Distributed Deadlock Detection

Preliminaries:

The problem of deadlocks has been generally studied in distributed systems the models:

- The systems have only reusable resources
- Processes are allowed only exclusive access to resources
- There is only one copy of each resource

A process can be in two states:

- running(active state): a process has all the needed resources and is either executing or is ready for execution
- blocked: a process is waiting to acquire some resources

Contd.

- Resource vs. Communication Deadlocks
- A Graph-Theoretic Model

Deadlock Handling Strategies in Distributed Systems:

There are three strategies to handle deadlocks:

- deadlock prevention
- deadlock avoidance
- deadlock detection

Contd.

Issues in Deadlock Detection and Resolution

Consider two basic issues:

- Detection

  A correct deadlock detection algorithm must satisfy the following two conditions:

  -Progress-No undetected deadlocks

  -Safety-No false deadlocks

- Resolution

Contd.

Control Organizations for Distributed Deadlock Detection

- Centralized Control
- Distributed Control
- Hierarchical Control

Centralized Control

A designated site (control site) has the responsibility of constructing the global WFG and searching it for cycles. The control site may maintain the global WFG constantly or it may build it whenever a deadlock detection is to be carried out by soliciting the local WFG from every site. Centralized deadlock detection algorithms are conceptually simple and are easy to implement.

Contd.

However in this type of algorithms have a single point of failure. Communication links near the control site are likely to be congested because the control site receives WFG information from all other sites. Also the message traffic generated due to deadlock detection activity.

Distributed Control

In distributed deadlock detection algorithms, the responsibility for detecting a global deadlock is shared equally among all sites. The global state graph is spread over many sites and several sites participate in the detection of a global cycle. Unlike centralized control, distributed control is not vulnerable to a single point of failure and no site is swamped with deadlock detection activity. Also, a deadlock detection is initiated only when a waiting process is suspected to be a part of a deadlock cycle.

Contd.

This type of algorithms are difficult to esign due to the lack of globally shared memory-

Sites may collectively report the existence of a global cycle after seeing its segments at different instants. Unlike centralized control, several sites in distributed control may initiate deadlock detection for the same deadlock. Also, the proof of correctness is difficult for these algorithms. In addition, deadlock resolution is often cumbersome in distributed deadlock detection algorithms, as several sites can detect the same deadlock and not be aware of the other sites or processes involved in the deadlock.

Contd.

Hierarchical Control

In these algorithms sites are arranged in a hierarchical fashion and a site detects deadlocks involving only its descendant sites. Hierarchical algorithms exploit access patterns local to a cluster of sites to efficiently detect deadlocks. They tend to get the best of both the centralized and the distributed control organizations in that there is no single point of failure(as in centralized control) and a site is not bogged down by deadlock detection activities with which it is not concerned (as sometimes happens in distributed control). However hierarchical deadlock detection algorithms require special care while arranging the sites in a hierarchy.

Contd.

Centralized Deadlock-Detection Algorithms

The Completely Centralized Algorithm

This is the simplest centralized deadlock detection algorithm where a designated site called the control site maintains the WFG of the entire system and checks it for the existence of deadlock cycles. All sites request and release resources(even local resources) by sending request resource and release resource messages to the control site respectively. When the control site receives a request resource or a release resource message, it correspondingly updates its WFG. The control site checks the WFG for deadlocks whenever a request edge is added to the WFG.

Contd.

This algorithm is conceptually simple and easy to implement. Unfortunately it is also highly inefficient because all resource acquisition and release requests must go through the control site, even when the resource is local. This results in large delays in responding to user requests, large communication overhead, and the congestion of communication links near the control site. Reliability is poor because if the control site fails, the entire system comes to a halt because all the status information resides at the control site.

Contd.

Several problems of this algorithm(such as large response time and the congestion of communication links near the control site) can be mitigated by having each site maintain its resource status (WFG) locally and by having each site send its resource status to a designated site periodically for construction of the global WFG and the detection of deadlocks. However due to inherent communication delays and the lack of perfectly synchronized clocks the designated site may get an inconsistent view of the system and detect false deadlocks.

Contd.

For example, suppose two resources R_1 and R_2 are stored at sites S_1 and S_2, respectively. Suppose the following two transactions T_1 and T_2 are started almost simultaneously at sites S_3 and S_4, respectively:

T_1	T_2
lock R_1	*lock* R_1
unlock R_1	*unlock* R_1
lock R_2	*lock* R_2
unlock R_2	*unlock* R_2

Contd.

Suppose that the *lock(R_1)* request of T_1 arrives at S_1 and locks R_1 followed by the *lock(R_1)* request of T_2, which waits at S_1. At this point S_1 reports its status $T_2 \rightarrow T_1$ to a designated site. Thereafter T_1 unlocks R_1, T_2 locks R_1, T_1 makes a *lock(R_2)* request to S_2, T_2 unlocks R_1 and makes a *lock(R_2)* request to S_2. Now suppose that the *lock(R_2)* request of T_2 arrives at S_2 and locks R_2 followed by the *lock(R_2)* request of T_1 which waits at S_2. At this point S_2 reports its status. $T_1 \rightarrow T_2$ to the designated site which after constructing the global WFG, reports a false deadlock $T_1 \rightarrow T_2 \rightarrow T_1$.

Contd.

The Ho-Ramamoorthy Algorithm

Ho and Ramamoorthy gave two centralized deadlock detection algorithms, called two-phase and one-phase algorithms to fix the problem of just said. These algo respectively collect two consecutive status tables at each site to ensure that the control site gets a consistent view of the system.

Contd.

The Two-Phase Algorithm

Here every site maintains a status table that contains the status of all the processes initiated at that site. The status of a process includes all resources locked and all resources being waited upon. Periodically, a designated site requests the status table from all sites, constructs a WFG from the information received and searches it for cycles. If there is no cycle then the system is free from deadlocks, otherwise, the designated site again requests status tables from all the sites and again constructs a WFG using only those transactions which are common to both reports. If the same cycle is detected again, the system is declared deadlocked.

Contd.

It was claimed that by selecting only the common transactions found in two consecutive reports, the algorithm gets a consistent view of the system(a view is consistent if it reflects a correct state of the system). If a deadlock exists, it was argued, the same wait-for condition must exist in both reports. However, this claim proved to be incorrect(i.e. a cycle in the wait-for conditions of the transactions common in two consecutive reports does not imply a deadlock) and two-phase algo may indeed report false deadlocks. By getting two consecutive reports, the designated site reduces the probability of getting an inconsistent view, but does not eliminate such a possibility.

Contd.

The One-Phase Algorithm

In this algo. it requires only one status report from each site, however, each site maintains two status tables: a resource status table and a process status table. The resource status table at a site keeps track of the transactions that have locked or are waiting for resources stored at that site. The process-status table at a site keeps track of the resources locked by or waited for by all the transactions at that site. Periodically, a designated site requests both the tables from every site, constructs a WFG using only those transactions for which the entry in the resource table matches the corresponding entry in the process table and searches the WFG for cycles. If no cycle is found, then the system is not deadlocked, otherwise a deadlocked is detected.

Contd.

This algo. does not detect false deadlocks because it eliminates the inconsistency in state information by using only the information that is common to both tables. This eliminates inconsistencies introduced by unpredictable message delays. For ex. if the resource-table at site S_1 indicates that resource R_1 is waited upon by a process P_2(i.e. $R_1 \leftarrow P_2$) and the process table at site S_2 indicates that process P_2 is waiting for resource R_1(i.e. $P_2 \rightarrow R_1$), then edge $P_2 \rightarrow R_1$ in the constructed WFG reflects the correct system state. If either of these entries is missing from the resource or the process table, then a request message or a release message from S_2 to S_1 is in transit and $P_2 \rightarrow R_1$ can not be ascertained.

The one-phase algo. is faster and requires fewer messages as compared to the two-phase algo. However it requires more storage because every site maintains two status tables and exchanges bigger messages because a message contains two tables instead of one.

Contd.

Distributed Deadlock Detection Algorithm

In these algo. all sites collectively cooperate to detect a cycle in the state graph that is likely to be distributed over several sites of the system. These algo. can be initiated whenever a process is forced to wait and it can be initiated either by the local site of the process or by the site where the process waits.

These algo. can be divided into 4 classes

- Path-pushing
- Edge-chasing
- Diffusion computation
- Global state detection

Contd.

In path-pushing algo. the wait for dependency information of the global WFG is disseminated in the form of paths (i.e. a sequence of wait-for dependency edges).

In edge-chasing algo. special messages called probes are circulated along the edges of the WFG to detect a cycle. When a blocked process receives a probe, it propagates the probe along its outgoing edges in the WFG. A process declares a deadlock when it receives a probe initiated by it. An interesting feature of these algo. is that probes are of a fixed size(normally very short).

Contd.

Diffusion computation type algo. make use of echo algo. to detect deadlocks. To detect a deadlock a process sends out query messages along all the outgoing edges in the WFG. These queries are successively propagated through the edges of the WFG. Queries are discarded by a running process and are echoed back by blocked processes in the following way: when a blocked process receives first query message for a particular deadlock detection initiation, it does not send a reply message until it has received a reply message for every query it sent (to its successors in the WFG). For all subsequent queries for this deadlock detection initiation, it immediately sends back a reply message. The initiator of a deadlock detection detects a deadlock when it receives a reply for every query it sent.

Contd.

Global state detection based algo. exploit the following facts:

- A consistent snapshot of a distributed system can be obtained without freezing the underlying computation
- A consistent snapshot may not represent the system state at any moment in time, but if a stable property holds in the system before the snapshot collection is initiated, this property will still hold in the snapshot.

Therefore distributed deadlocks can be detected by taking a snapshot of the system and examining it for the condition of a deadlock.

Contd.

A Path-Pushing Algorithm

In these algo. information about the wait-for-dependencies is propagated in the form of paths. Obermarck's algo. is chosen to illustrate a path-pushing deadlock detection algo. Because it is implemented on the distributed database system R* of the IBM Corporation.

As this algo. was designed for distributed database systems, therefore, processes are referred to as transactions which are denoted by T_1, T_2, ….,T_n. A transaction may consist of several transactions that normally execute at different sites.

Contd.

Obermarck's model assumes that at most one subtransaction within a given transaction can be executing at a time.

Obermarck's algo has two interesting features:

- the nonlocal portion of the global TWF graph at a site is abstracted by a distinguished node (called External or Ex) which helps in determining potential multisite deadlocks withhout requiring a huge global TWF graph to be stored at each site.
- transactions are totally ordered, which reduces the no. of messages & consequently decreases deadlock detection overhead. It also ensures that exactly one transaction in each cycle detects the deadlock.

Contd.

The Algorithm: Deadlock at a site follows the following iterative process:

1. The site waits for deadlock-related information (produced in Step3 of the previous deadlock detection iteration) from other sites. (note that deadlock-related information is passed by sites in the form of paths.)
2. The site combines the received information with its local TWF graph to build an updated TWF graph. It then detects all cycles and breaks only those cycles which do not contain the node ‘Ex’. Note that these cycles are local to this site. All other cycles have the potential to be a part of global cycles.

Contd.

3. For all cycles ‘Ex → T_1 → T_2 → Ex’ which contain the node ‘Ex’ (these cycles are potential candidates for global deadlocks), the site transmit them in string form “Ex, T_1, T_2, Ex’ to all other sites where a subtransaction of T_2 is waiting to receive a message from the subtransaction of T_2 at this site. The algo reduces message traffic by lexically ordering transactions and sending the string ‘Ex, T_1, T_2, T_3, Ex’ to other sites only if T_1 is higher than T_3 in the lexical ordering.

Contd.

Obermarck gave an informal correctness proof of the algo. However the algo is incorrect because it detects phantom deadlocks. The main reason for this is that the portions of TWF graphs that are propagated to other sites may not represent a consistent view of the global TWF graph. This is because each site takes its snapshot asynchronously at Step2.

This algo sends $n(n-1)/2$ messages to detect a deadlock involving n sites. Size of a message is $O(n)$. The delay in detecting the deadlock is $O(n)$.

Contd.

An Edge-Chasing Algorithm

Chandy et al.'s algo uses a special message called a probe. A probe is a triplet(i,j,k) denoting that it belongs to a deadlock detection initiated for process P_i and it is being sent by the home site of process P_j to the home site of process P_k.

A probe message travels along the edges of the global TWF graph and a deadlock is detected when a probe message returns to its initiating process.

Contd.

Some terms and data structures used:

A process P_j is said to be *dependent* on another process P_k if there exists a sequence of processes P_i, P_{i1}, P_{i2}, ..., P_{im}, P_k such that each process except P_k in the sequence is blocked and each process except the first one (P_j) holds a resource for which the previous process in the sequence is waiting. Process P_j is locally dependent upon process P_k if P_j is dependent upon P_k and both the processes are at the same site. The system maintains a boolean array, $dependent_i$, for each process P_i where $dependent_i(j)$ is true only if P_i knows that P_j is dependent on it. Initially $dependent_i(j)$ is false for all i and j.

Contd.

The Algorithm: To determine if a blocked process is deadlocked the system executes the following operations:

if P_i is locally dependent on itself

then declare a deadlock

else for all P_j and P_k such that

a) P_i is locally dependent upon P_j and

b) P_j is waiting on P_k and

c) P_j and P_k are on different sites,

send probe(i, j, k) to the home site of P_k

Contd.

On the receipt of probe (i, j, k) the site takes the following actions:

if

d) P_k is blocked and

e) $dependent_k(i)$ is false and

f) P_k has not replied to all requests of P_j

then

begin

$dependent_k(i)$ =true;

if k = i

then declare that P_i is deadlocked

else for all P_m and P_n such that

a) P_k is locally dependent upon P_m and

b) P_m is waiting on P_n and

c) P_m and P_n are on different sites

send probe (i, m, n) to the home site of P_n

end.

Thus a probe message is successively propagated along the edges of the global TWF graph and a deadlock is detected when a probe message returns to its initiating process.

Contd.

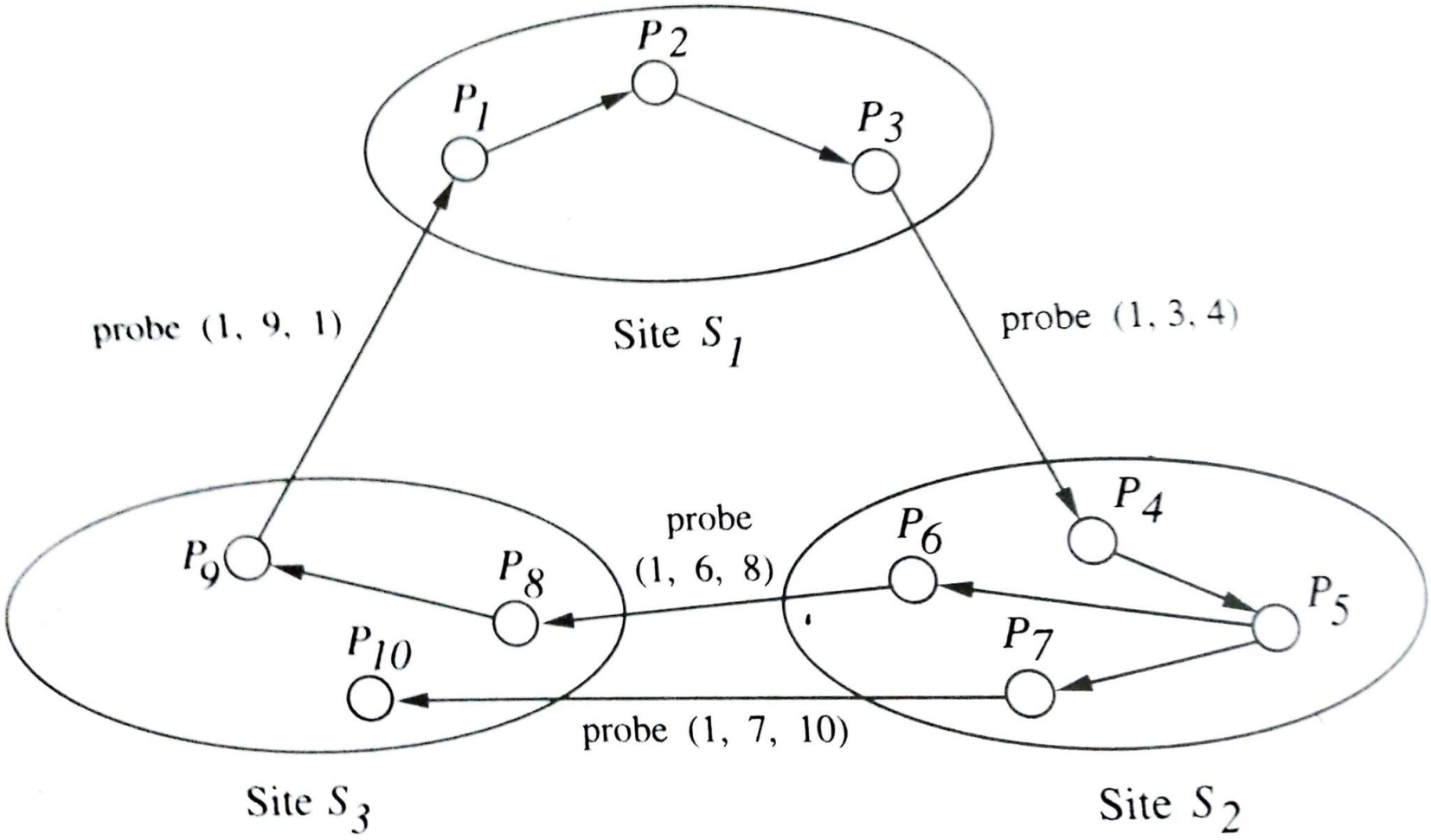

Fig. An example of Chandy et al.'s Edge-Chasing algorithm

Contd.

As an example, consider the previous figure. If process P_1 initiates deadlock detection, it sends probe(1, 3, 4) to S_2. Since P_6 is waiting for P_8 and P_7 is waiting for P_{10}, S_2 sends probes(1, 6, 8) and (1, 7, 10) to S_3 which in turn sends probe(1, 9, 1) to S_1. On receiving probe (1, 9, 1), S_1 declares that P_1 is deadlocked.

Chandy et al.'s algo sends one probe message (per deadlock detection initiation) on each edge of the WFG, which spans two sites. Thus, the algorithm at most exchanges m(n-1)/2 messages to detect a deadlock that involves m processes and spans over n sites. The size of messages exchanged is fixed and very small(only 3 integer words). The delay in detecting the deadlock is O(n).

A Diffusion Computation Based Algorithm

Here deadlock detection computation is diffused through the WFG of the system.

A process determines if it is deadlocked by initiating a diffusion computation. The messages used in diffusion computation take the form of a query(i,j,k) and a reply(i,j,k), denoting that they belong to a diffusion computation initiated by a process P_i and are being sent from process P_j to process P_k. A process can be in two states: active or blocked. In the active state a process is executing and in the blocked state a process is waiting to acquire a resource. A blocked process initiates deadlock detection by sending query messages to all the processes from whom it is waiting to receive a message(these processes are called the dependent set of the process).

Contd.

If an active process receives a query or reply message it discards it. When a blocked process P_k receives a query(i,j,k) message, it takes the following actions:

- if this is the first query message received by P_k for the deadlock detection initiated by P_i(called the engaging query) then it propagates the query to all the processes in its dependent set and sets of local variable $num_k(i)$ to the number of query messages sent.
- if this is not an engaging query, then P_k returns a reply messages to it immediately, provided P_k has been continuously blocked since it received the corresponding engaging query. Otherwise, it discards the query.

Contd.

A local boolean variable $wait_k(i)$ at process P_k denotes the fact that it has been continuously blocked since it received the last engaging query from process P_i. When a blocked process P_k receives a reply(i,j,k) message, it decrements $num_k(i)$ provided $wait_k(i)$ holds. A process sends a reply message in response to an engaging query only after it has received a reply to every query message it sent out for this engaging query messages it had sent out.

Chandy et al.'s Diffusion Computation based Deadlock Detection Algorithm

Initiate a diffusion computation for a blocked process P_i:

send query (i,i,j) to all processes P_j in the dependent set DS_i of P_i;

$num_i(i) := |DS_i|$; $wait_i(i) := true$;

When a blocked process P_k receives a query(i,j,k):

if this is the engaging query for process P_k

then send query (i,k,m) to all P_m in its dependent set DS_k;

$num_k(i) := |DS_k|$; $wait_k(i) := true$

else if $wait_k(i)$ then send a reply(i,k,j) to P_j.

Contd.

When a process P_k receives a reply(i,j,k)

if $wait_k(i)$

then begin

$num_k(i):=num_k(i)-1$;

if $num_k(i)=0$

then if i= k then **declare a deadlock**

else send reply(i,k,m) to the process P_m,

which sent the engaging query.

Contd.

In this algo it is assumed that only one diffusion computation is initiated for a process. In practice several diffusion computations may be initiated for a process. (a diffusion computation is initiated every time the process is blocked). However note that at any time only one diffusion computation is current for any process. All others are outdated. The current diffusion computation can be distinguished from outdated ones by using sequence numbers.

Hierarchical Deadlock Detection

In these algorithms sites are (logically) arranged in hierarchical fashion and a site is responsible for detecting deadlocks involving only its children sites.

The Menasce-Muntz Algorithm

Here all the controllers are arranged in tree fashion(a controller manages a resource or is responsible for deadlock detection). The controllers at the bottom-most level(called leaf controllers) manage resources and others(called nonleaf controllers) are responsible for deadlock detection. A leaf controller maintains a part of the global TWF graph concerned with the allocation of the resources at that leaf controller. A nonleaf controller maintains all TWF graphs spanning its children controllers and is responsible for detecting all deadlocks involving all of its leaf controllers.

Contd.

Whenever a change occurs in a controller's TWF graph due to a resource allocation, wait or release it is propagated to its parent controller. The parent controller makes changes in its TWF graph searches for cycles and propagates the changes upward if necessary. A nonleaf controller can receive up-to-date information concerning the TWF graph of its children continuously (i.e. whenever a change occurs) or periodically.

The Ho-Ramamoorthy Algorithm

In this algo sites are grouped into several disjoint clusters. Periodically a site is chosen as a central control site which dynamically chooses a control site for each cluster. The central control site requests from every control site their intercluster transaction status information and wait-for relations.

As a result a control site collects status tables from all the sites in its cluster and applies the one-phase deadlock detection algo to detect all deadlocks involving only itercluster transactions. Then it sends intercluster transaction status information and wait-for relations(derived from the information thus collected) to the central control site. The central site splices the intercluster information it receives constructs a system WFG and searches it for cycles. Thus a control site detects all deadlocks located in its cluster and the central control site detects all intercluster deadlocks.

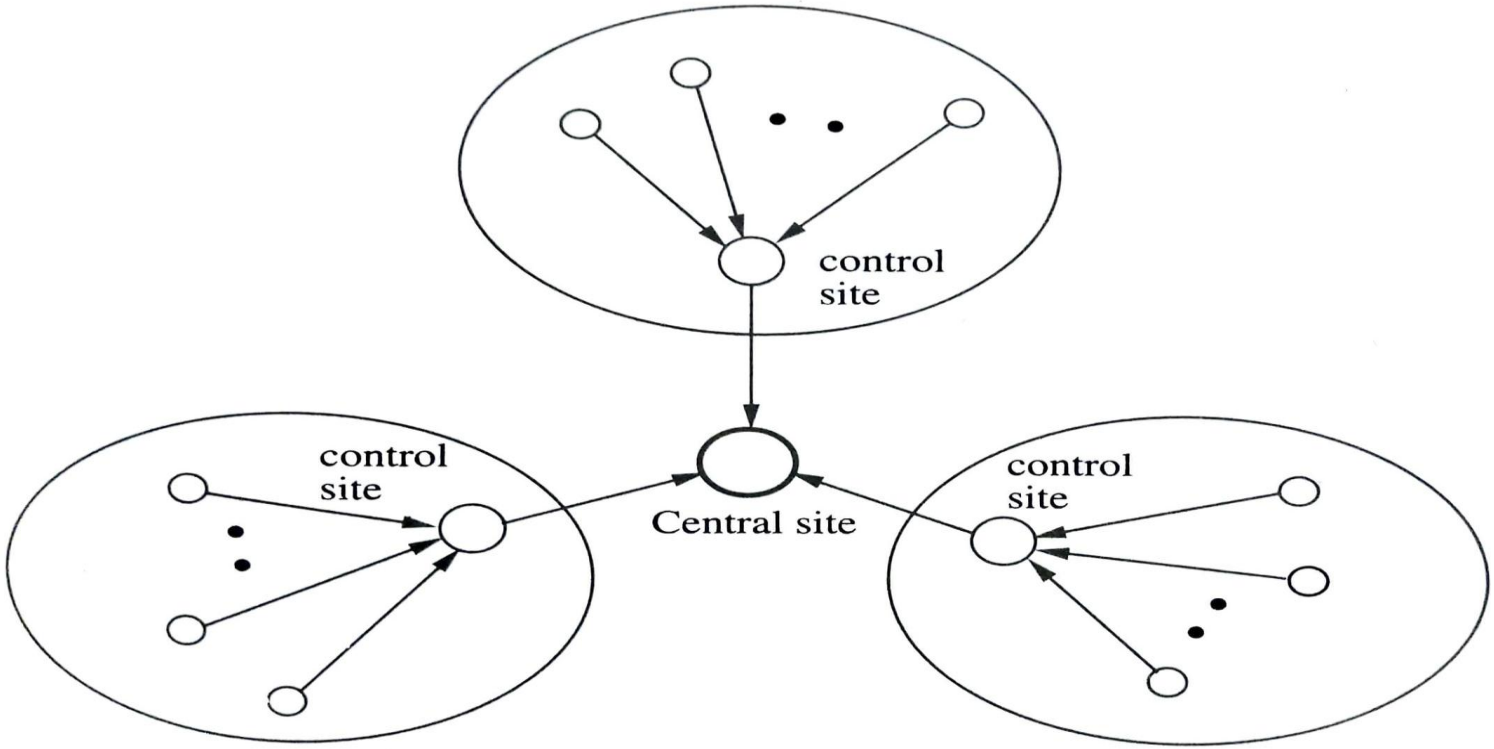

Fig. Hierarchical Organization in the Ho-Ramamoorthy algorithm

Summary

Among three deadlock handling strategies (deadlock prevention, deadlock avoidance, deadlock detection) deadlock detection is the most promising for distributed system. The detection of deadlocks requires performing two tasks: maintaining a WFG and searching the WFG for cycles. Depending upon the way the WFG is maintained and the way a control to carry out the search for cycles is structured, deadlock detection algo are classified into three categories: centralized, distributed and hierarchical.

Contd.

In centralized deadlock detection algorithms the control site has the responsibility of constructing the global state graph & searching it for cycles. Centralized deadlock detection algorithms are conceptually simple and easy to implement. However centralized deadlock detection algo have a single point of failure, communication links near the control site are likely to be congested and the control site may be swamped with deadlock detection activity.

Contd.

In distributed deadlock detection algo every site maintains a portion of the global state graph and every site participates in the detection of a global cycle. In these algo do not have a single point of failure and no site is overloaded by deadlock detection activities. However due to the lack of global shared memory, the design of distributed deadlock detection algo is difficult because sites may report the existence of a global cycle after seeing segments of the cycle at different instants, even though all the segments never existed simultaneously.

Contd.

Hierarchical deadlock detection algo fall between centralized and distributed deadlock detection algo and exploit access patterns local to a cluster of sites to detect deadlocks efficiently. These algo do not have a single point of failure and a site is not bogged down by the deadlock detection activities.

On the three control organizations to detect global deadlocks distributed control is the most widely studied. In distributed deadlock detection algo all sites collectively cooperate to detect a cycle in the state graph which is distributed over several sites of the system. All distributed deadlock detection algo have a common goal to detect cycles which span several sites in distributed manner – yet they differ from each other in the way they achieve this goal.

Contd.

Distributed deadlock detection algo can be divided into four classes:path-pushing,edge-chasing,diffusion computation and global state detection. In path-pushing algo wait-for dependency information of the global WFG is disseminated in the form of paths(sequence of wait-for dependency edges). In edge-chasing algo special message called probes are circulated along the edges of the WFG to detect a cycle. When a blocked process receives a probe it propagates the probe along its outgoing edges in the WFG. A process declares a deadlock when it receives a probe initiated by it. Diffusion computation type algo make use of echo algo to detect deadlock. Deadlock detection messages are successively propagated (diffused) through the edges of the WFG. Global state detection based algo detect deadlocks by taking a snapshot of the system and by examining it for the condition of a deadlock.