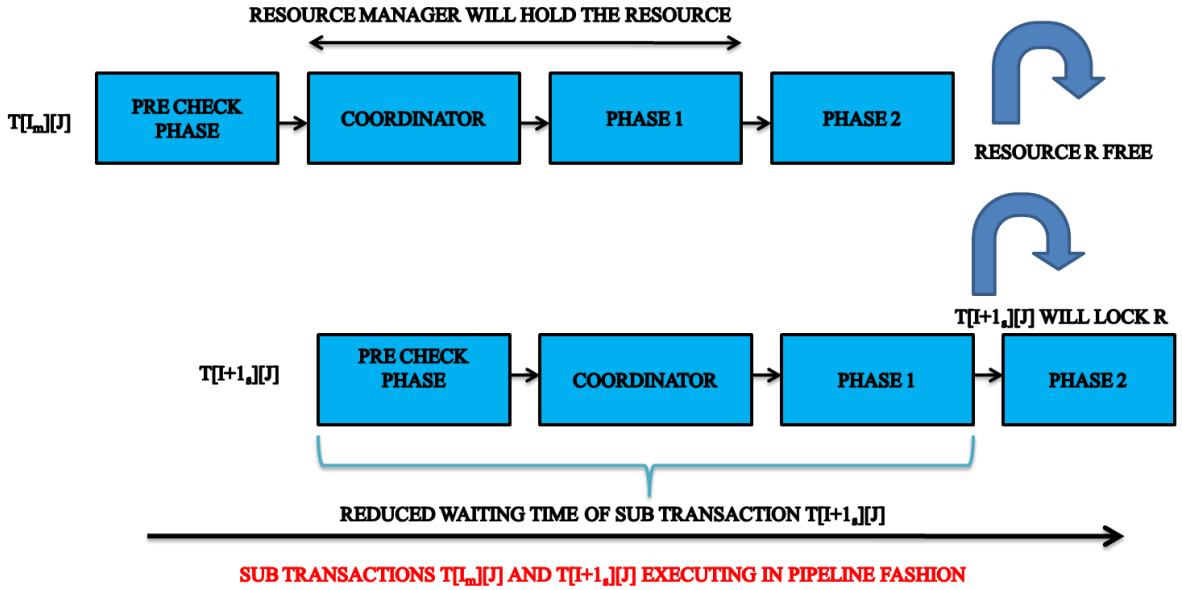

Fig 3: Diagrammatic representation of execution of sub transactions in phases.

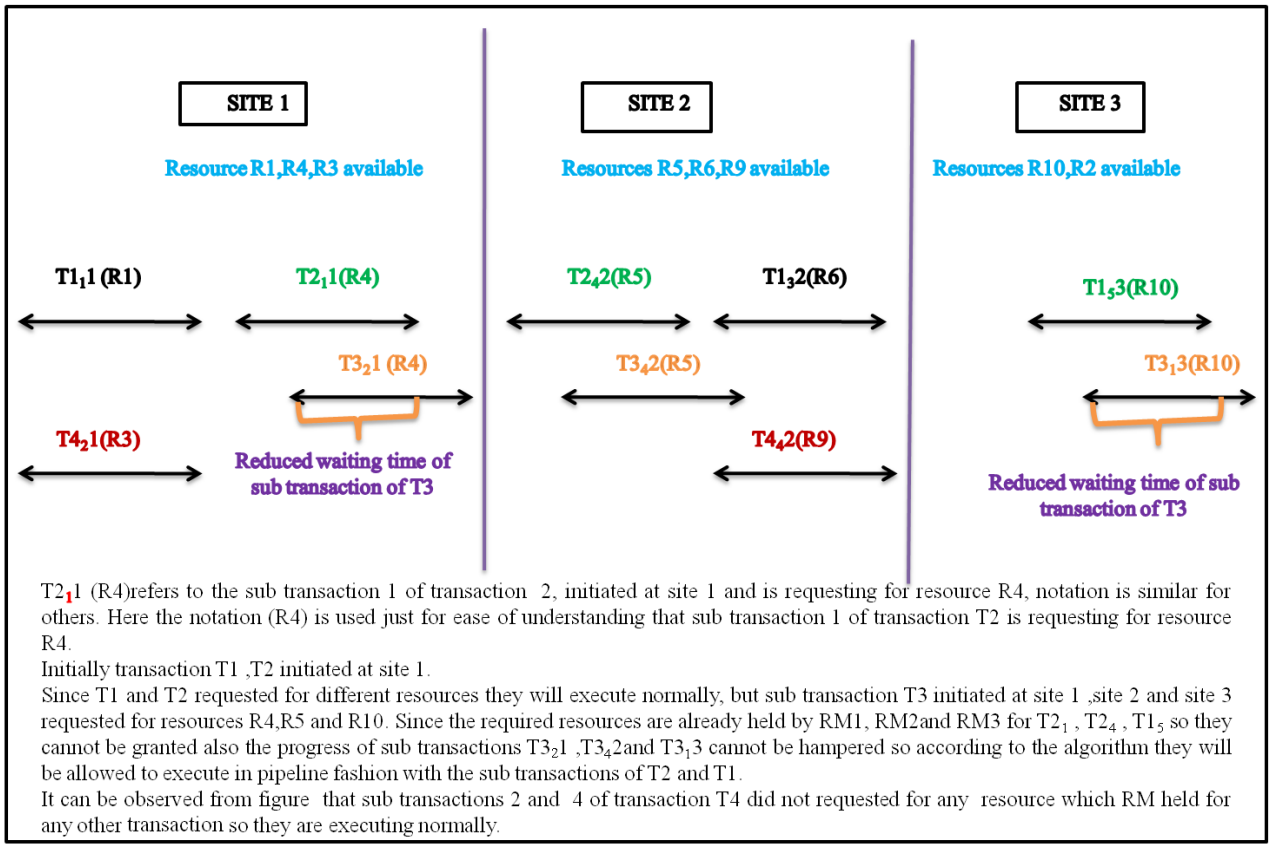

Fig 4: The diagrammatic representation of the execution of various transactions at different sites in pipeline fashion under 2PC protocol.

Above figure represents the diagrammatic representation of how the sub transaction of a transaction will execute in phases. there are Four phases: i) Pre check phase ii) Coordinator phase iii) Phase 1 of 2pc protocol iv) Phase 2 of 2pc protocol. Figure depicts that how the sub transactions of different transactions requesting for same resources (here R) will execute in pipeline fashion. Figure clearly shows that the resource manager will hold the resource till phase 1, then in phase 2 the resource will be acquired by the respective sub transaction and will lock it.

The advantage of the algorithm is that even if the sub transactions of the 2 or more transactions initiate at site S and request for the same resources then the algorithm lets the

transactions to commit successfully in pipeline fashion and reduces their waiting time i.e. it handles global deadlock properly and prevents the deadlock.

The limitation of the algorithm is that it cannot handle two sub transactions of the same transaction requesting for the same resources.

Therefore this algorithm is more preferable in cases where two phase commit protocol is being used to coordinate the sub transactions and the requesting transaction do not have more than one sub transaction requesting for the same resource at same site.

The algorithm has no affects in database consistency because all sub-transactions are proceeding sequentially but in a pipeline fashion.

5. CONCLUSION

The paper reviews the literature of the deadlock prevention mechanisms and proposes a deadlock prevention mechanism for distributed system applications which use two commit protocol. It effectively prevents deadlocks by eliminating circular wait condition and prevents the hold and wait condition. It reduces the waiting time of the transactions for the resources; in fact the algorithm executes transactions in a pipeline fashion and successfully enables the transactions to share the same set of resources.

6. REFERENCES

[1] U. Kapasi, W. Dally, S. Rixner, J. Owens, and B. Khailany, "The Imagine stream processor," Proc. International Conference of Computer Design, 2002, 282–288.

[2] D. Zobel, "The Deadlock problem: a classifying bibliography,"ACM SIGOPS Operating Systems Review, vol. 17, October 1983.

[3] H.M.Deite1,"An Introduction to Operating Systems", Addision-Wesley Company, Second Edition, 199003-8575-6/04, IEEE.

[4] A.D.Kshemkalyani and M. Singhal, **"A** One-Phase Algorithm to Detect Distributed Deadlocks in Replicated Databases", IEEE Transaction on Knowledge and Engineering, Vol. 11, No.6, November December, 1999.

[5] ZhiWu Li, NaiQiWu, and MengChu Zhou, "Deadlock Control of Automated Manufacturing Systems Based on Petri Nets—A Literature Review", IEEE transactions on systems, man, and cybernetics—part c: applications and reviews, Digital Object Identifier 10.1109/TSMCC.2011.2160626, IEEE, 2011.

[6] KeYi Xing, LiBin Han, MengChu Zhou and Feng Wang,"Deadlock-Free Genetic Scheduling Algorithm for Automated Manufacturing Systems Based on Deadlock Control Policy", IEEE transactions on systems, man, and cybernetics—part b: cybernetics, Digital Object Identifier 10.1109/TSMCB.2011.2170678, IEEE.2011.

[7] T. Murata, "Petri nets: properties, analysis and application," *Proceedings of IEEE*, vol. 77, no. 4, pp. 541–579, April 1989.

[8] Hesuan Hu, Zhiwu Li, Mengchu Zhou, "Two Generalized-Petri-net-based Strategies for Deadlock Prevention in Resource Allocation Systems", IEEE, 2008.

[9] E. G. Coffman, M. J. Elphick, and A. Shoshani, "Systems deadlocks,"*ACM Comput. Surv*, vol. 3, no. 2, pp. 66–78, 1971.

[10] Victor Fay Wolfe, Susan Davidson & Insup Lee, "Deadlock Prevention in the RTC Programming System for Distributed Real-Time Applications", IEEE, 1993.

[11] S. Venkatesh, J. Smith, "An evaluation of deadlock-handling strategies in semiconductor cluster tools," IEEE Trans.Semiconductor Manufacturing, vol 18, pp. 197-201, 2005.

[12] Lin Lou1, Feilong Tang, Ilsun You, Minyi Guo, Yao Shen, Li Li ,"An Effective Deadlock Prevention Mechanism for Distributed Transaction Management", 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing DOI 10.1109/IMIS.2011.109 IEEE Computer society,2011.

[13] Lei Gao, Gaurav Mittal, David Zaretsky, and Prith Banerjee, "Resource Optimization and Deadlock Prevention while Generating Streaming Architectures from Ordinary Programs",2011 NASA/ESA conference on adaptive hardware and systems(AHS- 2011), IEEE,2011.

[14] Nagi Z. Gebraeel and Mark A. Lawley, "Deadlock Detection, Prevention, and Avoidance for automated Tool Sharing Systems", IEEE transactions on robotics and automation, vol. 17, no. 3, June 2001.

[15] Jieqi Ding, Han Zhu_, Huibiao Zhu and Qin Li "Formal Modeling and Verifications of Deadlock Prevention Solutions in Web Service Oriented System", 2010 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems DOI 10.1109/ECBS.2010.48,IEEE computer society, 2010.

[16] Zhang Chuanfu Liu Yunsheng Zhang Tong Zha Yabing Huang Kedi, "A Deadlock Prevention Approach based on Atomic Transaction for Resource Co-allocation", Proceedings of the First International Conference on Semantics, Knowledge, and Grid (SKG 2005), 2006, IEEE.

[17] Jonghun Park, "A Deadlock and Livelock Free Protocol for Decentralized Internet Resource Co-allocation", IEEE Transactions on systems, man and cybernetics – Part A: systems and humans, Vol. 34, No. 1, and January 2004