Computers and Mathematics with Applications 63 (2012) 458–468

Contents lists available at SciVerse ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

An efficient deadlock prevention approach for service oriented transaction processing

Feilong Tang [a,*], Ilsun You [b], Shui Yu [c], Cho-Li Wang [d], Minyi Guo [a], Wenlong Liu [e]

[a] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
[b] School of Information Science, Korean Bible University, Nowon-gu, Seoul, South Korea
[c] School of Information Technology, Deakin University, Burwood, VIC 3125, Australia
[d] Department of Computer Science, The University of Hong Kong, Hong Kong
[e] School of Information and Communication Engineering, Dalian University of Technology, Dalian, China

ARTICLE INFO

Keywords:
Distributed transaction processing
Deadlock prevention
Service oriented architecture (SOA)
Replication
Two-phase commit (2PC)

ABSTRACT

Transaction processing can guarantee the reliability of business applications. Locking resources is widely used in distributed transaction management (e.g., two phase commit, 2PC) to keep the system consistent. The locking mechanism, however, potentially results in various deadlocks. In service oriented architecture (SOA), the deadlock problem becomes even worse because multiple (sub)transactions try to lock shared resources in the unexpectable way due to the more randomicity of transaction requests, which has not been solved by existing research results. In this paper, we investigate how to prevent *local deadlocks*, caused by the resource competition among multiple sub-transactions of a global transaction, and *global deadlocks* from the competition among different global transactions. We propose a replication based approach to avoid the local deadlocks, and a timestamp based approach to significantly mitigate the global deadlocks. A general algorithm is designed for both local and global deadlock prevention. The experimental results demonstrate the effectiveness and efficiency of our deadlock prevention approach. Further, it is also proved that our approach provides higher system performance than traditional resource allocation schemes.

1. Introduction

Business applications have necessitated distributed transaction management technologies. Existing distributed transaction models widely use the resource locking mechanism for keeping the consistency of transaction systems, among which two-phase commit (2PC) [1] is the most representative coordination protocol through requiring sub-transactions to lock resources before the transaction commit. Unfortunately, 2PC-like protocols potentially induce various deadlocks when multiple (sub-)transactions try to lock the same resource at the same time.

Service oriented architecture (SOA) presents new requirements and challenges to the transaction management. With the success of SOA, many large-scale information systems have been set up to provide business services simultaneously [2,3]. In SOA environments, the deadlock problem due to the resource competition among multiple (sub)transactions gets worse because of the randomicity of transaction requests and the uncontrollability of transaction execution order [4,5]. The deadlock in SOA environments will occur more often than that in traditional distributed systems. As a result, new deadlock prevention [6] approaches are needed for improving the performance of service-oriented systems.

* Corresponding author.
E-mail address: tang-fl@cs.sjtu.edu.cn (F. Tang).

0898-1221/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2011.07.063

Deadlock control technologies can be categorized as *deadlock avoidance* [7–9], *deadlock detection* [10,11] and *deadlock prevention* [12–14]. The *deadlock avoidance* strategy only accepts the requests that will lead to safe states. Although it allows more concurrency [15,16], it has to know the number and type of all resources before the actual resource allocation. For many systems, however, it is impossible to know all resources and their states in advance. The *deadlock detection* [17] tracks resource allocation and process states, and restarts one or more processes to remove deadlocks. To detect deadlocks introduced by concurrent resource accesses, some researchers proposed wait-for-graph-based detection algorithm [10]. Timeout based probabilistic analysis [18] is also used to detect global deadlocks in distributed systems, however, the timeout itself is different to be decided [19]. On the other hand, after a deadlock is detected, one of transactions in the wait cycle must be aborted. Wang et al. [20] proposed guaranteed deadlock recovery based on run-time dependency graph and incorporated it into distributed deadlock detection algorithm. Although the deadlock detection is effective, it costs more time. Moreover, with regard to local deadlocks defined in this paper, nothing can be done even if a deadlock is detected. Finally, the *deadlock prevention* mechanism often removes the "hold and wait" condition by requiring processes to request all the needed resources before starting up. Ezpeleta. et al. [12] proposed a Petri net based deadlock prevention policy based on the liveness or the reachability of Petri nets. An advantage of the deadlock prevention is that it does not need to know details of all resources available and requested. So, the deadlock prevention approach is more suitable for dynamical and open service-oriented environments.

In this paper, we propose an algorithm to prevent *local deadlocks*, caused by the resource competition among multiple sub-transactions of a transaction, and *global deadlocks* due to the resource competition among different global transactions. In our scheme, we control concurrent resource accesses through the resource manager, which is particularly useful for business transactions in service-oriented environments. We describe in brief our contributions in this paper as follows.

(1) We propose a replication based approach to avoid local deadlocks. In traditional deadlock prevention schemes, when two or more sub-transactions of a global transaction compete for the same resource, the global transaction will have to be aborted. On the other hand, whenever the global transaction restarts, it will inevitably fail again due to the same resource competition.
(2) We propose a timestamp based approach to prevent global deadlocks. In our scheme, the conflicted transactions that compete for the same resource are selectively aborted after a timeout, based on their transaction ID. Consequently, our approach avoids the live-locks due to resource competition among global transactions.
(3) A general algorithm is designed for preventing both local and global deadlocks, based on the solutions proposed above.
(4) We design an intelligent resource manager by merging the deadlock prevention function with the resource management function. The resource manger can detect and prevent local and global deadlocks and allocate appropriate lock(s) for each transaction.

The experimental results demonstrate that our replication based mechanism completely avoids the local deadlocks. On the other hand, our timestamp based mechanism significantly reduces the global deadlocks and live-locks.

The remainder of this paper is organized as follows. In Section 2, we briefly introduce the related background and formally define the local and global deadlocks. Section 3 presents our replication based approach for avoiding the local deadlocks, the timestamp based mechanism to prevent global deadlocks, and the general algorithm for local and global deadlock prevention. The implementation and performance evaluations are reported in Section 4. Finally, we conclude this paper in Section 5.

2. Preliminaries

The deadlock in the transaction processing is highly relevant to transaction commit protocols. Our deadlock prevention is designed for 2PC-like protocols. In this section, we briefly describe the 2PC protocol and its locking implementations, and then formally define local and global deadlocks.

2.1. Background

2PC was designed for coordinating distributed atomic transactions with the following properties: atomicity, consistency, isolation and durability. Usually, a distributed atomic transaction is managed by a transaction manager (TM) together with a set of resource managers (RM) responsible for allocating individual resources for corresponding sub-transactions. A TM controls multiple RMs involved in a global transaction. On the other hand, an RM also can be shared by multiple TMs for concurrent transactions.

A distributed transaction contains a set of sub-transactions executed in different networked nodes [21,22]. Each of them works as a participant under the control of the TM. The 2PC protocol guarantees that a transaction is either successfully committed or not performed at all. In 2PC-based transaction processing, the TM coordinates all the sub-transactions in the following two phases.

Phase 1. All participants (sub-transactions) receive instructions from a transaction coordinator (i.e., TM) to prepare for commit. In most cases, it is achieved through locking needed resources. If a resource manager can lock the needed resources for a corresponding sub-transaction, it votes OK; that means it is ready to commit. Otherwise, it responds Failed to the TM.