International Journal of Art & Humanity Science(IJAHS)
www.ijahs.com Volume 3 Issue 6, (Nov-Dec 2016), PP. 28-33

Study of Deadlocks, Its Detection and Prevention

Mamta Mishra
Azad Institute of Engineering and Technology
mamtamishrait@rediffmail.com

ABSTRACT A distributed system is much larger and more powerful than typical centralized systems due to the combined capabilities of distributed components. Examples of distributed systems include computer networks, distributed databases, distributed information processing systems and real time process control systems. Many challenging areas that span the middleware and language constructs, down to the implementation of the supporting application-level communication protocols and operating system mechanisms have drawn the attention of researchers in the area of distributed systems. The control tasks of operating systems and database systems like mutual exclusion, deadlock and concurrency control, are much more difficult to solve in distributed systems than in a centralized systems. This paper addresses various issues associated with one of the fundamental resource management problems, the deadlock.

INTRODUCTION

Deadlock can occur whenever two or more processes are competing for limited resources and the processes are allowed to acquire and hold a resource (obtain a lock) thus preventing others from using the resource while the process waits for other resources. Two common places where deadlocks may occur are with processes in an operating system (distributed or centralized) and with transactions in a database. The concepts discussed here are applicable to any system that allocates resources to processes. Locking protocols such as the popular Two Phase Locking (see concurrency control) give rise to deadlock as follows: process A gets a lock on data item X while process B gets a lock on data item Y. Process A then tries to get a lock on Y. As Y is already locked, process A enters a blocked state. Process B now decides to get a lock on X, but is blocked. Both processes are now blocked, and, by the rules of Two Phase Locking, neither will relinquish their locks.

This is a deadlock: process A is waiting for a resource held by process B and process B is waiting for a resource held by process A. No progress will take place without outside intervention. Several processes can be involved in a deadlock when there exists a cycle of processes waiting for each other. Process A waits for B which waits for C which waits for A.

Four conditions must hold for deadlock to occur:

1. Exclusive use – when a process accesses a resource, it is granted exclusive use of that resource.
2. Hold and wait – a process is allowed to hold onto some resources while it is waiting for other resources.
3. No preemption – a process cannot preempt or take away the resources held by another process.
4. Cyclical wait – there is a circular chain of waiting processes, each waiting for a resource held by the next process in the chain.

The problem of deadlocks can be handled in several ways: Prevention, Avoidance, and Detection. In prevention, some requirement of the system makes deadlocks impossible so that no runtime support is required. Avoidance schemes require decisions by the system while it is running to insure that deadlocks will not occur. Detection requires the most

sophisticated runtime support: the system must find deadlocks and break them by choosing a suitable victim that is terminated or aborted and restarted if appropriate.

TYPES OF DEADLOCK DETECTIONS

There are three types of deadlock detections:

1. Centralized deadlock detection
2. Distributed deadlock detection
3. Hierarchical deadlock detection

Centralized Deadlock Detection

In this type of detection, a designate site, called the control site, has the responsibility of constructing the global WFG of the system. Cycles are searched by this site and resolved by the control site. It is conceptually simple to implement such a system. Since the control site has the full picture of the system, optimal decisions can be made. However, this approach suffers from drawbacks that centralized deadlock control site is a single point of failure. As the system consists of a larger number of sites and processes, the centralized control site has to serve a larger number of processes, message traffics in that site will be increased and the computational load of the cycle search algorithm will be increased. These affect the performance of the system as well as the stability of it.

Distributed Deadlock Detection

In distributed deadlock detection, processes are responsible to detect the deadlock by themselves. They utilize control messages between the processes to detect deadlocks. This type of detection enjoys the concurrency of the algorithm as well as the tolerance to process failures. However, this type of detection also suffers from a number of drawbacks. First, as the messages between processes are asynchronous and the system is dynamic, a distributed algorithm solving the problem is hard to implement and design correctly. Second, this type of algorithms is not as efficient as the centralized type because no processes can have the full picture of the WFG. Third, all processes need to run the deadlock detection algorithm continuously and concurrently with the underlying computation. While it is possible that the algorithm would use a small portion of the computational resource, it is a performance leak.

Hierarchical Deadlock Detection

In hierarchical deadlock detection, sites are arranged into clusters hierarchically, where sites detect deadlocks that involve only its descendant sites. Hierarchical algorithms tend to get the best out of the two types of deadlock detection algorithms presented above. There is no single point of failure and sites are not going to be overloaded with the deadlock detection algorithm when it is unnecessary for deadlock detection. This kind of algorithm make uses of the access patterns of the system in order to design the hierarchy of the clusters so that deadlocks are as localized in a cluster as possible. This is one of the biggest challenges in implementing such kind of detection algorithm.
In the following section, we are going to discuss some algorithms in distributed deadlock detection.

DISTRIBUTED DEADLOCK DETECTION ALGORITHMS

OBERMARCK'S ALGORITHM

Obermarck's algorithm in deadlock detection is a path-pushing algorithm. Path-pushing algorithms are those that information of the global WFG is distributed in the form of paths. It was developed for the distributed database system R* of the IBM Corporation. In a distributed database system, the computation is done in a set of participating sites. A transaction is an abstraction for the application processing performed to take the database from one consistent state to another consistent state in a way that it can be viewed as atomic. Transactions involve agents in different sites. Therefore, given a site, there can be multiple agents in the site working on different or the same transaction. When the first transaction is waiting for the second transaction and the second transaction is waiting for the first transaction at the same time, the system is said to be deadlocked. the term Transaction Wait-For-Graph (TWFG) is used to stress the difference between transaction model and communication model. However, the mechanism of the graph is roughly the same as WFG.

Each transaction is represented by a globally unique identifier. Since each site consists of a number of agents that belongs to different transactions, the global TWFG is split into smaller parts when viewed by each site. In a given site S, the algorithm uses a non-existing virtual agent EXTERNAL ("EX") to denote external sites that are not S. Only one $T_1 \rightarrow T_2$ edge is created in the TWFG no matter how many times transaction T_1 is waiting for transaction T_2. The algorithm at each site builds and analyzes a directed TWFG, where the nodes represent the transaction agents and the edges denote the wait-for relationships. When an agent is dependent (waiting for) some agents in external site, it is denotes by an edge to "EX". Similarly, when an agent is determined that an agent in external sites is waiting for itself, it is represented by an edge from "EX" to that agent. For example, in a TWFG "EX" $\rightarrow T_1 \rightarrow T_2 \rightarrow$ "EX" denotes a potential global deadlock that involves some external sites, where there is an agent in external sites is waiting for T_1, and T_2 is waiting for some agent in external sites.

The deadlock detectors are roughly synchronized when they exchange control messages.

Chandy-Misra Algorithm

In Chandy-Misra algorithm for deadlock detection, two algorithms are given. First algorithm is for AND models, and second algorithm is for OR models.

1) Algorithm for AND model (Resource Model)

This algorithm is an example of edge-chasing algorithm . For an edge-chasing algorithm, the existence of a cycle in a WFG can be detected by propagating special control messages called probes along the edges of the wait-for-graph. Probes are concerned with the deadlock detection and are distinct from other computation messages. When the initiator *init* receives a probe that is originated from itself, it can be determined that there is a cycle in the graph and thus, a deadlock exists.

2) Algorithm for OR model (Communication Model)

This algorithm is designed for the communication model, where the process can change from idle to executing when it receives some number of replies for the requests sent. This algorithm is an example of diffusion computation based algorithm [8]. The base idea of a diffusion computation based algorithm is that it is activated by a process that suspects a deadlock. Query and reply messages are used to detect the activity of the processes in the

dependent set. Query messages are in the form of (i, m, j, k) which denotes that it is the mth query sent by initiator process Pi, and the query message is sent by process Pj to process Pk. A reply message is also in the form of (i, m, j, k). When the initiator Pi initiates the deadlock detection, it sends query message to all processes in its dependent set. When an active process received a query message, it discards it. When an idle process Pk receives a query message (i, m, j, k), and if the message is received for the first time, called engaging query, then Pk forwards the query message to its dependent set. These properties follow [3]:

Kshemkalyani-Singhal Algorithm

Kshemkalyani-Singhal algorithm is an example of global state detection based algorithm. The works in this area are largely based on results by Chandy and Lamport. The key to global state is that we are going to find a consistent global state without freezing of the underlying computation. Consistent global state is a state that is possible to exist in the system. Chandy and Lamport show how to obtain a consistent global state of the computation by propagating markers along the links of the system. A maker separates the messages in the links into those to be included in the snapshot (i.e. channel state or process state) from those not to be recorded in the snapshot. It acts as delimiters for the messages in the channels so that the channel state recorded by the process at the receiving end of the channel satisfies the condition that if a message m is sent by i to j, if sending m is not in the consistent state, receive event of m in j should not in the consistent state and m should not in the channel state. After finding such a consistent global state, the algorithm
The algorithm consists of a single phase. The single phase comprises a diffusion of messages outward from the initiator process along the edges of the WFG (outward sweep), and then the echoing of diffusion messages inward to the initiator process (inward sweep). During the outward sweep of messages, the system records a snapshot which each process on receiving the message records its local state. During the inward sweep of messages, the system reduces the snapshot in a way that it can simulate the possible future of the system in terms of unblocking events.

Deadlock Prevention

Prevention is the name given to schemes that guarantee that deadlocks can never happen because of the way the system is structured. One of the four conditions for deadlock is prevented, thus preventing deadlocks. One way to do this is to make processes declare all of the resources they might eventually need, when the process is first started. Only if all the resources are available is the process allowed to continue. All of the resources are acquired together, and the process proceeds, releasing all the resources when it is finished. Thus, hold and wait cannot occur.

The major disadvantage of this scheme is that resources must be acquired because they might be used, not because they will be used. Also, the pre-allocation requirement reduces potential concurrency.
Another prevention scheme is to impose an order on the resources and require processes to request resources in increasing order. This prevents cyclic wait and thus makes deadlocks impossible.

One advantage of prevention is that process aborts are never required due to deadlocks. While most systems can deal with rollbacks, some systems may not be designed to handle them and thus must use deadlock prevention.

Deadlock Avoidance

In deadlock avoidance the system considers resource requests while the processes are running and takes action to insure that those requests do not lead to deadlock. Avoidance based on the banker's algorithm, sometimes used in centralized systems, is considered not practical for a distributed system. Two popular avoidance algorithms based on timestamps or priorities are wound-wait and wait-die. They depend on the assignment of unique global timestamps or priority to each process when it starts. Some authors refer to these as prevention.

In wound-wait, if process A requests a resource currently held by process B, their timestamps are compared. B is wounded and must restart if it has a larger timestamp (is younger) than A. Process A is allowed to wait if B has the smaller timestamp. Deadlock cycles cannot occur since processes only wait for older processes. In wait-die, if a request from process A conflicts with process B, A will wait if B has the larger timestamp (is younger). If B is the older process, A is not allowed to wait, so it dies and restarts.

In timeout based avoidance, a process is blocked when it requests a resource that is not currently available. If it has been blocked longer than a timeout period, it is aborted and restarted. Given the uncertainty of message delays in distributed systems, it is difficult to determine good timeout values.

These avoidance strategies have the disadvantage that the aborted process may not have been actually involved in a deadlock.

A reliable distributed system should include a distributed deadlock detection algorithm for efficient processing. As more interest in building a reliable distributed system grows, the necessity of systematically conducting a performance study of those algorithms will also increases. Although there has been a comprehensive study on distributed deadlock detection algorithms, very little attention has been paid on the performance measurements (Kumar and Harous 1991). The performance of distributed algorithms can be accessed through three methods, namely: analytical model, simulation model and a real system/ part of the real system.

In the analytical model, performances measures that are expressed as a set of mathematical equations are used to predict the performance of algorithms. In the literature, very few works have been carried out for evaluating the performance of distributed deadlock detection algorithms analytically. Min and Belford (1990) have proposed an analytical model to compare the performance of two probe-based algorithms, namely: Obermarck's algorithm and Chandy et al's algorithm in distributed database systems. This study has used the performance measures such as mean unblocking time of lock requests conflict, execution time of a transaction and the length of Transaction Wait-For Chain (TWFC).

Lee and Kim (2001) have proposed probabilistic based analytic model by overcoming the limitations of Min and Belford's model. Moreover, the time dependent behavior of each process and the relations among the times when deadlocked process become blocked are considered for improving the accuracy of the model. It compares a modified version of Lee and Kim's algorithm (1995) and Chandy et al's algorithm in terms of Deadlock Duration (DD), the number of algorithm initiations and the mean waiting time of a process. However, the analytical models can not represent all the

characteristics of a real distributed system. Some features of distributed systems have to be dropped to keep this model simple and easy to implement.

The second method relies on simulation which can represent more complex behaviors than an analytical model. Since simulation model has few assumptions as compared to the analytical model, it gives more flexibility to represent the real system. The simulation is run over a fixed period of time and the performance measures are estimated from model-generated data with statistical procedures. In contrast to analytical model, simulation is widely used to estimate the performance measures of distributed deadlock detection algorithms. Choudary et al (1990) have demonstrated that their algorithm, named LIST, outperforms Sinha and Natarajan's algorithm in heavily loaded systems. Bukhres (1992) has compared the performance of centralized and distributed algorithms. And, the result reveals that the centralized algorithms performs better than distributed algorithms in lightly loaded systems, and distributed algorithms outperforms centralized algorithms in heavily loaded systems. Lee (2004) has used event driven simulation to compare the performance of generalized deadlock detection algorithms in terms of mean algorithm execution time, mean deadlock latency and mean number of deadlock detection messages. Razzaque et al (2007) have used simulation to compare the performance of their algorithm with that Kim et al's algorithm (1997) and Lee and Kim's algorithm (1995).
The third method measures the performance of the algorithms by running the algorithms in real distributed systems. Although it provides accurate experimental measurements, it needs more cost, time and resources. Hence, it is not used for evaluating distributed deadlock detection algorithms. Since the experimental measurements are difficult in real distributed environment, most of the previous works have been done through simulation. Perhaps, the simulation provides an efficient way to measure the performance of the algorithms before the real system is implemented (Wiesmann and Schiper 2005). Therefore, the simulation technique is chosen for assessing the performance of generalized deadlock detection algorithms in this paper. The event driven simulator is used to evaluate the performance of generalized deadlock detection algorithms. The simulator measures the key performance metrics such as deadlock detection, message complexity and message length under different multiprogramming levels.

CONCLUSIONS

A Deadlock is a canonical, yet an overlooked problem in distributed systems. DD&R is the cheapest method to handle distributed deadlocks. However, deadlock detection is cumbersome in distributed systems since no site has complete knowledge about the resource requirements of all processes. The distributed deadlock detection algorithms are classified based on the underlying deadlock models such as AND model, OR model, AND-OR model and P out-of Q model. Among the models, P out-of Q model, also called as generalized request model, has the modeling power of all other models and much more concise expressive power than other models. However, it is difficult to detect deadlock in the generalized request model. It is observed that only very few algorithms have been proposed to detect deadlock in the literature. Though the algorithms have reduced the deadlock duration through the years, they have paid very little attention on other key measures such as number of messages and message size. This paper concludes by addressing various issues associated with one of the fundamental resource management problems, the deadlock.