Equatorial Journal of Communication Technology Vol. 3 (02), pp 20 – 24 November, 2020
ISSN: 2231 - 3364

DISTRIBUTED DEADLOCK AND DISTRIBUTED RECOVERY

Dickson, Makasda Solomon

Department of Information and Communications Technology, National Institute for Nigeria Languages, Aba, Abia State, Nigeria.
dicksonsolomon1988@gmail.com

Keywords:
Distributed databases, deadlock detection, deadlock recovery, transaction queue, distributed transaction structure, transaction manager.
Available online: 6th November, 2020

ABSTRACT

Distributed system consists of two kinds of components: sites, which process information, and communication links, which transmit information from site to site, this paper therefore aimed at examining the impact of distributed deadlock and distributed recovery in system performance and data transition. Through a critical review of extant literatures, it was established that: as the need of distributed processing increases, the complexity in handling of deadlocks also increases. In distributed databases, the conditions for the deadlocks are same as that in centralized but harder to detect, avoid and prevent. This paper assumes that special procedures are required to resolve the deadlock through engaging algorithm to reduce the number of transactions that are to be aborted to resolve the deadlocks, and improving the performance of the system.

Introduction

A distributed system consists of a collection sites that are interconnected through a communication network. Each site has the local database and transactions running on them. Although the sites are dispersed, a distributed database system manages and controls the entire database as a single collection of data. A deadlock can occur because transactions wait for one another. Informally, a deadlock situation is a set of requests that can never be granted by the concurrency control mechanism.

In computer science, deadlock refers to a specific condition when two or more processes are each waiting for another to release a resource, or more than two processes are waiting for resources in a circular chain. One issue that required special attention when using either primary copy or fully distributed locking is deadlocking detection. Each site maintains a local waits for graph and a cycle in a local graph indicates a deadlock.

There are two categories of distributed deadlock detection algorithms: Probe based detection algorithms and edge chasing algorithms. Many authors proposed various algorithms under these categories. Chandy (2007) for instance, proposed an algorithm that uses Transaction Wait For Graphs (TWFG) and probes to detect the local and global deadlocks respectively. It uses colored graphs for detecting the deadlocks and has the disadvantage of large space complexity and no deadlock resolution mechanism in order to make the system deadlock free. Sinha (2005) on the other hand, observed that an algorithm that was based on priorities of transactions to reduce the number of messages is required for deadlock detection. In this scheme, a transaction's request for a lock on a data item is sent to the data manager for the item. When a transaction begins to wait for a lock, all the probes from its queue is propagated. When a data manager gets back the probe it initiated, deadlock is detected. Since the probe contains the priority of the youngest transaction in the cycle, the youngest transaction is aborted. In this algorithm data managers do not store probes and transactions are used as nodes of the graph. Due to this, another level of non-atomicity is added and complicated rules are required to add the new probes and delete the previous ones, whenever the WFG changes.

According to Obermack (2010), algorithm, builds and analyzes directed TWFG and uses a distinguished node at each site. The detection algorithm builds a TWFG and adds on all the information received from others processes also. Then it creates wait-for edges from external to each node representing agent of transaction that is expected to send on communication link and that is waiting to receive from communication link. Then it analyzes the TWFG and breaks down the youngest transaction creating the cycle. The algorithm does not work correctly because the WFG constructed at any instant does not represent a snapshot of the global WFG resulting into the detection of false deadlocks. This paper therefore theoretically examined the level of distributed deadlock and distributed recovery.

Condition Necessary for Deadlock

The following are conditions that are necessary for deadlock:

Mutual exclusion: A resource may be acquired exclusively by only one process at a time. This is expressed in figure 1 below:

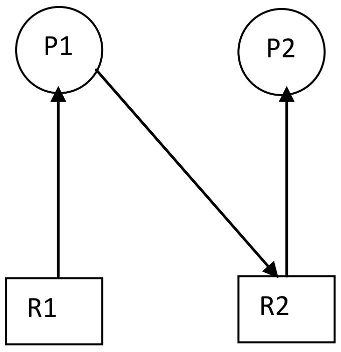

Figure 1: Mutual exclusion condition (Source: Dhaval, 2017).

Hold and Wait: Processes currently holding resources that's were granted earlier can request new resources.

Deadlock Detection: Deadlock detection is the process of actually determining that a deadlock exists and identifying the processes and resources involved in the deadlock. Detection of a cycle in WFG proceeds concurrently with normal operation in main of deadlock detection. To detect such deadlocks, a distributed deadlock detection algorithm must be used and we have three types of algorithm:

1. Centralized Algorithm.
2. Simple Algorithm.
3. Hierarchical Algorith.

Centralized Algorithm: it consist of periodically sending all local waits for graphs to some one site that is responsible for global deadlock detection. At this site, the global waits for graphs is generated by combining all local graphs and in the graph the set of nodes is the union of nodes in the local graphs and there is an edge from one node to another if there is such an edge in any of the local graphs.

Simple Algorithm: if a transaction waits longer than some chosen time out interval, it is aborted. Although this algorithm causes many unnecessary restart but the overhead of the deadlock detection is low.

Hierarchical Algorith: this algorithm groups the sites into hierarchies and the sites might be grouped by states, then by country and finally into single group that contain all sits.Every node in this hierarchy constructs a waits for graph that reveals deadlocks involving only sites contained in (the sub tree roots at) this node.

Deadlock Prevention: The deadlock prevention approach does not allow any transaction to acquired locks that will lead to deadlocks. The conventions request for locking the same data items, only one of them is granted by the lock. One of the most popular deadlock prevention is methods is pre-acquisition of all the locks.

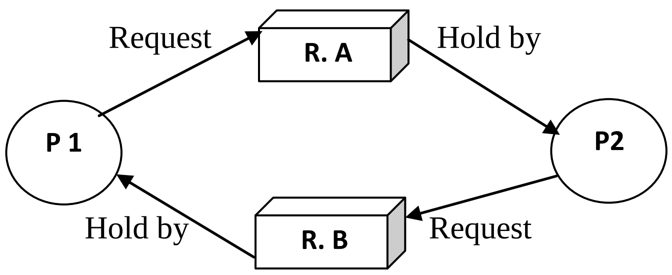

Figure 2: hold and wait condition (Source: Dhaval, 2017).

Deadlock Avoidance: The deadlock avoidance approach handles deadlocks before they occur. It analyzes the transactions and the locks to determine whether or not waiting leads to a deadlock.There are two algorithms for this purpose, namely wait-die and wound-wait.

Let us assume that there are two transactions, T1 and T2, where T1 tries to lock a data item which is already locked by T2. The algorithms are as follows:

Wait Die: if T1 is older than T2, T1 is allowed to wai. Otherwise, if T1 is younger than T2, T1 is aborted and later restarted.

Wound Wait: if T1 is older than T2, T2 is aborted and later restarted. Otherwise. If T1 is younger than T2, T1 is allowed to wait.

Distributed Recovery

Recovery: refers to restoring a system to its normal operation state. Once a failure has occurred, it is essential that the proces where the failure happened can recover to correct state.

Following are some solution on process recovery:

1. Reclaim resources allocated to process.
2. Undo modification made to database.
3. Restart the process.
4. Or Restart the process from point of failure and resume execution.

Recovery in distributed DBMS is more complicated than in a centralized DBMS for the following reasons:

New kinds of failure can arise: failure of communication links and failure of remote site at which a sub transaction is executing.Either all sub transactions of a given transaction must commit or none must commit and this property must be guraanteed desipite and combination of site and link failures. This guaranteed is achieved using a commit protocol.

Normal execution and Commit Protocols:

During normal execution each site maintains a log and the actions of a sub transaction are logged at the site where it executes. The regular logging activity is carried out which means a commit protocol is followed to ensure that all sub transactions of a given transction either commit or abort uniformaly.

The transaction manager at the site where the transaction originated is called the coordinator for the transaction and the transaction managers where its sub transactrions execute are called Subordinates.

Two Phase Commit Protocol: When the user decides to commit the transaction and the commit command is sent to the coordinator for the transaction.

This initiates the 2PC protocol: The coordinator sends a prepare message to each subordinate. When a subordinate receive a Prepare message, it then decides whether to abort or commit its sub transaction. It force writes an abort or prepares a log record and then send a No or Yes message to the coordinator

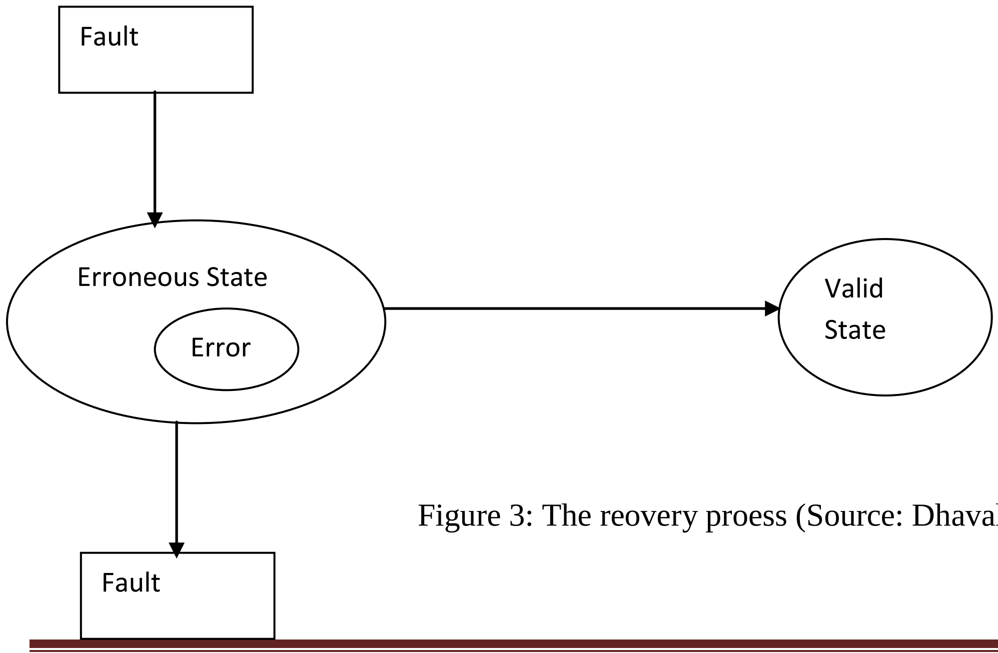

Figure 3: The reovery proess (Source: Dhaval, 2017).

Concept of recovery

System failure: Syetem does not meet requirements

Erroneous system state: State which could lead to a system failure by sequence of valid state transitions.

Error: the part of the system state which different from its intended value.

Fault: Anormal physical condition. Eg, design errors, manufacturing problems, damage, external disturbances.

Classification of Failure

In a distributed database system, we need to deal with four types of failure:

1. Transation failures (aborts)
2. Site (system) failures
3. Media (disk) failures
4. Communication line.

Some of these are due to hardware and others are due to software. Software failures are typically caused by “bugs” in the code. Most of software failures are soft failures.

Transaction Failures: can fail for a number of reasons. Failure can be due to an error in the transaction caused by incorrect input data as well as the detection of a present or potential deadlock. Some concurrency control algorithms do not permit a transaction to proceed or even to wait if the data that they attempt to access are currently being accessed by another transaction. This might also be considered a failure.

Site (system) failures: A system failure is assumed to result in the loss of main memory contents. Therefore, any part of the database that was in main memory buffers is lost as a result of a system failure. In distributed database terminology, system failures are typically referred to as site failures, since they result in the failed site being unreachable from other sites in the distributed system.

Total failure: refers to the simultaneous failure of all sites in the distributed system.

Partial failure: indicates the failure of only some sites while the others remain operational.

Media Failures: media failure refers to the failures of the secondary storage devices thst store the database. Such failures may be due to operating system errors, as well as to harfware faults such as head crashes or controller failures. The important point from the perspective of DBMS reliability is that all or part of the database that is on the secondary storage is considered to be destroyed and inaccessible.

Communication Failures: are unique to the distributed case. There are a number of types of communication failures. The most common ones are the errors in messages, improperly, and communication line failures. Lost or undeliverable messages are typically the consequence of communication line failures or (destination) site failures. The detection will be facilitated by the use of timers and a timeout mechanism that keeps track of how long it has been since the sender site has not received a confirmation from the destination site about the receipt of a message.

Methods to control failure

Failure affected transactions must be aborted.Site failure message is broadcasted to all sites. Checking must be done periodically to see whether the failed site has recovered or not.After restrating the failure site, site must initiate a recovery procedure to abort all partial transaction that were active at the time of failure.

Conclusion

Deadlock detection is the most important problem that must have a strong attention in case of distributed systems. Several algorithms have been proposed for detection and resolution of deadlocks. Also have analyzed the performance of algorithm and compared with techniques presented in the literature. I observed that the technique to resolve deadlock is by terminating less number of transactions. 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. A deadlock is a state where a set of processes request resources that are held by other processes in the set.

Deadlock resolution involves breaking existing wait-for dependencies between the processes to resolve the deadlock. It involves rolling back one or more deadlocked processes and assigning their resources to blocked processes so that they can resume execution.

References:

Efrem, G. Mailach (2009). *Advance database techniques*. New York: McGraw Hill. Pp. 23-29.

Peter, R. and Coronel, T. (2005). *Database systems, Design implementation and managmnet.* New Delhi: Thomson Learning. Pp. 32 – 34.

Dhaval, R. C. (2017). *Fundamentals of Database Systems*. New York: Person Education. Pp 4-7.

Chandy, X. M. and Misra, J. (2007). A Distributed Algorithm for Detecting Resource Deadlocks in Distributed Systems. Proceeding of the First ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, New York, pp. 157-164, 1982.

Obermack, R. (2005). Distributed deadlock Detection Algorithm. *ACM Transaction on Database Systems, 7 (2): 144 -156.*

Sinha, M. K. and Natarjan, N. (2010). A Priority Based Distributed Deadlock Detection Algorithm. *IEEE Transaction on Software Engineering, 11 (1): 67-80.*