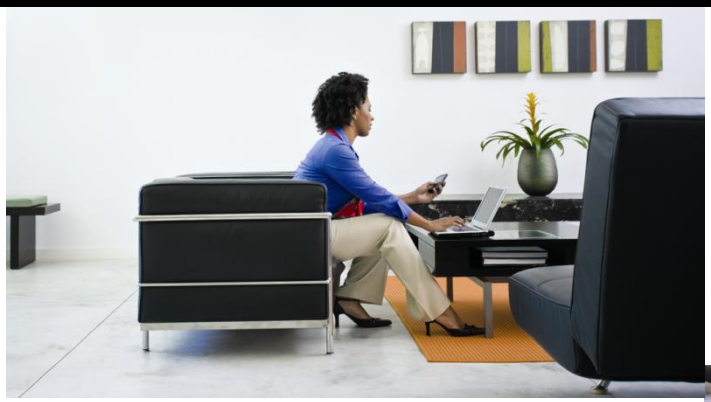
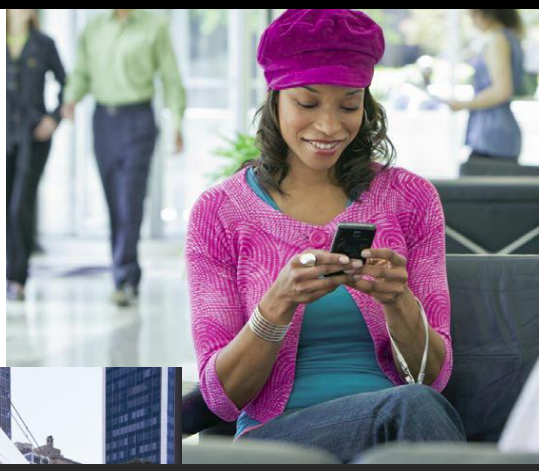
SYBASE | An SAP Company

Replication Server 15.5 Release Performance and Tuning Tips

[TECHNICAL WHITEPAPER]





www.sybase.com

Executive Summary

Replication Server (RS) 15.5 release delivers greatly enhanced performance and scalability for mission-critical replication systems. This technical white paper describes the main performance enhancements in the RS 15.5 release, as well as the performance tuning changes required to achieve the high levels of performance offered by RS 15.5 release.

The main innovation in RS 15.5 is known as High Volume Adaptive Replication (HVAR), which applies to high-volume replication into ASE as well as IQ. HVAR dramatically improves real-time replication into IQ (available as the RS 15.5 RTL edition).

Benchmarks run at our performance lab show that RS 15.5 release offers superior performance compared to pre-RS 15.5 release for both ASE and IQ targets.

- *For OLTP workload, over 200% performance gain in the ASE to ASE replication environment*
- *Over 600% performance improvement for real world application with transaction profiles dominated by 'pure inserts'*
- *Real Time Load to target IQ database using Sybase Replication from OLTP data source*
- *High concurrency users support fully utilizing SMP systems*
- *Significant reduction in device creation and replication setup*

RS 15.5 (with RTL edition) makes it possible for the first time to perform direct ASE-to-IQ replication of OLTP transactions in real time, without the need for intermediate staging steps, thus greatly reducing the complexity of replication systems involving IQ.

1 Introduction

Sybase Replication Server is an industry standard for real-time transaction replication software and data distribution solution; it moves and synchronizes data across the distributed enterprise. Replication Server is capable of addressing challenges of enterprise information management, such as heterogeneous data replication and synchronization, real-time reporting, and disaster recovery and high availability. This allows businesses to comply with increasingly rigid industry regulations, remove risk from their distributed enterprise environments, and leverage existing investments of acquired assets.

The purpose of this write-up is to document various performance related findings and provide an understanding of different tuning techniques in RS 15.5. This document outlines the results of various performance benchmarks undertaken with RS 15.5. The findings from these benchmarks should act as a guideline for the RS administrators. Some of these guidelines will be useful not just for RS 15.5 release, but also the pre RS 15.5 releases. The goal of this exercise is to provide the RS users with a useful tool for quickly resolving performance problems, thus saving valuable time and effort.

The intended audience of this whitepaper is:

- Users wanting to know performance features introduced in RS 15.5 and how relevant these features are to their use cases. This paper, though, is not intended to have a comprehensive description of every new feature in RS 15.5. For such a list, please refer to the "*What's New in Replication Server Version 15.5"* guide.
- DBAs wanting to tune their RS System environment for maximum performance.
- In scenarios where the replication rate is less than expected or desired, this paper should help the DBAs analyze where the bottleneck lies and how that can be resolved.

It is expected that the reader is already familiar with Replication Server internal processing and basic replication terminology as described in the product manuals. This paper focuses heavily on Replication Server in an Adaptive Server Enterprise environment, though a section on IQ as a replicate database is included.

2 System Specifications

The benchmark(s) used in this paper are run on the below hardware configuration

Hardware	HP Proliant DL580
CPU	16 cores - Intel(R) Xeon(R) CPU X7350 @ 2.93GHz
Memory	32GB
OS	Red Hat Enterprise Linux Server release 5.2 (Tikanga) Linux 2.6.18-92.el5 #1 SMP x86_64 x86_64 x86_64 GNU/Linux
HDD	EMC Symmetric Disk Array Rev: 5874 Type: Direct-Access
Software	Replication Server: 15.5 (64bit) Adaptive Server Enterprise 15.0.3 (64bit)

Table 1

3 Benchmarks Used

The benchmarks used for the results shown in this white paper were run for an OLTP application with ASE 15.0.3 and RS 15.5 servers on the Linux system. An OLTP application was chosen since it is the most common business application supported by the Replication Server in real-world business environments. The other benchmark used for some of the results consists of a large number of pure inserts into the database using a mix of small and large transaction sizes. The primary ASE was set up on one machine while the replicate ASE and Replication Server were set up on a second machine (the system specifications in Table 1 are for the machine used by Replication Server). The replication of the data was done in MSA mode by creating table replication definitions and subscriptions for all the tables of the primary database. RepAgent/ASE (the Replication Agent Thread or RAT) was configured in the primary ASE for reading and transferring the log.

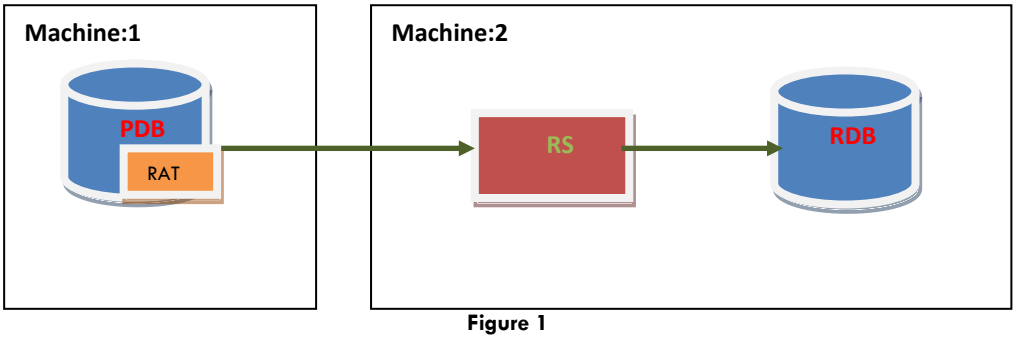


Figure 1

4 What's new in RS 15.5?

RS 15.5 is a performance oriented release. High Volume Adaptive Replication (HVAR) is a patent pending technology to improve replication performance dramatically in a high volume OLTP environment. In versions earlier than 15.5, Replication Server sends each operation (insert/delete/update) to the replicate database directly, row-by-row and in log order in the continuous replication mode (with the exception of stored procedure replication and SQL Statement Replication). The HVAR feature in RS 15.5 uses compilation and bulk apply processes that result in data reduction and achieves better performance compared to the continuous replication mode:

- Compilation – rearranges replicate data, by clustering it by each table, and insert, update, and delete operations, and then compiling the operations into net-row operations.
- Bulk apply - applies the net result of the compilation operations in bulk using the most efficient bulk interface (this is facilitated by the above Compilation's clustering mechanism) for the net result. This is much more efficient than sending the row changes individually.

Replication Server uses an in-memory net-change database to store the net row changes which it applies to the replicate database. Instead of sending every logged operation, compilation removes all the intermediate operations and sends only the final states of a replicated transaction. HVAR is especially useful for creating online transaction processing (OLTP) archiving and reporting systems where the replicate databases have the same schemas as the primary databases.

Other salient performance optimizations in RS 15.5 are (many of these changes are purely internal to RS and transparent to the user and may not be documented):

- Optimizing the Distributor module to read directly from the Stable Queue Transaction(SQT) cache
- Eliminating the expensive overhead of LRU chains for fully cached RSSD tables
- Reducing the number of block allocations and de-allocations by a larger SQM block size
- Making parallel the parsing and normalization activity of EXEC module by separating it into two threads
- Eliminating unnecessary scheduling overheads in the case of SMP
- Reducing the contention on internal thread resource locks.
- Improving the DSI module by overlapping the execution in the database server for the current command with the preparation in Replication Server of the next command to be sent to the database server.
- Avoiding spin locks by implementing assembly language atomic add/delete for hot variables.

Figure 2 shows some of the results of the benchmarks that were run in-house.

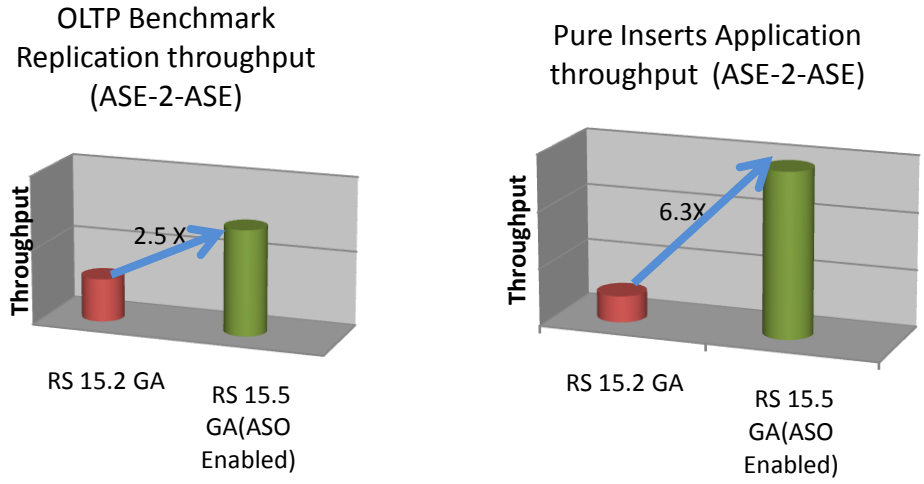


Figure 2

In addition to the above enhancements some IO related optimization were done in RS 15.5 to dramatically reduce the setup time of the stable devices

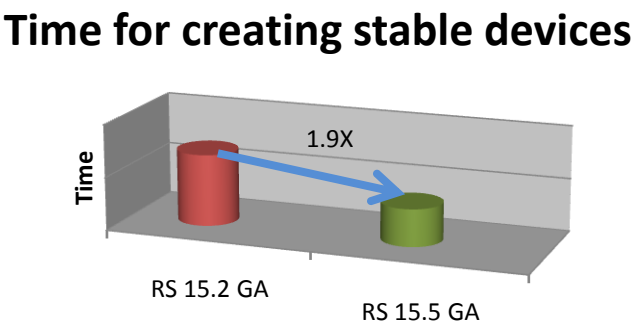


Figure 3

The rest of this paper talks about various performance analysis tips for identifying bottlenecks in both ASE-to-ASE as well as ASE-to-IQ replication. It suggests ways to tune Replication Agent/ASE, Replication Server and the replicate ASE or IQ for better performance and also introduces the new performance features of RS 15.5.

5 Primary Data Server

As far as replication is concerned, the Replication Agent is the main module of interest at the primary data server.

5.1 Analysis of Replication Agent

In ASE, the Replication Agent is implemented as an ASE task. This thread scans the log generated by user transactions, converts the log into Log Transfer Language (LTL) and sends it to the Replication Server. The next few sections describe in detail some of the common performance issues observed in Replication Agent and some ASE or system adjustments which could be useful in addressing these issues.

5.1.1 Communication with Replication Server

The first thing to look at while analyzing Replication Agent performance is the amount of time it waits for responses from Replication Server. This data can be obtained from the Replication Agent section of ASE's sysmon report under „I/O waits from RS' tab.

	per sec	per xact	count	% of total
I/O Wait from RS				
Count	n/a	n/a	23142	n/a
Amount of Time (ms)	n/a	n/a	69723	n/a
Longest Wait (ms)	n/a	n/a	296	n/a
Average Wait (ms)	n/a	n/a	3.0	n/a

Table 2

The data in Table 2 is from a benchmark in which Replication Agent required around 82 seconds to read the log, create LTL and send to RS. As the above data shows around 70 seconds were spent in waiting for responses from Replication Server.

There are three aspects to be considered to reduce this I/O wait time of Replication Agent:

1. Replication Server (the EXEC module's) processing time
2. Network between Replication Agent and Replication Server
3. ASE scheduling issues.

Ways to improve the EXEC module's performance will be discussed in detail in section 6.4. ASE scheduling issues are discussed in section 5.1.3, while dealing with network latencies is described in section 5.1.4. In this section, some of the ways of reducing the amount of interaction between Replication Agent and Replication Server are described.

The amount of "handshaking" between Replication Agent and Replication Server is mainly governed by new Replication Agent configuration parameter „ltl batch size'. This parameter sets the maximum size, in bytes, of LTL data that a Replication Agent can send to the Replication Server in a batch. The value of „ltl batch size' ranges from 16,384 to 2,147,483,647 bytes. Its default value is 16,384 bytes. Prior to ASE 15.0.3 this value was

hard coded to 16,384 bytes. Replication Agent performance can be improved by increasing the LTL batch size to a bigger number. At the end of each LTL batch, Replication Agent checks for errors in the previous batch. Increasing the LTL batch size decreases the frequency of Replication Agent checking for LTL errors. This can increase the amount work that needs to be redone by Replication Agent in case of retrial errors. However, in the absence of errors, increasing this parameter is quite useful, especially when Replication Agent and Replication Server are located at geographically different locations or when they are on a slow network. The graph below illustrates how the „IO waits from RS' decreased as „ltl batch size' was increased for such a benchmark setup.

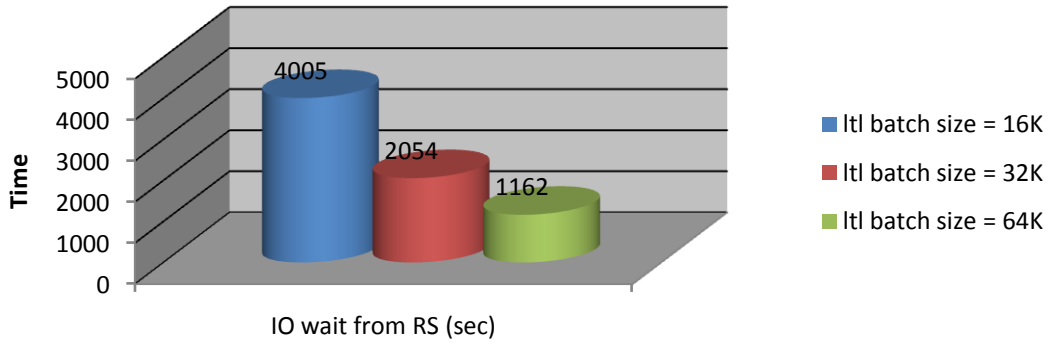


Figure 4

5.1.2 Amount of time spent in reading log

Table 3 shows the part of Replication Agent section of sysmon report which indicates the amount of time Replication Agent spends in reading log:

	per sec	per xact	count	% of total
		Log Scan Summary		
Log Records Scanned	n/a	n/a	4244984	n/a
Log Records Processed	n/a	n/a	2693452	n/a
Number of Log Scans	n/a	n/a	1	n/a
Amount of Time for Log Scans (ms)	n/a	n/a	3523	n/a
Longest Time for Log Scan (ms)	n/a	n/a	3523	n/a
Average Time per Log Scan (ms)	n/a	n/a	3523.0	n/a

Table 3

If the amount of time for log scan is found to be high and the physical reads are observed in the ASE sysmon report for the corresponding log device, try increasing the cache or configure log cache as shown below. This should lower the log scan time and improve the Replication Agent performance.

sp_cacheconfig 'logcache', '<size>', logonly
go
sp_bindcache logcache, '<pdb>', syslogs
go

Replication Agent needs to scan the log pages continuously. The current page of the transaction log is written to disk when transaction commits. The objective in sizing the cache for the transaction log is not avoiding writes. Instead, try to size the log to reduce the number of times that Replication Agent that need to read log pages must go to disk because the pages have been flushed from the cache, this will eliminate the expensive disk reads. When sizing a cache for a transaction log, start with a small cache and try increasing to the value where reads were not seen from the primary database log device.

Note: The server may get intermittent bursts of batch processing and this may require much more log to be cached (sometimes there may not be enough memory to cache all the log pages). If the cache is configured for these loads, cache may be underutilized and wasted.

5.1.3 ASE Scheduling

It is observed that when ASE runs at more than 90% CPU busy, the performance of Replication Agent degrades badly. One of the causes of this problem is related to the use of deferred CT-lib I/O mode by Replication Agent to communicate with Replication Server. Note that after issuing each CT-lib I/O, the Replication Agent task in ASE has to yield. Completion of this I/O is polled using ct_poll () call from scheduler context. The more CPU busy the ASE engine is, more is the delay expected before ASE scheduler gets a chance to poll for completed CT-lib I/O.

In ASE 15.0.3 onwards (and in ASE 12.5.4 ESD#8), a new Replication Agent configuration parameter *„bind to engine'* was added to bind Replication Agent to an ASE engine. If one engine could be left exclusively for Replication Agent and other user applications could be bound to the remaining engines, then using this parameter, Replication Agent could be bound to the engine left for it. With no other ASE task running on this engine, the scheduler should get chance to poll for completed CT-lib I/Os much more frequently. This „bind to engine' parameter is also useful when there are more than one Replication Agent threads running in the same ASE server and DBAs need to manually perform some load balancing by distributing these Replication Agents to different engines.

Another issue related to scheduling could come up when Replication Agent is the only task running on ASE. After the Replication Agent task sends something to Replication Server, it goes to sleep waiting for response. When the ASE scheduler finds that there is no runnable task, it may relinquish the CPU after spinning for a stipulated number of times looking for runnable task (the number of times it spins is governed by ASE configuration parameter '*Runnable Process Search Count*'). If the response to Replication Agent's communication comes during this interval, it may not be received by Replication Agent until the ASE engines start running again. This may impact Replication Agent performance. One workaround to deal with this is to try with *'Runnable Process Search Count'* configuration parameter in ASE.

5.1.4 Tuning the network between Replication Agent and Replication Server

When **'ltl batch size'** is set to a high value (typically, larger than 64K), Replication Agent would be sending a lot of data to Replication Server in a batch. However, the data that is being sent is buffered by the TCP layer at the source machine and this data cannot be discarded until the TCP-level acknowledgement is received from the destination machine. If the window size (the size of the buffer) used by the TCP layer is small (the default value is 16K on Linux), then it may be full when Replication Agent is trying to send the next packet. The send of this packet will not happen until the TCP layer frees up some space in its buffer after receiving acknowledgement for the previously sent packets. The delay in sending the packets will cause further delay in receiving acknowledgement for this packet and finally in receiving the response from Replication Server at the end of the batch. This can significantly affect performance if the network between Replication Agent and Replication Server is very slow (10MB/s or lesser; for e.g., when Replication Agent and Replication Server are running on machines which are at geographically different locations). The delay in sending command batches can be monitored using the Replication Agent raw counter ra_sum_io_send_wait.

field_name	group_name	field_id	value
ra_io_send	repagent_4	7	17879
ra_sum_io_send_wait	repagent_4	8	652118
ra_longest_io_send_wait	repagent_4	9	766
ra_io_recv	repagent_4	10	3998
ra_sum_io_recv_wait	repagent_4	11	537161
ra_longest_io_recv_wait	repagent_4	12	503

Table 4

Table 4 shows a section of raw monitor counter output collected for a Replication Agent run of 20 mins (1200 secs). The value column in the above table is reporting time in milliseconds. The total time spent for just sending the packets is 652 sec out of 1200 sec. This is quite high. In such cases, it might be useful to increase the size of TCP buffers at the sender machine using the TCP tunable(s) net.ipv4.tcp_wmem and net.core.wmem_max.

6 Replication Server

This section describes some standard tunings to be considered in Replication Server. It talks about module-by-module analysis to determine and resolve bottlenecks within Replication Server. Since Replication server counters are often used in this paper as means of analyzing performance issues, section 6.1 explains briefly how to collect the counters and interpret their values.

6.1 Using counters to monitor performance

Replication Server has several hundred different counters that can monitor performance at various points and areas in the replication process. By default, most counters are not active until the user chooses to activate them. To view descriptive and status information about Replication Server counters, use the rs_helpcounter stored procedure.

There are several options for gathering counters. Generally, the counter data can be collected by executing „**admin stats'** with the „**save'** option which instructs Replication Server to collect information for a specified number of observations within a specified time period and save that information to the RSSD. For example, to start sampling and saving statistics to the RSSD for one hour at 20-second intervals, enter:

admin stats, "all", save, 20, "01:00:00"

The **rs_dump_stats** stored procedure dumps the counter data from the relevant tables of RSSD to a CSV file that can be loaded into a spreadsheet for analysis. Table 5 shows a snapshot of an output of rs_dump_stats stored procedure. (Comments to the right of the output are included to explain the example. They are not part of the rs_dump_stats output.)

```
…
CM: Outbound non-database connection requests    *Counter external name*
CMOBNonDBReq                                     *Counter display name*
13004           , , 13,  -1                      *Counter ID, instance ID,
                                                     instance value*
Nov  5 2005 12:29:18:930AM,  103,  103,  1,  1   *Dump timestamp, observed,
Nov  5 2005 12:30:28:746AM,  103,  103,  1,  1   *total, last, max values for
Nov  5 2005 12:31:38:816AM,  107,  107,  1,  1   *the counter*
Nov  5 2005 12:32:49:416AM,  104,  104,  1,  1
Nov  5 2005 12:33:58:766AM,  114,  114,  1,  1
...
Nov  5 2005 12:46:51:350AM,  107,  107,  1,  1
ENDOFDATA                                        *EOD for counter*
…
```

Table 5

6.2 SMP Enable

In RS 15.5 release, by default, the 'smp_enable' parameter is set to „ON'. Setting the parameter 'smp_enable' to 'ON', enables the Replication Server's multithreaded architecture to make use of multiple cores efficiently on an SMP system.

configure replication server set smp_enable to 'on'|'off'
go

In the benchmark on a sixteen core (Intel(R) Xeon(R) CPU X7350 @ 2.93GHz) machine, a significant gain in throughput was seen with smp_enable set to „ON' for the pure inserts benchmark.

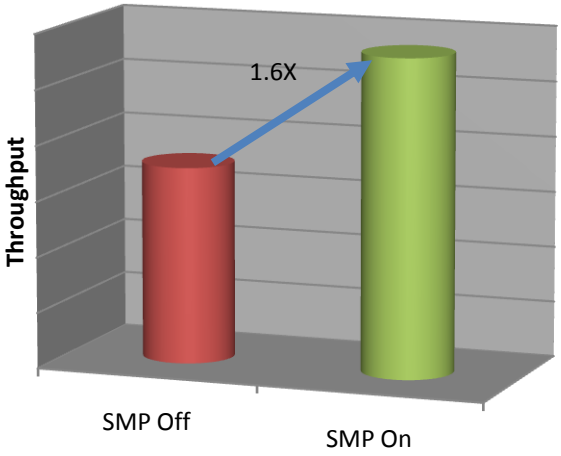


Figure 5

A comparison of output of 'top' command with and without smp_enable for the benchmark runs is shown in Table 5. It shows how Replication Server made use of more cores when smp_enable is „ON'.

'TOP' **command is run several times and it's output is captured at various intervals, each line of output represents the server's load at that interval.**

SMP: OFF

```
PID    USER      PR   NI VIRT  RES  SHR  S  %CPU %MEM    TIME+  COMMAND

17065 sybase    17   0 8921m 4.5g 9304 S   99 14.4   1:29.57 repserver
```

SMP: ON (It is evident from the below data that RS with up to ~488% CPU utilization is using 5 cores)

```
PID   USER      PR   NI VIRT  RES SHR  S %CPU %MEM    TIME+  COMMAND

18708 sybase    15   0 2152m 1.3g 9280 S  488  4.2  16:10.78 repserver
```

Table 6

6.3 System Table Services (STS)

The System Table Services (STS) module is the interface between the Replication Server and the RSSD database. The STS module is responsible for submitting all SQL to the RSSD. This module's performance can be improved by caching RSSD data locally, which may help speed internal Replication Server processing.

In RS 15.5 three new objects, 'rs_objects', 'rs_columns' and 'rs_functions' are added to the existing STS default-fully-cached object list. Further performance enhancements are made to reduce the internal processing overhead for the fully cached tables.

In the STS monitor counters, if higher value is seen for the counter STSCacheExceed or if the warning message shown below is seen in errorlog, it is an indication that the configuration parameter 'sts_cachesize' needs to be increased. The counter STSCacheExceed is incremented every time there is a cache overflow leading to replacement of an existing entry.

I. 2010/01/27 17:32:48. A cached row for system table 'rs_config' was swapped out of the cache in order to accommodate another row. If this message appears multiple times it may be advisable to increase sts_cachesize from its present value of '5'.

```
STSCacheExceed
11008, , 11,  -1
Jan 27 2010 05:33:08:696PM,  0,  0,  0,  0
Jan 27 2010 05:33:38:845PM,  365,  365,  1,  1
```

Table 7

The performance of STS module can be further improved by identifying the hot (frequently accessed) tables of the RSSD and then configuring them to be fully cached, so that data from those tables is always available within Replication Server without the necessity of requesting the information from the RSSD. For example if the rs_columns table is identified as a hot table, the following command could be used to cache it fully in STS cache:

configure replication server set sts_full_cache_rs_columns to 'on'
go

6.4 Analysis of EXEC module

This section describes the data points to be looked at in EXEC module if it turns out to be a bottleneck.

6.4.1 Separate NRM thread

RS 15.5 introduces a new thread called NRM thread to share the work of EXEC thread. The purpose of this new thread is to improve the turnaround time to the Replication Agent so that its performance and thus the overall inbound performance is improved. The pre-RS 15.5 EXEC thread used to parse, normalize, pack the incoming batch of LTL commands and queue them for writing to stable queues before responding back to Replication Agent. In RS 15.5, the EXEC thread only parses the commands and passes them on to the NRM thread for further processing. Thus it can respond back to the Replication Agent much faster now.

The graph in Figure 6 shows the improvement in Replication Agent performance for a benchmark, with and without NRM thread.

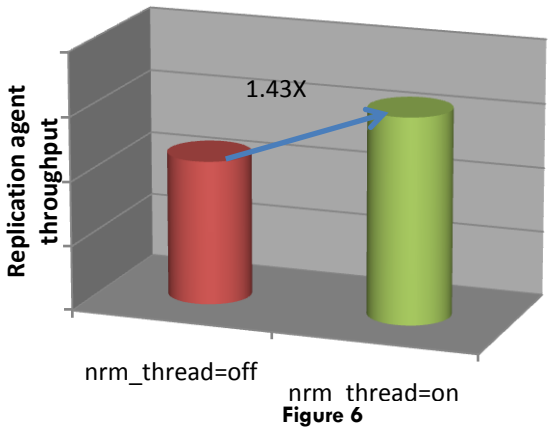


Figure 6

6.4.2 Amount of time waiting for writes

The counter RAWritesWaitTime of Replication Agent section in Replication Server stats indicates the amount of time EXEC module had to wait for writes to finish.

If this counter shows up only occasionally, then this could be fixed by increasing the configuration parameter *exec_sqm_write_request_limit*. This parameter controls the maximum number of outstanding bytes the EXEC thread can hold before it must wait for some of those bytes to be written to the inbound queue. The default value is 1M, which means that EXEC thread blocks only when the number of bytes waiting to be written exceeds 1M.

However, if the counter RAWritesWaitTime shows high value consistently, then it means that the speed of SQM writer module is lower than the speed at which EXEC module can process data and so SQM Write thread is not able to keep up. Section 6.6.2 describes some of the ways of improving the speed of writing data to stable queues.

6.4.3 Execution of 'get truncation' command

After sending every „*scan batch size*' number of commands, the Replication Agent queries Replication Server for the latest secondary truncation point („*scan batch size*' is a configuration parameter of Replication Agent). To do this, it sends a special packet containing the command „get truncation'. When the EXEC thread in Replication Server receives this packet, it executes this command itself (unlike other LTL commands which it passes to NRM thread). It fires a transaction called Save_Locater in RSSD to update the last written OQID (Origin Queue IDentifier) in rs_locator table and returns the same to Replication Agent.

It has been observed that the execution of Save_Locator transaction is relatively expensive if the RSSD is an embedded RSSD. In such cases, increasing „scan batch size' parameter of Replication Agent could be useful. From the performance point of view, the larger the scan batch size, the better, since that would mean lesser communication with Replication Server and lesser executions of Save_Locator transactions. However, it could delay the movement of secondary truncation point in the primary server, which could delay the log from being truncated.

Generally, a frequency of around one OQID update per minute is a reasonable balance so as to not have too frequent executions of Save_Locator transactions in RSSD and at the same time ensure that primary ASE's secondary truncation point moves forward frequently enough for ASE to be able to truncate log before it gets full. The Replication Server counter UpdsRslocater could be used to identify the number of OQID updates in RSSD happening from EXEC module.

Figure 7 shows the gain obtained in one of the benchmarks by increasing „scan batch size' from 1000 to 20000.

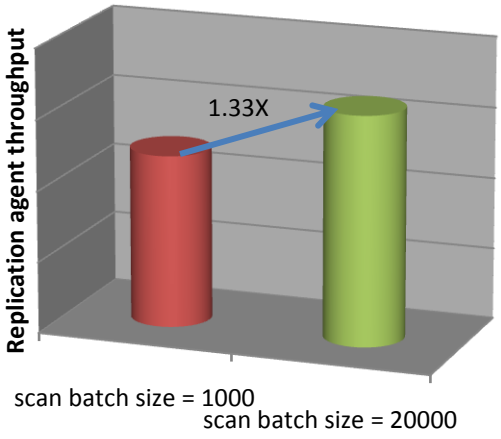


Figure 7

6.5 Analysis of DIST module

6.5.1 When SQT cache is not big enough

If a non-zero value is seen for the counter TransRemoved and/or a warning message like the one shown below is seen in the Replication Server error log, it could be an indication that the SQT cache configured is too low for the current needs.

```
2009/10/28 23:49:13. WARNING #24068 SQT(102:1  DIST PRIM.tpcc) -
t/sqtint.c(1370)
SQT cache size is too low to load more than one transaction into the cache.
```

```
SQT: Transactions removed from memory
TransRemoved
Jan 25 2010 12:35:08:362PM,  0,  0,  0,  0
Jan 25 2010 12:35:23:471PM,  8,  8,  1,  1
Jan 25 2010 12:35:38:811PM,  12,  12,  1,  1
Jan 25 2010 12:35:53:860PM,  17,  17,  1,  1
```

Table 8

The non-zero numbers above indicate transactions whose constituent commands have been removed from the SQT cache. Removal of transactions could be caused by a single transaction exceeding the available cache or a large transaction in the cache whose commit is not seen yet. SQT module, when loading the cache, if the SQT cache is full and it has no Closed Transaction (Transaction whose commit is already seen is a Closed Transaction) or Read Transaction (Transaction whose contents have been read by DIST/DSI is Read

Transaction), then the Open transaction list is scanned, to locate the largest transaction and the memory consumed by this transaction is freed. This will affect the performance since freed-transaction needs to be read from the disk later. Try increasing the SQT cache size to make this counter value to zero or until the message disappears. Note, however, that if the transactions are too large (containing millions of commands), then no reasonable size of SQT cache will be able to cache it.

6.6 Analysis of SQM module

6.6.1 Configurable queue block size

Replication Server 15.5 onwards, the queue block size can be increased by user to allow more data in a single block. In earlier releases, the block size was fixed at 16 KB. In Replication Server 15.5, the block size can be increased up to 256KB. Increasing block size not only improves I/O read and write rates, but also reduces processing time since it would require lesser allocation and deallocation of segments. This should improve replication performance significantly whenever SQM module is the bottleneck.

configure replication server set block_size to '<size>' with shutdown
go

As shown in Figure 8, in the internal benchmark testing, around 15% gain was observed in the end-to-end replication throughput by increasing block size from 16 KB to 128 KB.

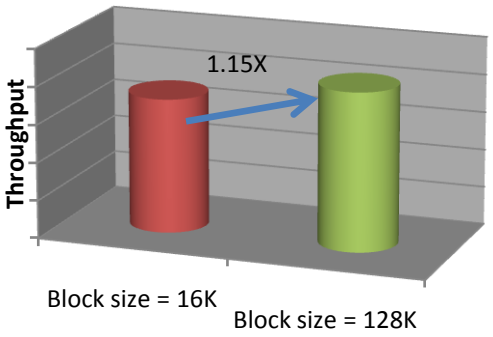


Figure 8

6.6.2 Are the writes fast enough?

The Replication Server counter SQMWriteTime indicates the amount of time taken for writing data onto the stable queues. Since this writing is done by a separate thread (the SQM thread), whether writes are bottleneck or not depends on the amount of time its clients have to wait for writes to finish. For e.g., for the inbound side, where the client is EXEC module, the EXEC counter RAWriteWaitsTime and for the outbound side, where the client is DIST module, the larger values of DIST counter DISTTDDeliverTime indicates this.

If the writes are found to be bottleneck, other than increasing the block size as explained in section 6.6.1, there are two other Replication Server parameters that could be considered for making writes faster – sqm_write_flush and sqm_page_size.

The configuration parameter sqm_write_flush is applicable only when the stable queues are on file system. It controls whether or not the writes to memory buffers are flushed

to the disk before the write operation completes. By default this is set to 'on' which means writes are flushed from memory buffers to the disk before write operations complete.

Write performance can be improved by setting this parameter to „dio'. This will make Replication Server use direct IO for writes and reads to stable queues thus bypassing the file system buffers and thus eliminating the overhead of manipulating them.

The parameter sqm_page_size sets stable queue page size in blocks of "block_size" per page. For Example setting sqm_page_size to 16 instructs Replication Server to write to the stable queue in "16 * block_size" chunks. By increasing the value of sqm_page_size, more data could be written to stable queues per write call and the performance increases significantly. The graph in Figure 9 compares the SQMWriteTime for a test run with stable queues on file system.

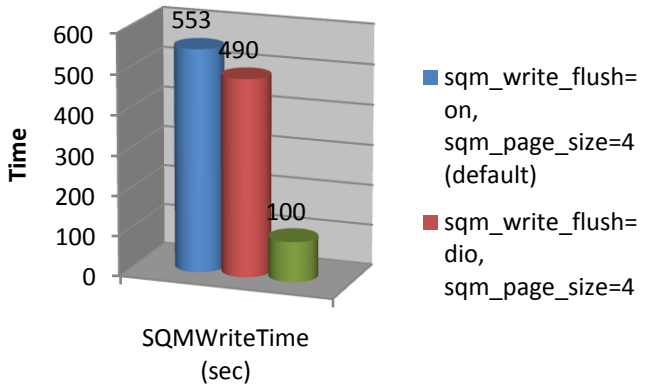


Figure 9

6.6.2.1 Are the stable queues big enough?

Sometimes the clients of SQM Writer thread may seem to be waiting for a long time even though the actual writes are not taking that much time. For example, RAWriteWaitsTime may be seen as too high, though the SQMWriteTime for the corresponding stable queue may not be so high. This scenario happens when the stable queue had got full and all the subsequent writes had to wait for some segment to get free and this caused slowness of writes. The following message in errorlog indicates that this is the problem.

W. 2009/06/09 04:25:16. WARNING #6047 dSEG( ) - qm/sqmsp.c(896)

SQM_ADD_SEGMENT ('103:0'): Going to wait for a segment to be freed. Administrative action is needed.

This also shows up in the counter SQMWaitSegTime, which gives the amount of time the SQM writer thread had to wait for segment allocation.

```
SQM: Time waiting for segment allocation
SQMWaitSegTime
6059              , , 102,  1
Jun  9 2009  3:55:21:086AM,  1,  14,  14,  14
...
Jun  9 2009  4:00:53:633AM,  63,  24524,  209,  691
Jun  9 2009  4:01:23:726AM,  64,  26207,  409,  648
Jun  9 2009  4:01:53:930AM,  72,  26639,  447,  534
Jun  9 2009  4:02:24:236AM,  63,  26636,  423,  710
```

Table 9

In such scenarios, it may be useful to increase the size of stable queues.

6.6.3 Are the reads fast enough?

6.6.3.1 Are the reads satisfied from sqm cache or not?

The counter SQMRReadTime gives the amount of time SQMR module had to spend to read blocks from stable queues. Note that unlike SQM writer thread, SQMR is not a separate thread but is part of the client thread itself (SQT for inbound and DSI-S for outbound). A high value for SQMRReadTime indicates a significant time of the calling thread being spent in just reading the blocks.

```
SQMRReadTime
62011, , 102,  11
Apr 01 2010 02:53:15:412AM,  0,  0,  0,  0
Apr 01 2010 02:53:45:425AM,  6050,  4197,  0,  51
Apr 01 2010 02:54:15:738AM,  15688,  11866,  0,  54
Apr 01 2010 02:54:45:786AM,  16281,  12061,  0,  68
Apr 01 2010 02:55:15:839AM,  20105,  16434,  1,  54
ENDOFDATA
```

```
SQMR: Blocks read from queue
BlocksRead
62002, , 102,  11
Apr 01 2010 02:53:15:412AM,  0,  0,  0,  0
Apr 01 2010 02:53:45:425AM,  15372,  15372,
1,  1
ENDOFDATA
```

```
SQMR: Blocks read from cache
BlocksReadCached
62004, , 102,  11
Apr 01 2010 02:53:15:412AM,  0,  0,  0,  0
Apr 01 2010 02:53:45:425AM,  9280,  9280,  1,  1
ENDOFDATA
```

Table 10

In an ideal situation, all the reads should be satisfied from SQM cache and physical reads should not be required. However, this is often not the case. The effectiveness of SQM cache can be monitored by comparing the counters for number of blocks read from cache (BlocksReadCached) to the total blocks read (BlocksRead).

If all the reads are satisfied from SQM cache, the values of BlocksReadCached and BlocksRead would be exactly same. If the counters indicate that not many reads are satisfied from cache, it may be worth trying to increase the SQM cache using the configuration parameter „sqm cache size'.

But note that it is not always possible to ensure that all the reads are satisfied from cache. If the client of SQMR module is not able to consume data faster than the rate at which client of corresponding SQM writer is producing data, then eventually, the SQM cache will have to be overwritten with new data before older blocks are read from it and so subsequent reads will have to go to disk. Increasing SQM cache is useful when there could be intermittent burst in data volumes.

6.7 Analysis of DSI module

6.7.1 High Volume Adaptive Replication (dsi_compile_enable)

The current replication process is continuous i.e. replicate every logged row changes. HVAR (High Volume Adaptive Replication) is a new replication methodology where only the net row changes are replicated. HVAR replication processes data as follows:

1. Grouping large number of transactions into a single transaction.
2. HVAR compilation tracks log-ordered row-by-row changes and converts them into net row changes. Net row changes are stored in net change database as three tables: insert table, update table, and delete table. All tables use primary keys defined by replication definitions. For tables without replication definitions, all columns are used as primary keys.
3. Bulk applying compiled transactions. Net row changes are sorted as per replicate table and per operations (insert/update/delete), and are ready for bulk interfaces. Since IQ and ASE do not support bulk update and delete, HVAR replication bulk inserts net changes into work tables inside replicate database and then performs join-update or join-delete with replicate tables.

Siginficant gain could be achieved with this feature in replication throughput while replicating OLTP transactions' log. The chart in Figure 10 gives the distribution of the total commands sent as per the batch size of the bulk operations which they were part of, for an in-house experiment.

As seen in this chart, more than 50% of the commands were sent in bulks of 1000-5000 commands. Note that these are 1000-5000 inserts or deletes or updates on a particular table (and not just miscellaneous inserts/update/deletes on different tables just batched together, as used to be the case before HVAR) sent at one go using the bulk API. This is significantly better than sending these operations separately using language commands and this is where the performance improvement with HVAR comes from.

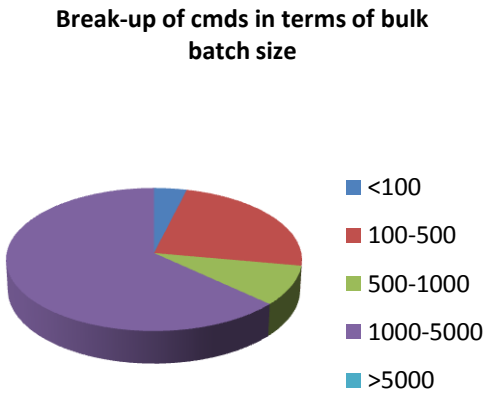


Figure 10

The maximum number of commands that HVAR can group in one transaction is governed by the configuration parameter '*dsi_compile_max_cmds*'. However, the actual number of commands that HVAR is able to group may be much less if SQT cache is not big

enough. In order to ensure that HVAR is able to group large number of transactions, the SQT cache must be configured sufficiently large.

6.7.2 Parallel DSI

In the DSI monitor counters if a large number for either "***AllThreadsInUse***" or "***AllLargeThreadsInUse***" (below is an example of the counter) is noticed, consider using parallel DSI threads by setting "***dsi_num_threads"*** or "***dsi_num_large_xact_threads"*** respectively. This counter is incremented each time a Parallel Transaction must wait because there are no available parallel DSI threads. The risk in enabling the parallel DSI is the introduction of contention between multiple DSI connections' transactions. The simultaneous application of transactions against the replicate may introduce competition between the transactions for various types of database locks and other server resources. If you notice increased deadlocks on the replicate database, try reducing the number of threads.

```
AllThreadsInUse
5057, , 103,  -1
Jan 21 2010 04:41:42:782PM,  49,  49,  1,  1
Jan 21 2010 04:41:57:999PM,  1015,1015,  1,  1
Jan 21 2010 04:42:13:032PM,  567, 567,  1,  1
```

Table 11

With HVAR, the amount of internal processing that needs to be done by DSIEXEC thread has naturally increased. So generally while using HVAR, having two DSIEXEC threads is always useful even if used with „wait after commit' serialization method. With this serialization method, though there will be no parallelism in replicate DB, the processing of one DSIEXEC thread can happen parallel while the other DSIEXEC is firing transactions into the replicate DB. Around 20% gain was observed in ASE-to-IQ replication benchmarks by turning on parallel DSI with two DSIEXEC threads.

6.7.3 Bulk copy-in

In normal replication, when Replication Server replicates data to Adaptive Server, Replication Server forms a SQL insert command, send the command to Adaptive Server, and waits for Adaptive Server to process the row and send back the result of the operation. This process affects Replication Server performance when large batches of data are being replicated, such as in end-of-day batch processing or trade consolidation.

Replication Server version 15.2 introduced support for bulk copy-in to improve performance when replicating large batches of insert statements on the same table in Adaptive Server® Enterprise 12.0 and later. Replication Server implements bulk copy-in in DSI module using the Open Client™ Open Server™ Bulk-Library.

The database connection parameters „dsi_bulk_copy' and „dsi_bulk_threshold' control bulk operations in DSI.

7 Replicate Data Server

7.1 ASE

7.1.1 Statement cache and literal autoparam

When replication is being done using language commands (unlike the dsi_compile_cmds or dsi_bulk_copy feature which use the bulk interface, or the dynamic_sql feature which uses dynamic sql), the statement cache feature of ASE is known to give significant benefit.

The statement cache allows ASE to store the text of ad hoc SQL statements. ASE compares a newly received ad hoc SQL statement to cached SQL statements and, if a match is found, uses the plan cached from the initial execution. In this way, ASE does not have to recompile SQL statements for which it already has a plan. This allows the application to amortize the costs of query compilation across several executions of the same statement.

For maximum benefit, this feature must be used with literal autoparam, which was introduced in ASE 15.0.1. In earlier versions of ASE, two queries that were identical except for one or more literal values resulted in the statement cache storing two separate query plans. ASE 15.0.1 onwards, literal values in SQL queries can be automatically converted to parameter descriptions (similar to variables). A **sp_configure** option supports this feature, which is called enable literal autoparam. Statement cache, and enable literal autoparam parameters can be configured by executing sp_configure stored procedure:

sp_configure "statement cache size", 20000
go
sp_configure "enable literal autoparam", 1
go

Configuring statement cache at replicate ASE improves replication performance considerably. The effectiveness of the configured statement cache could be determined from the replicate server's **sysmon** report as shown in Table 10.

Procedure Cache Management	per sec	per xact	count	% of total
SQL Statement Cache:				
Statements Cached	0.1	0.0	23	n/a
Statements Found in Cache	22.7	0.4	6946	n/a
Statements Not Found	0.1	0.0	23	n/a
Statements Dropped	0.0	0.0	0	n/a
Statements Restored	8.8	0.2	2689	n/a
Statements Not Cached	0.0	0.0	0	n/a

Table 12

The data in `Table 12` indicates that 6946 statements were found in cache and only 23 statements were not found in cache i.e. a large percentage of the statements were found in cache. Thus the size of statement cache was large enough.

7.2 IQ

7.2.1 Effect of indexes on tables in IQ

Benchmarks show that having indexes on IQ tables (other than the default FP index) has a negative impact on replication performance. This may be due to the fact that these indexes cause an overhead of updating them for every DML operation. However, these indexes may be needed for the day to day operations, like reporting, on the IQ server. The actual benefit and need of these indexes should be closely weighed in conjunction with impact on the ongoing replication into IQ. The join algorithms used by HVAR in the joins with worktables for replication of deletes and updates into IQ have been observed to not benefit from any index on the IQ tables.

The most commonly available indexes on IQ tables are unique HG indexes on primary key columns (they resemble clustered indexes in ASE). Figure 11 shows comparison of replication performance for the OLTP benchmark with and without having unique HG indexes on the primary key columns of the replicate IQ tables.

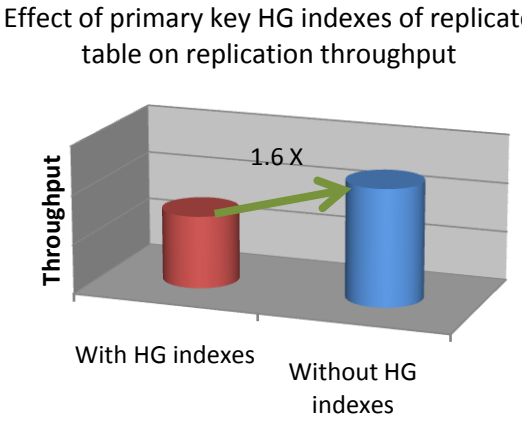


Figure 11

7.2.2 Using u2di to improve update performance in IQ

If the replication into IQ involves updates on some wide character columns (hundreds of bytes wide) of a table, it may be useful to set the Replication Server option dsi_command_convert to 'u2di' for that table for the particular connection. With this options, updates are replaced with deletes followed by inserts. Note that the option dsi_command_convert allows values to be set for a particular table – it need not be set for the entire connection. The downside of using u2di for an IQ table is that it causes some fragmentation in that table.

This tuning may not be required in future releases of Replication Server (RS 15.5 EDS#2 onwards) where replication of updates is being optimized.

Replication of OLTP benchmark involved updates on two character columns of one of the table. These columns were of 250 bytes each. Figure 12 shows comparison of performance numbers with and without the u2di option set for that table.

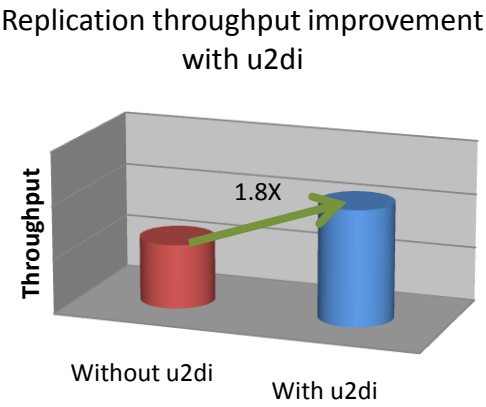


Figure 12

8 Conclusion

From the benchmark runs, it is evident that RS 15.5 offers superior performance for both ASE and IQ targets. For OLTP workloads, the gain over RS 15.2 is more than 2X in case of ASE-to-ASE replication and up to six times faster for some user applications with insert oriented transaction profiles. For the first time, replication into IQ can be seen happening in real time with RS 15.5release.

The tuning recommendations mentioned in this white paper could be applied when the corresponding data points indicating the problem are observed. The information from monitor counters mentioned throughout this whitepaper should help users understand what data points to look for while analyzing or optimizing performance of a replication system.