Two-Phase Commit Processing with Restructured Commit Tree

George Samaras[1]*, George K. Kyrou[1,2]**, and Panos K. Chrysanthis[2]**

[1] Department of Computer Science, University of Cyprus
Nicosia, Cyprus
{cssamara,kyrou}@cs.ucy.ac.cy

[2] Department of Computer Science, University of Pittsburgh
Pittsburgh, PA 15260, USA
panos@cs.pitt.edu

Abstract. Extensive research has been carried out in search for an efficient atomic commit protocol and many optimizations have been suggested to improve the basic two-phase commit protocol, either for the normal or failure case. Of these optimizations, the read-only optimization is the best known and most widely used, whereas the flattening-of-the-commit-tree optimization is the most recent one proposed for Internet transactions. In this paper, we study in depth the combined use of these two optimizations and show the limitations of the flattening-of-the-commit-tree method in committing large trees. Further, we propose a new restructuring method of the commit tree and show using simulation that it performs better than flattening method even when dealing with large trees.

Keywords: Atomic Commit Protocols, Commit Optimizations, Distributed Transaction Processing

1 Introduction

A transaction provides reliability guarantees for accessing shared data and is traditionally defined so as to provide the properties of *atomicity, consistency, isolation, and durability* (ACID) [8]. In order to ensure the atomicity of distributed transactions that access data stored at multiple sites, an *atomic commit protocol* (ACP) needs to be followed by all sites participating in a transaction's execution to agree on the transaction's final outcome despite of program, site and communication failures. That is, an ACP ensures that a distributed transaction is either *committed* and all its effects become persistent across all sites, or *aborted* and all its effects are obliterated as if the transaction had never executed at any

* This work was supported in part by the European IST project DBGlobe, IST-2001-32645.

** This work was supported in part by the National Science Foundation under grant IIS-9812532.

site. The *two-phase commit protocol* (2PC) is the first proposed and simplest ACP [7, 14].

It has been found that commit processing consumes a substantial amount of a transaction's execution time [22]. This is attributed to the following three factors:

Message Complexity, that is the number of messages that are needed to be exchanged among the sites participating in the execution of a transaction in order to reach a consistent decision regarding the final status of the transaction. This captures the cost of delays due to network traffic and congestion.

Log Complexity, that is the amount of information that needs to be recorded at each participant site in order to achieve resilience to failures. Log complexity considers the required number of *non-forced* log records, which are written into the log buffer in main memory, and the *forced* log records, which are written into the log on a disk (stable storage) that sustains system failures. Typically, however, log complexity is expressed only in terms of forced log writes because during forced log writes, the commit processing is suspended until the log record is guaranteed to be on stable storage. Thus, this captures the cost due to I/O delays.

Time Complexity, that is the number of *rounds* or *sequential exchanges* of messages that are required in order for a decision to reach the participants. This captures the cost due to propagation and network latencies.

An increase to any one of these factors, it will increase the delay in making the final decision to commit a transaction and consequently will increase the transaction response time. Further, any delay in making and propagating the final decision to commit a transaction, it delays the release of resources which in turn decreases the level of concurrency and adversely affects the overall performance of a distributed transaction processing system. For these reasons, extensive research has been carried out over the years to reduce message, log and time complexities in an effort to optimize 2PC, improving both response time and system throughput during normal processing or during recovery from failures.

These research efforts resulted in a number of 2PC variants and new atomic commit protocols [16, 13, 24, 21, 10, 19, 15, 5, 1, 4] and an even greater number of commit optimizations for different environments [20, 6]. Of these optimizations, the *read-only* optimization is the best known and most widely used, whereas the *flattening-of-the-commit-tree* optimization is the most recent one originally proposed in [20] for *Internet transactions*, distributed across sites in a wide area network. Interestingly, the performance effects on transaction processing when combining some of these optimizations have not been studied in depth.

In this paper, we concentrate on the combined use of read-only and flattening-of-the-commit-tree optimizations. We show using simulation how the flattening-of-the-commit-tree optimization improves distributed commit processing by minimizing propagation latencies and allowing forced log writes to be performed in parallel. A major shortfall of the flattening method when dealing with large