

### Using Transactional Replication for Off-Loading Reporting Function

*Edward Whalen, Performance Tuning Corporation*

*Geoff Langos, Verizon Online*

*Alexander Stamenkovich, Verizon Online*

#### Introduction

In the early part of 2000, extensive performance metrics were collected over several months on a call center processing system. Analysis of this data clearly showed that at the existing growth rate for the database and new user connections, the current system would run out of capacity within one year. It was determined that the current database system could reach its near fullest potential and performance capacity by year-end 2000. The applications would take much longer than one year to rewrite, which may not even solve the problem. Therefore, creative, methodical database architecture changes were needed to solve the problem quickly.

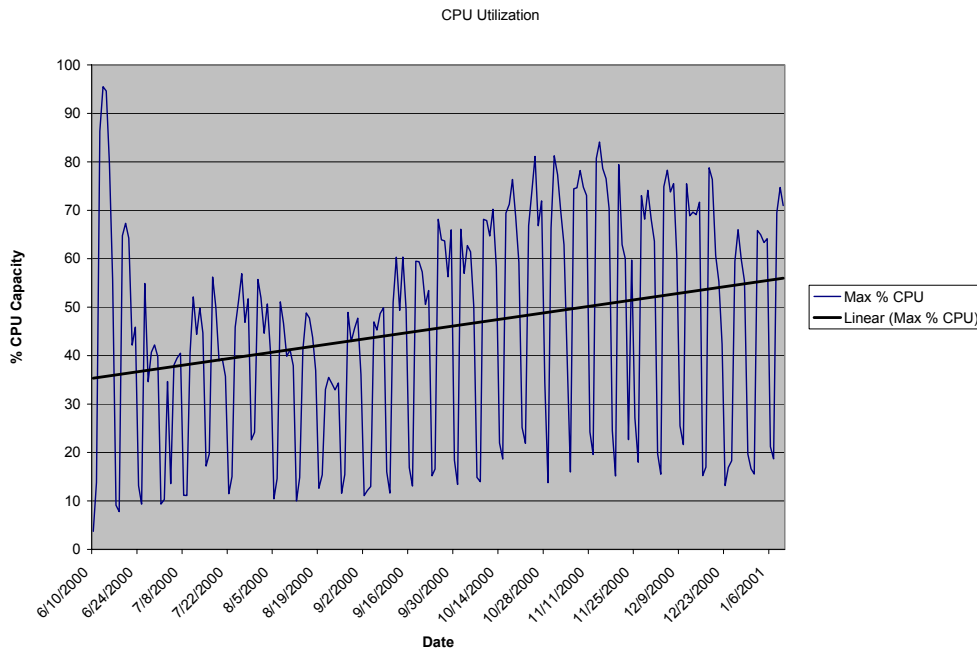
The first solution considered was to add faster hardware, but hardware technology alone was not keeping up with the growth of the system. The second solution was to optimize the database, the hardware and the SQL statements, both of which were done. Although both of these solutions helped, they would not solve the problem in the long run.

The final solution was to replicate portions of the database to multiple secondary servers and off-load specific read-only functions to those secondary servers. Initial architecture designs included replication of the entire database. This would keep design issues simple by not causing the application to be rewritten, however, the administration required to maintain the entire database was too great for the available DBA resources. Therefore, we decided to only replicate the read-only portions of the database and off-load them to secondary servers. This has not only increased the available system capacity in the short term, but also provided an innovative and scalable solution for the future.

This article will provide the reader with insight into how the problem of nearing our system capacity limit was discovered, what steps were taken, and how the final solution was implemented.

## The Problem

The first step in determining the long-term viability and capacity of any system is to implement performance-monitoring metrics. This was accomplished by taking snapshots of certain performance metrics every half-hour and storing them in a SQL Server database. Initially, this information was not extremely useful. However, once several months of data had been gathered, the information became quite valuable. The metrics included CPU utilization, IO utilization, number of users and processes, orders processes, and transactions. Over a several month period it was obvious that CPU utilization was already rising to unacceptable rates. This is illustrated in Figure 1.



**Figure 1. CPU Utilization.**

In addition to overall system metrics, several key stored procedures were instrumented with time tracking routines and their execution times inserted into a SQL Server table. These business-specific metrics allowed us to determine whether acceptable service levels were being met.

## Initial Solutions

Initial solutions involved the addition of better and faster hardware as well as performance tuning of the application's SQL and stored procedures. These solutions helped, but did not provide a long-term solution. However, this approach was used to provide immediate relief to allow the final solution to be implemented. Pain-staking work was done to capture and analyze SQL traces and metric data to find stored procedures and ad hoc SQL code that was utilizing the most CPU on the system. In order to realize system improvements from tuning, we had to research what code was taking the longest to run.

Tuning this code to improve query plans and execution times gave us the biggest improvement in the least amount of time. Tuning the wrong code, or infrequently used code, was not worth the immediate investment. We had to focus on code that was used hundreds and thousands of times a day to improve the overall transaction time of the call center application.

Eventually the longest running stored procedures and SQL statements were well optimized. Although optimization and performance tuning was considered an ongoing effort, the growth of the system was outpacing the optimization efforts. Something more drastic had to be done. The stored procedures and SQL statements were put into two different categories requiring two different optimization techniques: (1) Short-running on-line transaction processing (OLTP) and (2) Long-running functions associated with reporting.

Unfortunately these long-running reporting queries could not be off-loaded to a data warehouse or data mart. The reports were used constantly throughout the day as an integral part of the application and needed to be close to real-time. However, one advantage noted about these reports was that they were 100% read-only in nature. This led to the final solution.

## Off-Loading Reporting Functions

In the later part of 2000, we began to off-load some of the reporting functions to replicated servers. There were several key components to this project including:

- Designing the architecture
- Slightly modifying the application
- Implementing transactional replication
- Testing and tuning the solution
- Maintaining the systems

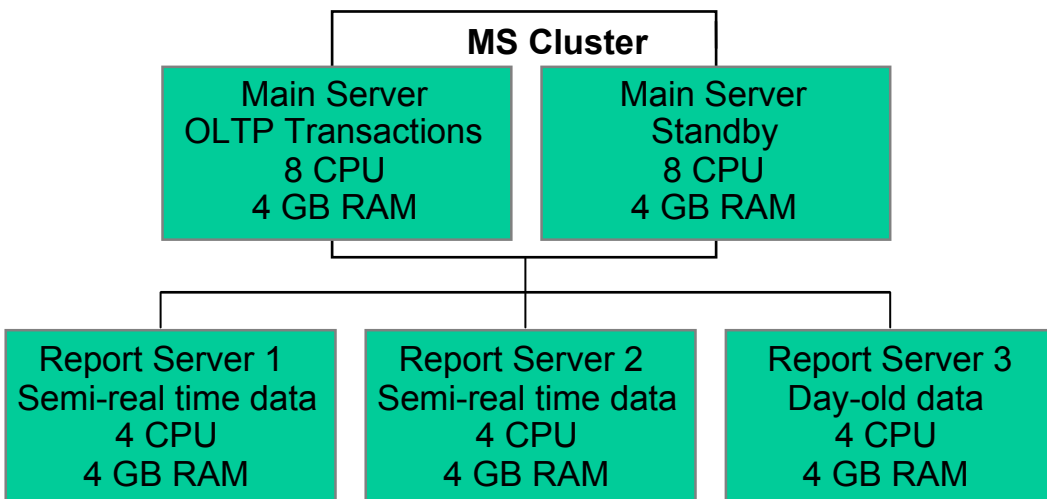
## Architectural Design

Three critical success factors were identified in the initial design: it should be dynamic, fault-tolerant and scalable. Initially, the application was programmed to determine at startup time which server would run a particular reporting function. However, if a reporting server failed after application startup, its functions would need to be rerouted. With the server location being determined during startup, the only way to correctly determine where to reroute the reporting functions would be to restart the application. This approach was not dynamic enough, so we eventually deployed a real-time look-up at the time the report is run. Although there is some overhead associated with accessing a look-up table, the performance gains more than offset this overhead.

The reporting functions within the applications were divided into three categories according to their business purpose: real-time, almost real-time, and day-old. The categories specified how current the data must be. Real-time must run on the main server since decisions are based on the instantaneous state of the database. Almost real-time can tolerate the 8 second delay involved in replication. Day-old reporting functions involve long-term data and an image of yesterday's data will suffice. This reduced the amount of data that needed to be replicated and provided lots of scalability by allowing functions to be spread out among multiple systems.

One advantage was that all the almost real-time off-loaded reports used most of the same data. This gave us the added benefit of spreading multiple reports across multiple subscribers because the data being replicated was all the same. In the event of a hardware failure or replication error we can modify the application look-up table to switch subscribers or point the report back to the main server. Currently this must be done manually. With the same load on the publisher, two reporting servers can be implemented for less cost than one clustered reporting server. This architecture is shown in Figure 2.

Although the main server was a Microsoft Cluster Server, it was unnecessary to cluster the reporting servers. In the event of a failure in a reporting server, the functions can be switched to a functional, almost real-time reporting server or run on the main server for a short period of time. This was implemented by using a table on the main server. This table contains a list of functions and the server name where they run. Whenever a user ran a report, the application read from the table and routed the queries accordingly. In addition, whenever a reporting server fails to respond, the main server is used.



**Figure 2. Architectural design**

## Modifying the Application

Since the application is written and maintained in-house, modifying the application was not impossible. However, it was still a large task. The application must be able to determine where to run each of several functions that have been replicated. This is done each time a report runs by reading a SQL server table, making a connection to the appropriate location, and disconnecting when the function is complete.

As the application is enhanced, most reporting functions are modified to use this look-up table, even though they might initially still point to the main server. This provided for greater flexibility, ability to move them at a later time, and scalability for future reporting requirements.

## Implementing Transactional Replication

Great care was taken to coordinate the SQL Server transactional replication with the application modifications. It was important to make sure that all of the necessary tables and stored procedures were replicated. This required a great deal of coordination and good communication between the development and DBA groups. Initially, the data replication was implemented before the application modifications were done in order to test the performance and configuration of replication. During testing it was determined that performance was not acceptable. By default, the value of the polling interval for both the *logreader* agent and the distribution agent was set to 10 seconds. This meant that a transaction could take as long as 20 seconds from publisher to subscriber. That latency was unacceptably long since the users were accustomed to real-time reports. By modifying the polling intervals on both *logreader* and distributor to 2 seconds that time was shortened to a maximum of 4 seconds.

One key architectural design issue was discovered while incorporating transactional replication into the change control process. By default, normal practice was to put all of the replicated objects into one publication. This made administration easier; however, it also makes change control difficult. Each time a change was made that affected a replicated object, we would have to break and recreate the publication. The solution was to make each object its own publication.

This way, when a change is made to an object, all we have to do is modify specific publications and subscriptions. We don't have to create a snapshot for the entire publication set, only the objects in question. In the long run, there are more publications to manage, but administration is much easier.

Once replication was implemented, the stored procedures in question were tested on the replicated servers as follows: The databases on the day-old server were reloaded each night from the previous day's full database backup, and hourly transaction logs were applied to insure the databases contained data up to midnight the day before. In order for this process to work effectively, operators were assigned to manually monitor the time-critical process to ensure proper availability for the next business day.

Since no one runs day-old reports during early morning hours, none of the day-old reports are pointed back to the main server; they are simply unavailable during this reload period. If times were missed and the server was not ready by the next business day, then the day-old reports were pointed back to the main server for the remainder of the day until the next run.

## Testing and Tuning

Each time a new function was moved to the replicated servers, it was thoroughly tested. It was important to test each execution path in order to make sure that a table or stored procedure wasn't missed. A full application regression test was done with each new change to the replication model. In addition, since only reporting functions were being run on these servers, it was possible to tune the indexes for only reporting functions and not OLTP.

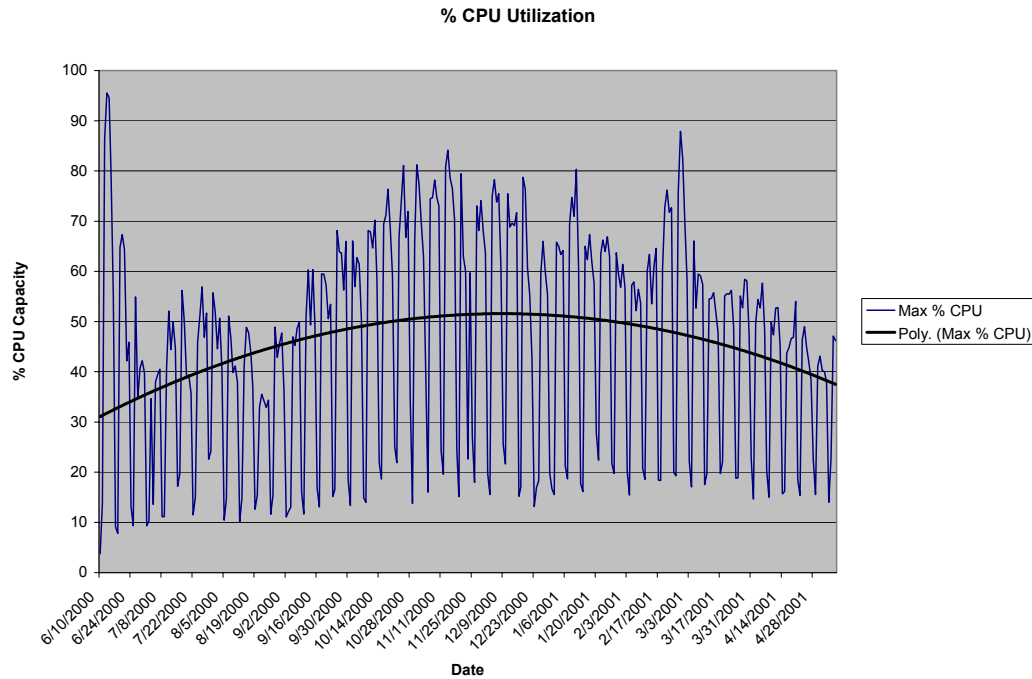
This made it possible to remove some indexes and modify others in order to achieve greater performance. This is an ongoing process that is still being done today. To keep the data and indexes in an optimum state, *DBCC Checkdb* and *Reindex* are run on a weekly basis

## Maintaining the Distributed System

Maintaining the system became slightly more difficult as servers were added. In addition to administering the main server, it was now necessary to administer a distributor and two or more subscribers. As more servers are added, the amount of administration increases. The administration of the replication architecture requires at least 30% of resources from two Senior DBAs. This amount of administration is required in order to properly and efficiently handle change requests and monitor the 'overall health and availability' of the replication publications. Table rowcounts are checked and discrepancy thresholds were established for each table publication. Table rowcounts are compared against the main server and when the discrepancy threshold is exceeded an urgent notification is placed to the database team to resolve the issue. During the testing process it was determined it was necessary to replicate user accounts and passwords to the subscribers. Since transactional replication will not and cannot do this for you, it was necessary to develop our own method of replicating user information. This was done with DTS packages. The DTS packages are executed using the SQL Server Agent to keep the login and user information up-to-date every 5 minutes. The most typical changes to user information are password change, role change (application security feature to change groups for different types of data access), and new or deactivated users. As the complexity and number of database servers increased, so did the work involved in maintaining these systems. A great DBA staff is needed to keep these systems running. Fortunately we have an outstanding DBA staff.

## Results

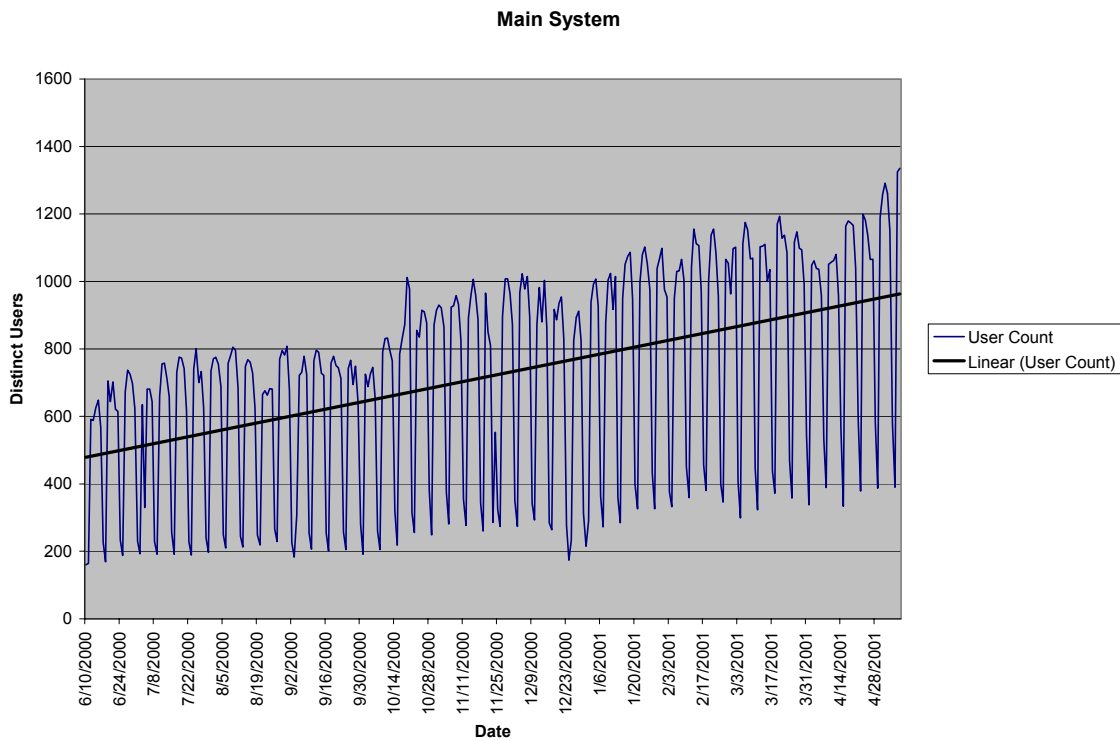
After implementing the replicated systems and slowly migrating more reporting functions off of the main server, we have seen a dramatic drop in CPU utilization as well as an increase in performance. The drop in CPU utilization is shown Figure 3. (The replication was implemented in the later part of 2000.)



**Figure 3. CPU Utilization including time period after replication was implemented.**

Although a large amount of data is being replicated very quickly, the CPU resources on the distributor are fairly low because of the efficiency of transactional replication. This CPU reduction shown in Figure 3 happened as users were continually being added to the server. (See Figure 4 for user counts.)

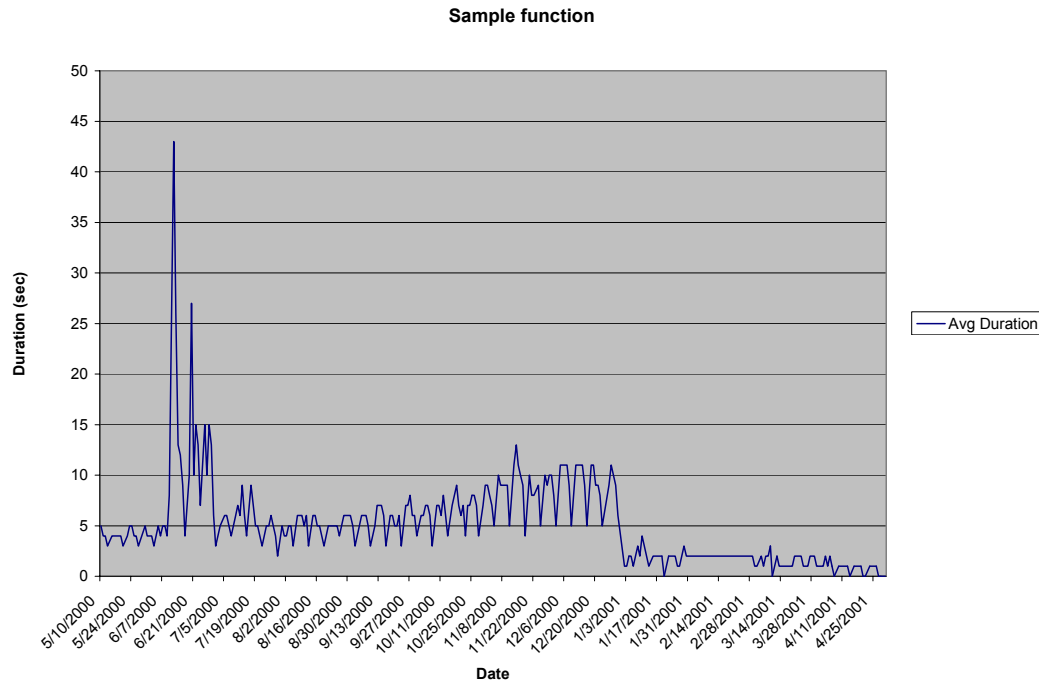
Although, in general, this solution is very successful, there are a few limitations. Some issues with replication require rebuilding a publication and resyncing the data using a fresh snapshot. We were unable to consistently create new snapshots during normal business hours with a full user connection load. This type of work is only done during non-peak business hours or during scheduled maintenance windows. This is one of the reasons why we have the ability to switch reports back to the main server when there are irresolvable issues with replication.



**Figure 4. Distinct User Counts.**

The resources used on the reporting servers are also within acceptable limits. The resources are constantly being monitored. If they begin to increase to unacceptable levels, we can simply add more subscribers, update the pointer table, and distribute the load further without the users ever knowing that this is happening, and without additional changes to the applications.

Another way of measuring our success was in the latency of certain functions. Figure 5 shows an example of one of those functions



**Figure 5. Average Duration of function.**

As you can see, there was a 500% performance improvement for this function once replication was implemented.

## Conclusion

Some of the things that made this effort successful were good coordination between the development team and the DBA team, excellent system performance metrics and long term monitoring of system resources as well as equipping the application for response-time measurements. It is very important to define metrics early in the process so that you can measure your degree of success.

By implementing transactional replication to off-load reporting functions, it is possible to reduce the CPU utilization on the main server while new users are continually added. In addition, we have developed a scalable, robust solution that will last for years to come.

---

## Bio

Edward Whalen, CEO and founder of Performance Tuning Corporation. Edward Whalen has over 10 years experience in solving database performance problems. His company, Performance Tuning, specializes in solving tough performance problems. Edward is also the author of four SQL Server books for Microsoft Press and two Oracle books. Edward Whalen can be emailed at [ewhalen@perftuning.com](mailto:ewhalen@perftuning.com).

Geoff Langos, Manager of Information Technology, Verizon.

Alexander Stamenkovich, Manager of Database Operations at Verizon.

Alexander Stamenkovich has been working with SQL Server for 5 years and manages over 50 SQL Servers.