An Introduction to Replication Server 12.5

By Irfan Khan

A look at the latest in Sybase replication functionality

Sybase released the latest version of Sybase Replication Server 12.5 on May 15, 2002. The goal of this release has been to provide new features and enhancements for users who are working on eCommerce projects such as portals, corporate websites, Internet-based procurement, and others. It can be safely assumed that the volume of transactions will continue to increase significantly in this area now and into the future. Replication Server 12.5 is focused on:

- Greater availability of data to eliminate disruptions in delivering online services and transfer of data, including geographic distribution of data for disaster recovery.
- Secure, guaranteed replication of data.
- Bi-directional replication of data between heterogeneous databases.
- Complete support for ASE 12.5 features.

Performance Enhancements

Transaction Partitioning for Parallel DSI

A *transaction* is normally defined as a group of one or more DML commands committed together at a dataserver under a single commit command. Within Replication Server, one or more transactions from any primary dataserver can be further grouped together under a single commit for delivery to a replicate dataserver to improve performance. These groups of one or more primary transactions are what the DSI uses to schedule and distribute work to executor threads, and are used to track Begin and Commit sequencing. *Transaction partitioning* impacts the logic used to group primary transactions, and the logic to schedule and distributes these groups.

However, Replication Server performance can be victimized by resource contention at the Replicate database when attempting to process transactions concurrently using the Parallel DSI feature. This contention is particularly detrimental to performance when it results in rollback processing. Rep Server's new transaction partitioning enhancement will allow users some control in identifying transactions that should process serially in order to help avoid this contention. Transaction partitioning will allow for partitioning by User ID, Transaction Name, or Origin Begin/Commit times, or a combination of these items.

To support the partitioning feature, the following new configuration parameters have been introduced:

- dsi_partitioning_rule: This configuration parameter establishes one or more rules at the connection level which the DSI may use to partition transactions amongst available parallel DSI threads.
  Parameters: **none**|**rule** [,**rule**]
  where rule is one of: **time**|**user**|**name**.
- dsi_ignore_underscore_names: This configuration parameter will specify, at the connection level, whether or not the transaction partitioning rule "name" will ignore transaction names that begin with an underscore. ASE, by default, will assign a transaction name, even if none was explicitly provided. For example, a single **insert** command will be assigned a transaction name of "_ins".

To avoid partitioning of transactions using these ASE-generated names, this parameter, when set to "on," instructs the transaction partitioning logic to ignore (treat as NULL) any transaction name that begins with an underscore. When set to "off," transaction names with under-

Irfan Khan is a senior technical architect for Sybase's enterprise solutions division, with a focus on Replication Server. He can be reached at irfan.khan@sybase.com.

scores are treated the same as any other transaction name. This parameter only affects transaction partitioning, and only for the rule setting of "name". The default is "ON". Parameters: **on**|**off**.

- dsi_num_threads: This existing configuration parameter controls the number of parallel DSI threads to be used for each replicate database. The pre-12.5 maximum of 20 has now been increased to 40.

This new transaction partitioning feature enhances the degree of parallelism in Replication Server's interaction with the replicate database. This allows the Replication Server to apply queued-up transactions to the replicate database with a greater degree of parallelism, improving replication throughput. It attempts to mimic the behavior of the client applications at the primary database in applying the replicated transactions in parallel. At the replicate database, it partitions the transactions being applied at the replicate database into parallel groups, based on characteristics such as transaction name, transaction time, or user ID of the transaction, and applies the groups in parallel.

The partitioning characteristics, as well as the degree of parallelism managed by Replication Server, are all configurable, giving administrators control over their environments and providing them with tools to tune the system to their requirements. The 12.1 feature "monitors and counters" has been extended to provide metrics on the effectiveness of the partition strategy assigned to the DSI connection.

Support for SMP

While earlier releases of Replication Server could run on SMP systems but only use a single CPU, the enhancements in this release allow the server to take advantage of multiple CPUs on the system. This feature greatly enhances the scalability of Replication Server and its ability to support numerous sources and targets in a single environment. Even within simple replication environments with limited number of sources and targets, this feature enhances the throughput of the replication system by parallelizing the numerous internal tasks and will help cut down on replication latency.

The Problem of Commit Sequencing for Parallel DSI

Pre-12.5 Replication Server offers a technique for applying transactions at a replicate database in parallel. Actually, the transaction updates may be performed in parallel, but *commit processing* is serialized in order to maintain commit consistency with the primary database.

Parallel DSI processing must be able to resolve the transaction contention that results in a *commit consistency deadlock*. In this case, a transaction cannot commit because it must wait for another transaction to commit first (must preserve commit consistency), but the other transaction cannot complete because needed resources are locked by the first transaction. For example, assume DSI threads (database connections) 1 and 2 are each processing a transaction, and Thread 1's transaction must commit before Thread 2's. Further assume that both threads are executing in parallel. If Thread 2 issues a transaction that blocks Thread 1, Thread 1 will wait. When Thread 2 is done, it must wait for Thread 1 to commit first. We now have a commit consistency deadlock.

How should this situation be handled? Part of the difficulty is that the deadlock is spread across two domains. Transaction contention is defined by the activity in the replicate database. The commit consistency contention is defined by the DSI. Neither domain, replicate or DSI, knows about the other. The pre-12.5 solution was to externalize the DSI's contention into the replicate database using the *rs_threads* table. This allows the replicate to "see" the deadlock and take action.

Each thread has its own row in the rs_threads table. At the start of each transaction, a thread will issue an update to its row in rs_threads. Before any thread commits, it verifies that the preceding thread's row is selectable (no longer being updated). In our example, Thread 2 has a pending select on the rs_threads row for which Thread 1 has issued an update. The replicate database is assumed to be able to detect this deadlock and can select one of the transactions for rollback. The DSI recognizes that the replicate forced a rollback. The DSI can then resubmit the transactions serially to avoid the contention the second time.

Aside from the need to manage the rs_threads table, RepServer has no control over which transaction is selected for rollback. This means that all currently active transactions must be rolled back and re-started. In particular, Transaction 1 is rolled back even though we only really require that Transaction 2 be rolled back, so that Transaction 1 can continue to completion.

An Alternative to rs_threads for Ensuring Commit Consistency

As we can see, with parallel DSI in use, each transaction at the replicate database requires more (potentially network) I/O than the same transaction required at the primary. Elimination of this I/O, so that replicated transactions will more closely resemble their counterparts at the primary database, is highly desirable.

Sybase has solved this problem by taking a different

approach than that of the rs_threads table. In RepServer 12.5, the rs_threads table has not been removed, in order to maintain backward compatibility. However, it is now an *option* for the commit control functionality. It is anticipated, however, that in all cases of parallel DSI implementation, regardless of transaction partitioning technique, commit control handled within Replication Server will yield better performance than that afforded by rs_threads. Without rs_threads to assist in commit control, two issues immediately become apparent:

1) There needs to be some means of identifying transaction commit order, and,
2) Commit consistency deadlock needs to be handled expeditiously.

Commit consistency can be preserved by creating a list of the OQIDs (Origin Queue IDs) of committed transactions being processed and maintained in a list in commit order. Threads ready to commit will only be allowed to do so if theirs is the first commit OQID on the list, in which case the transaction will commit and remove itself from the list. If theirs is not the first on the list, they must wait.

Recognizing that it may be the case that the transaction waiting to be committed may be locking database resources required by transactions needing to commit sooner, a thread will also need to determine if indeed its transaction is blocking other transactions. If true, then the thread must roll back its transaction (freeing any held resources that are blocking other transactions from continuing) and start processing the transaction again. For as long as it is not true, the thread is free to wait, trying to commit and checking itself for blocking others.

The list of OQIDs is maintained in the correct commit order, and therefore there will be no real problem with long and short transactions running concurrently on different threads. The list is only checked when a transaction is ready to commit, and the first OQID is removed when the commit is successful.

This *transaction commit order processing* resolves the problem of commit consistency deadlock. Once Transaction B is ready to commit, it must check the commit OQID list to see if its commit OQID is first. If it is not, then Thread B must check to see whether it is blocking a thread needing to commit earlier and roll back accordingly.

To support the commit-sequencing feature, the following new configuration parameters have been introduced:

- dsi_commit_control: This configuration will specify at the connection level whether Commit Control processing is internalized "on" (new internal non-rs_threads model) or externalized "off" (uses the existing rs_threads tables and associated logic). By allowing a choice, users may select the method that provides the greatest benefit to their unique environment. Parameters: on | off. Default: on.
- dsi_commit_check_locks_max: Maximum number of *rs_dsi_check_thread_lock* executions before rollback and retry. This function string calls the maximum number a DSI Executor thread may execute before pessimistically terminating itself, under the assumption that the thread cannot commit because it is in conflict with prior transactions. This parameter allows the user to provide some limit, in case the thread checks indefinitely. This parameter is of most benefit to non-ASE Replicates, where a function string to check for locks cannot be devised, to force a long-waiting thread to roll back and retry. This value will be configurable at the connection level according to the following parameters:
  - Units: Integer
  - Max: 1,000,000
  - Min: 1
  - Default: 400 (approximately 33 minutes if the *dsi_commit_check_locks_intrvl* default of 5000 milliseconds is used)
- dsi_commit_check_locks_log: Number of *rs_dsi_check_thread_lock* executions to wait before logging a message. When the return from the *rs_dsi_check_thread_lock* function string is zero (no locks are being held on resources that are needed by other processes), and the maximum executions of *dsi_commit_check_locks_max* has not been reached, the DSI Executor thread is allowed to loop trying to commit its transaction. Administrators must be able to set a maximum number of function string executions to occur before logging a warning message to the administrator. This configuration will specify at the connection level the maximum number of *rs_dsi_check_thread_lock* function string calls a DSI Executor thread may execute. When the warning message is logged, the counter will be reset and the wait period will start again.
  - Units: Integer
  - Max: 1,000,000
  - Min: 1
  - Default: 200 (approximately 16 minutes if *dsi_commit_check_locks_intrvl* default of 5000 milliseconds is used)
- dsi_commit_check_locks_intrvl: Number of milliseconds to wait before querying locks. This configuration parameter will allow the user to set the number of milliseconds a process must allow to elapse before executing the function string *rs_dsi_check_thread_lock*. It allows the administrator to increase or decrease the time a thread will wait before

expending the replicate database time answering the "am_I_blocking" question. After executing the function, if the return is 0 (thread is not blocking), the timer will be reset and the wait period will start again.

- Units: Milliseconds
- Max: 86,400,000 (24 hours)
- Min: 0 (check continuously)
- Default: 5000 (5 seconds)

Pre-12.5 Replication Server attempts to apply transactions at the replicate site in parallel (Parallel DSI). However, in order to preserve the integrity of the database, the transaction commits are sequenced with the help of the replicate database so that the commit order of the transactions at the primary database is preserved. With this release of Replication Server, this sequencing of transactions commits in the stable queue is done internally in Replication Server, so that the replicate database is not burdened with this task.

This new feature removes a major bottleneck in the replicate database by moving the commit sequencing internally to Replication Server. Moreover, by minimizing the dependencies on the replicate database for transaction sequencing, it allows Replication Server to support heterogeneous databases for parallel DSI. Since sequencing is now done internally, this feature also reduces network I/O and contention in the database, resulting in faster commit processing of replicate transactions and increased throughput overall.

Note: The SMP and commit sequencing enhancements for parallel DSI will be available in a few months, and will be available as an EBF on top of Replication Server 12.5.

Security Enhancements

All wide area network (WAN) transaction systems must guarantee a safe and secure environment. In this regard, all communication to and from the database server as well as communications between servers must be secure. This release of Replication Server addresses this need through the new "advanced security" option that supports SSL encryption of data passing between servers.

The Secure Socket Layer (SSL) protocol is the industry standard protocol used over the Internet for integrity checking, digital certificates to identify users, and encryption mechanisms to protect data from unauthorized users. Industry standard "X.509" v3 digital certificates are now supported in Replication Serer 12.5 and client communication. The security provided by SSL complements the PKI-based security through server and client authentication, encryption of network data and integrity checking. Replication Server 12.5's capabilities in this area include:

- SSL-based security using PKI on all communication channels (connections from ASE to Replication Server, connections from Replication Server to Replication Server).
- SSL support that uses PKI and digital certificates for authentication, encryption, and end-to-end security.
- SSL use on specific connections or routes.

Flexibility Enhancements

Unicode Support

Since our business world has become more global, there is an increase of Asian and other language content being stored in databases. ASE 12.5 provides for UCS-2 Unicode in the form of new datatypes called "Unichar" and "Univarchar." Replication Server's support for replication of these new datatypes keeps users providing data to ASE 12.5 environments to any location, in multiple languages and different type character sets. Support includes the ability to create subscriptions on these data types and in general, treating these data types no different from char/varchar data types.

Replication Server 12.5 supports "utf8" as an additional character set that can be specified as the default character set for the server runtime environment. By running all Replication Servers in the environment with the utf8 character set, users minimize data conversions and thereby minimize data loss.

LDAP Support

Many companies need a standard, central way of managing directory information. Replication Server 12.5 now supports LDAP directory services, meaning that server information previously required to be in an interfaces file can be looked up from an LDAP server. This greatly improves the manageability of large installations.

Extended Limits

Replication Server 12.5 fully supports replication of extended limits data, of wide columns, wide tables and wide stored procedures. The new limits are available in both warm standby and full replicate environments. They provide:

- Column widths up to 32K bytes
- Tables up to 1024 columns
- Stored procedures up to 1024 parameters
- Elimination of internal limitation of replication message length to 16K bytes
- Support for larger page sizes in ASE (2K, 4K, 8K or 16K)

Continued on page 23

The Complete Sybase ASE Quick Reference Guide

Reviewed by Anthony Mandic

The Complete Sybase ASE Quick Reference Guide
By Rob Verschoor
Sypron Publications. 2nd Edition, 2002. ISBN 90-806117-1-9.

Regular readers will recognize Rob Verschoor as a frequent contributor to the *Journal*. He also maintains a set of tools and documents on his website at www.sypron.nl. This book grew out of one of his documents—a downloadable quick reference guide that he had built up over the years. The book itself can only be purchased from his website.

This second edition volume, at about 120 pages, is approximately 20 percent larger than the first edition and now includes coverage of ASE 12.5 up to and including the 12.5.0.1 Interim Release (see page 2 for a more extensive article on this subject). The second edition still maintains the same handy, pocket book format, and convenient layout as the first. But, given the density of the information and tips on each page, there is very little room to make marginal notes. So, a worthwhile addition has been the inclusion of a few blank pages at the back for notes.

The book is divided into four sections: Developer Topics, DBA Topics, Advanced DBA Topics, and Miscellaneous Topics. For each entry, notations indicate whether the entry is newly introduced for a particular server version or changed (either deleted or superceded). No notation indicates whether the entry is common to all three of the server versions.

The Developer Topics section covers all the Transact-SQL commands needed by developers, as well as the datatypes, operators, date and time, math, string, and system functions. Objects, compiled objects, cursors, locking, Java, XML, set commands and some stored procedures are also covered. This topic will also be of interest to DBAs.

The DBA Topics section covers devices, databases, segments, roles, logins and users, permissions, dump and load commands and concepts, configuration options, memory management topics such as caches and pools, languages, character sets and sort orders, parallel processing, **update statistics**, reorgs and sp_sysmon.

The Advanced DBA Topics section covers the dbccdb and its stored procedures, supported and unsupported dbcc calls, trace flags, RPCs, CIS, XP server, auditing, recovery, the resource governor and logical process manager, abstract query plans, replication server basics and shared memory dumps.

"Miscellaneous Topics" covers the ASE interfaces file, various related programs and their command line options and arguments, ASE environment variables, global variables, catalog stored procedures, monitor and historical server, ASE limits, licensing, guidelines to essential DBA tasks, and a list of useful resources on the Internet.

The information included is comprehensive, but due to the nature of the book it is very terse. Many may find it useful out in the field, but as Rob himself notes, it is no 1substitute for a shelf full of ASE manuals. It has made it into my briefcase and I find it most useful when visiting client sites where I don't have access to the Internet, the manuals, or the TechLib CD. ■

Continued from page 22

Heterogeneous Database Support

Replication Server 12.5 supports replication of data that exceeds even the new ASE 12.5 limits, so that replication in heterogeneous environments can meet the limits of other databases. It also supports LOB (Binary Large Objects) to and from Oracle and Microsoft SQL Server, and from IBM DB2 UDB databases.

Packaging Changes

And finally, there is new packaging for Replication Server with this release. RepServer will now be packaged as a base product and a set of options. The base product, Replication Server, allows replication between ASE databases. The advanced security option is also packaged with the base Replication Server product, but is optionally priced. This contains the SSL support feature.

Replication Server 12.5 introduces Replication Server options for heterogeneous replication for each of the following non-ASE databases: Oracle, Microsoft SQL Server, IBM DB2 UDB (Unix/NT version only), and Informix. Each option can be licensed in addition to the Replication Server base license (and the base license is required to license any of the options) and includes the Replication Agent and DirectConnect for the corresponding target database. ■