Market Requirements Document

Feature Name: Lock Server Performance Improvement

Version: 1 **Date Submitted:** 11/08/2007 **Completed By:** Leon Guzenda and Aziz Ahmad

Description of the Problem

Background

Lock server processes support four major functions:

- They implement a derivative of hierarchical Gray-Code locking to serialize access to groups of user and system data. This is done by granting, queuing or denying locks to client processes.
- They enforce deadlock detection within the group of databases that they control. Only one lock server controls a given database at a particular time.
- They can perform automatic recovery, i.e. the rolling back of transactions that have terminated incorrectly, such as when a client processor halts.
- Each lock server supports various kinds of instrumentation:
  - There is a command line tool called oolockmon that prints statistics gathered by an individual lock server.
  - There is a Lock Server Performance Monitoring interface that allows a listener to see the flow of events within a lock server.

It is possible to link an In-Process Lock Server to an individual client process. It services lock requests from that process, its threads and other clients.

The implementation of 'poll' and 'poll7' improved the connection time and increased the limit on the number of simultaneous transactions in Unix environments. However, the use of 'select' in Windows environments continues to be a limiting factor in increasing the number of simultaneous transactions.

The current lock and transaction table design relies on processing one request at a time, inhibiting recent performance improvements in the Unix environment.

Problem

In highly active systems a lock server can become a performance bottleneck. There are many reasons, the most significant of which (currently) are:

a) The clients sometimes wait for a long time and occasionally lose connections while waiting to get locks from the lock server.
b) On Microsoft Windows the lock server is limited to 1031 simultaneous transactions.
c) The lock server is single threaded and cannot take advantage of multiple CPUs on a single host.

Description of the Requested Feature

The purpose of this project to is to:

- Reduce the time taken to connect to a lock server.
- Reduce the time to acquire a lock from the lock server
- Increase the simultaneous transactions limit to 10,000.

Part of an existing feature or does it require another feature, if so, which one?

- This feature is a part of the standard product.

How is this problem being solved now, and why isn't that acceptable?

Application developers can either:

- Adopt AMS and partitions and use multiple lock servers ,
- Or, reduce the frequency or quantity of lock requests by redesigning the physical layour of a federation.

In some cases introducing AMS adds unacceptable overheads to intensive I/O applications. Either way, we are forcing the application developer to work around unreasonable limitations of our current implementation of ACID and MROW transactions.

What languages must support this capability?

- All supported languages.

Which platforms must be supported?

- All platforms.

Do any competitors already have this feature?

- Versant supports object level locking. However, its scalability is doubtful.

Customers who require this feature

- Most VLDB customers, including Northrop Grumman Corporation (NGC, ex-TRW).

Revenue at risk, or which could be won

- We are not aware of any business lost as a result of this problem, but it was an impediment in a recent attempt to partner with Violin Technologies at NGC.

When is this required?

- Release 10.

Additional Notes

1. We will also need:

- Updated Technical Publications (in the event that table sizes or other limitations change).
- New QA material to prove that the changes have indeed improved the performance and scalability of the lock server without impacting reliability or locking behavior.

2. It may be possible to have multiple threads servicing lock requests for subsets of the group of databases being controlled. However, this should not be done at the expense of deadlock detection.