Demonstrating a Scalable STP Solution

perspectives

PERSPECTIVES

helps business and information systems executives address critical management and technology issues.

Table of Contents

About the Author

Philip Czachorowski is a partner in CSC's Consulting Group with responsibility for its Systems Performance Center. He brings deep expertise in core technologies to industry-focused solutions such as STP.

Executive Summary

The industry initiative to implement straight through processing (STP) is being driven by the need to reduce cost as much as by the move to T+1. Financial services institutions (FSIs) are facing four technology inhibitors as they consider STP: connectivity, multiple technology integration, high-latency batch processing, and the need for data aggregation and distribution. Global straight through processing (GSTP) provides an opportunity to fundamentally change the operating models for each industry participant. Asset managers can eliminate today's dependencies and achieve significant cost savings. Broker/ dealers can drive down infrastructure costs and ensure their survival. Custodians can protect their franchise by achieving competitive advantage through advanced technology and global service offerings.

CSC built a prototype of a GSTP concentrator to demonstrate how STP solutions can be built with message queuing and message broker technology using IBM's MQSeries® and MQSeries® Integrator (MQSI). A realistic simulation of a GSTP workload was implemented and tested under high message rates to gain an in-depth understanding of the STP design and performance issues.

Our key findings are:

- Message queuing and brokering products, like MQSeries and MQSI, make it technically feasible to integrate a wide variety of STP applications, both internal and external. These products provide the core messaging and integration services that are key to implementing STP solutions.
- Our results show that, with the appropriate design, MQSeries and MQSI can support the largest FSIs and their trading volumes. An impressive throughput rate of over 300 GSTP messages per second with persistence was attained on an IBM RS/6000 S80 using a single MQSI broker. This volume corresponds to about ten trades per second. Based on the test findings, cost effective design strategies were developed for horizontally scaling solutions to virtually any message volume.
- Simplifying the implementation of STP solutions is critical toward meeting the business goals of reducing costs, improving business flexibility, and increasing responsiveness to change. The prototype demonstrated how to architect and design a STP solution that meets these goals, while still building a robust system. For example, a dynamic message routing technique was implemented, using XML, which makes connecting systems flexible and easy.
- STP solutions, with their demanding processing loads and complex interactions between applications and technologies, pose significant design risks -- especially around performance and scalability. FSIs should use rigorous techniques, like benchmarking and predictive modeling, to help them architect systems and choose the appropriate design options. This evaluation confirmed CSC's belief in using a performance-based approach for designing and implementing high-volume systems.

Our GSTP testing demonstrates how a scalable STP solution can be built in a timely and cost-effective manner. Using proven message queuing products along with the appropriate design approaches, like dynamic routing, can facilitate the integration of disparate internal and external applications while minimizing risk. Designing and implementing a flexible architecture can reduce the deployment, testing, and maintenance costs. Innovative ways to maintain state data and the feasibility of using transactional integrity with high-volume workloads were also demonstrated.

GSTP*, sponsored by the Global Straight Through Processing Association, is an industry-wide initiative to enhance and speed up the post-trade, pre-settlement processing of cross border transactions. Brokers/dealers, asset managers, and custodians send trade information to a central matching engine, the Transaction Flow Manager (TFM). The GSTP architecture allows participants to connect to regional service providers called concentrators, who aggregate connections so that each participant does not have to connect directly to the TFM. Concentrators may provide added services to the participants, such as the ability to translate messages and calculate fees.*

Background

Meeting the STP Challenge

The global financial services industry finds itself operating in one of the most complex marketplace environments in its history. In addition to new and unexpected challenges, financial services executives are still wrestling with the numerous formidable tasks they faced prior to September 11th.

Across the industry, firms are reporting significantly reduced earnings and preparing for staff reductions, even as they struggle to enact tactical solutions to rebuild facilities and infrastructure. The environment is such that the SIA has delayed the implementation date for T+1 to 2005. However, other industry initiatives to reduce risk and trade failure rates, like GSTP, continue to move forward. Realizing the potential cost reduction benefits of STP, FSIs are beginning to aggressively leverage automation to address a core business problem pervasive in the industry today: the unprofitable nature of low-margin, mature product lines operating on inefficient and costly infrastructures.

The Global Straight Through Processing Association (GSTPA) is leading the charge to streamline cross-border trade processing. The challenge is to reduce trade failure rates that, in turn, reduce daily operating risk and associated costs. One of the chief aims of the GSTP solution is to accelerate the flow of information between institutional trade partners involved in cross-border transactions. The unique feature of the GSTPA approach is the notion of multilateral interconnectivity among participants involved in post-trade, pre-settlement securities processing using an industry utility that acts as a real-time matching and enrichment engine.

As FSIs evaluate the requirements necessary to participate in this new model, they must consider the effort required to make the third-party interaction happen, as well as the changes needed internally to meet the new just-in-time information flow demands. CSC research has identified four prevailing technology-related issues that act as inhibitors:

- Connectivity - connecting disparate applications that frequently use specialized formats
- Multiple Technologies - resolving redundant applications across multiple platforms, products and message protocols
- High-Latency Batch Processing - converting batch processes to process messages in near-real time
- Data Distribution/Aggregation - providing a single, consistent, logical view of data that is physically distributed across multiple platforms

Firms should solve these issues in the context of a best practice future-state operating model for the front-, middle- and back-office that embodies these characteristics:

- Core business processes must not be "hard wired" to the application infrastructure - enabling them to adapt quickly without being constrained by technology.
- STP operations should only require manual intervention to deal with exceptions that adaptive rules-based systems cannot resolve.
- Technical architecture principles should provide open-ended integration and scalability.

Clearly, messaging and workflow technologies are central to the solution. This white paper focuses on CSC's recent GSTP solution prototyping effort and the successful results achieved using message queuing and brokering technology (MQSeries and MQSI).

__MQSeries__ is a software product that provides reliable messaging services between multi-vendor platforms using asynchronous queues. It handles the underlying communications protocols, complex network interfaces, and error handling. Applications use a simple API set to put and get messages, which is common to all platforms.

__MQSI__ is a powerful integration broker that reacts to business events, using MQSeries to deliver messages. It enhances the capability of a MQSeries network by orchestrating the flow of information based on policies or business rules. It can enrich and transform data, dynamically routing it in the format required by the recipients.

Messages are processed by a message broker in message flows. A message flow is comprised of logical constructs called processing nodes. IBM provides a set of nodes, called primitives that developers may use to implement application logic. Node types include Compute, Filter, MQInput and MQOutput. Additional nodes can be added by building 'plug-ins,' many of which are available from IBM and other vendors.

Business logic is coded in processing nodes using Extended SQL, ESQL. Based on SQL, ESQL includes extensions for procedure logic and iteration. It is used to access relational tables as well as to manipulate message fields.

Message flows execute in execution groups, which run as operating system processes. Additional instances of a message flow execute as threads. MQSI supports transaction integrity and advanced features like publish and subscription services.

Building a GSTP Prototype System

CSC built a prototype of the GSTP concentrator to demonstrate how message queuing and brokering technology can be used to enable STP. The GSTP concentrator solution is particularly applicable to the STP workload as it processes trade and settlement messages and has the same requirements for messaging, data translation, data validation, transaction integrity, state management, and processing volumes.

MQSeries was used for messaging and MQSI Version 2 was used for data integration and workflow management. MQSeries makes it easy to connect to just about any system, as it is the most widely used queuing software in the industry. It supports most platforms and offers an extensive set of adapters. MQSI is a powerful message broker that acts as a message hub. It simplifies connections between applications, integrates information from databases and other sources, applies enterprise-defined business rules intelligently, directs the flow of information dynamically, and transforms data.

CSC's Systems Performance Center

CSC's Systems Performance Center evaluates and tests key products using a rigorous approach with an emphasis on performance. Architects must understand a product's performance characteristics and architectural constraints in order to make sound design decisions. This understanding can only be gained through real in-depth testing. As a result of its extensive evaluations, CSC produces best practices, design guidelines, and performance metrics. Architects use the results to build predictive models when evaluating design alternatives, and to craft optimal solutions for FSIs.

Product testing starts by implementing a real business solution, such as the GSTP concentrator. A proven, structured methodology has been developed with a set of supporting tools that are continually refined. High-volume workloads are executed to test a product's performance under realistic conditions. All performance tests are executed using standard benchmarking practices, like having a stand-alone environment, defining the system-under test, and demonstrating the repeatability of results. The entire testing process is automated, especially the collection of performance measurements -- which are stored in a database. CSC's approach ensures meaningful and accurate results.

Evaluation Objectives

The goal of the MQSI evaluation was to:

- Explore how message queuing and brokering products could be used to implement a STP solution
- Demonstrate how MQSI could be used to implement the GSTP concentrator functionality
- Learn about the MQSI performance characteristics, especially under high-volume message processing
- Evaluate and test the MQSI features

The features evaluated and tested were determined, for the most part, by what CSC needed to implement the GSTP concentrator. Most features were tested, with a few major exceptions. For example, the publish/subscribe feature was not tested, nor did we use any third-party adapters. The main focus in building the GSTP prototype was solving the major STP technical issues.

Performance Stress Testing. Probably the greatest technical challenge associated with STP is processing complex messages at high message rates in near-real time. A performance objective of processing 10 trades per second using the GSTP basic trade scenario was established. Each trade resulted in 33 messages flowing through the concentrator for the workload implemented in the test, equating to a total of 330 messages per second.

__ISO 15022__ is the new emerging International Standards Organization (ISO) message standard for the securities industry to allow real-time electronic exchange of securities transactions. This standard is notable for two reasons. First, it converges the existing standards from the two major industry standard groups, the Society for Worldwide Interbank Financial Telecommunications (SWIFT) and the Financial Information eXchange Protocol, Limited (FPL). For the first time it will link the front- and back-office operations of the securities industry. SWIFT is the industry owned co-operative that provides secure messaging services mainly for the back-office transactions and FPL owns the Financial Information eXchange (FIX) protocol, used mainly for front-office transactions.

Secondly, the standard will support eXtensible Markup Language (XML), which describes content independent of physical format or presentation. Data elements are identified by 'tags' and are contained in nested structures that conform to a set of parsing rules for open access. XML is being widely adopted as the standard protocol for exchanging information electronically. ISO 15022 XML is defining the standard tags and business rules for the securities industry that will encompass the current standards.

These objectives were selected because 10 trades per second is considered the high end for trading systems and 330 messages per second is considered a high message rate for persistent messages with MQSeries and MQSI. Our goal was to attain this processing volume using a single MQSI broker.

This prototype is an approximation of a GSTP workload selected to demonstrate high-volume message processing and the feasibility of using MQSI for STP solutions. The distribution of message types may not reflect actual production systems, as messages per trade, message sizes, and processing may vary. The basic trade process implemented is only one of many trading scenarios.

Innovative Technical Capabilities. Several innovative technical capabilities, using MQSeries and MQSI for STP processing, were demonstrated in this implementation. Areas of particular interest were formatting messages, maintaining transaction integrity, preserving state data, and routing messages. For messaging, XML was used extensively, as the industry is clearly converging on ISO 15022 XML as the standard.

Transactional integrity is extremely important for the securities industry and is mandatory in most financial systems. The ability to demonstrate the feasibility of using persistent message queues and distributed units of work for high-volume workloads was highly desirable. These features are the basic building blocks for implementing transactional integrity in a messaging system.

Keeping state is important for process control and to be able to offer enhanced services. Database processing was incorporated into the message flows by updating a state table and validating fields against reference tables.

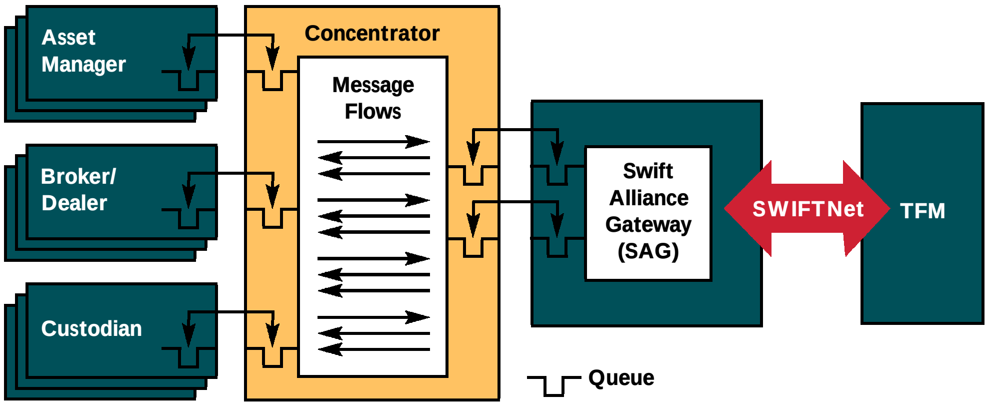
The primary role of the concentrator is to route messages between a large number of participants and the TFM. CSC wanted to demonstrate its ability to build a routing scheme that was robust and flexible but inexpensive to maintain.

GSTP Workload and Testing Approach

GSTP Workload Tested

CSC implemented a prototype of the GSTP concentrator using the basic trade message. As shown in Chart #1, the concentrator manages the message flows between the participants and the TFM. It provides communication, data translation, data enrichment, and enhanced services. The three participants -- asset managers, broker/dealers, and custodians -- send messages through the concentrator to the TFM via the Swift Alliance Gateway. Only the concentrator logic was implemented. Test drivers were used to send and receive the messages flowing from and to the participants and the TFM.

Chart #1: GSTP Logical Processing Flow



Hardware Products Used

IBM RS/6000 S80
12 CPUs (450MHz RS64 III)
6 GB memory

IBM ESS Model E20
1.7 Terabytes of total disk
96 18.2 GB disks

Software Products Used

MQSeries
Version 5.2

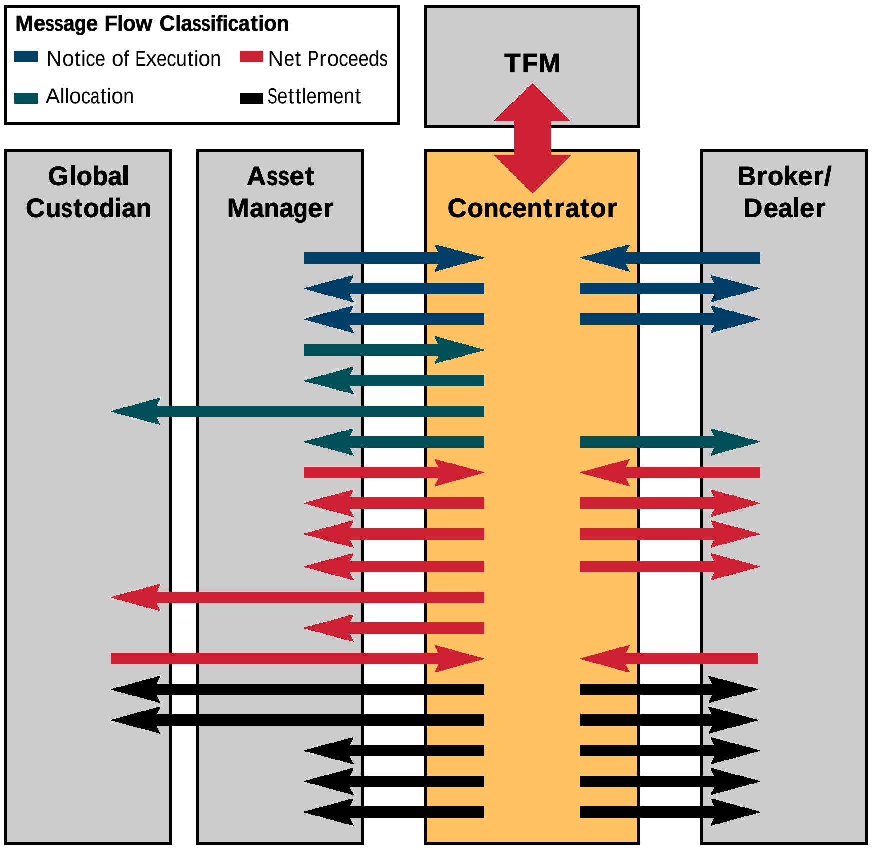
MQSI
Version 2.0.1
Version 2.0.2

DB2 for NT
Version 7.1

The basic trade process was chosen because it is most representative of the trade processing workload. The workload equivalent to 33 message types in the GSTP basic trade process was tested for the evaluation, as shown in Chart #2. Message flows are for the major trading functions of notice of execution, allocation, net proceeds, and settlement. Seven messages flow from the participants to the concentrator and 26 messages flow back to the participants.

Chart #2: GSTP Basic Trade Message

Message flow implemented for the evaluation, based on GSTP processing.



MQSI message flows were coded to process and route the GSTP basic trade messages through the concentrator. Block order notification (BON), notification of execution (NOE), and notification messages were implemented to represent the range of workloads. The message flows for the other 30 message types were cloned from the first three, based on comparable workloads.

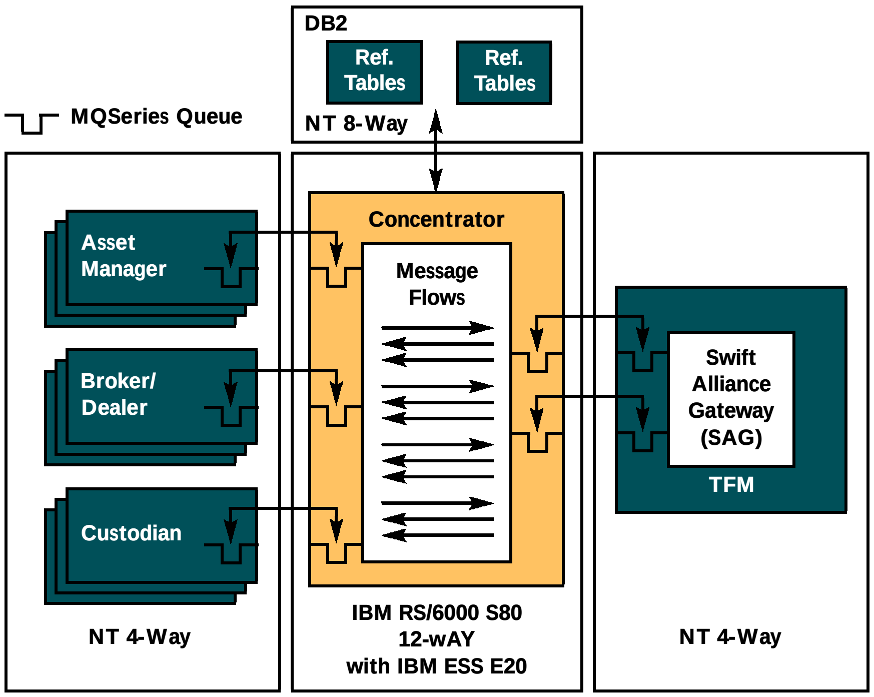
The message flow processing included field validation, data transformation, error checking, database accesses, and message routing. These message flows simulated the approximate workload and functions envisioned for the processing in the concentrator by message type. Using the GSTPA and ISO 15022 message specifications, a range of edits was implemented appropriate for the three message types. The BON and NOE message flows access five database tables. TFM notification messages access fewer database tables and validate fewer fields but do more complex message routing. All message flows insert a row to a state table to demonstrate how process management could be implemented. Message sizes ranged from 1 KB to 2 KB with approximately 20 fields per message.

Test System Configuration

The concentrator was deployed, implemented as MQSI messages flows, on a 12-way RS/6000 S80 as shown in Chart #3. The message workload was driven from two 4-way NT systems, one serving as the participants and the other as the TFM. The S80 was the system under test.

MQSeries. MQSeries was installed on all three systems for transporting messages between systems, using distributed queues. For messages flowing from the participant to the concentrator, one local queue was defined for each message type per participant type that mapped to a single queue by type on the concentrator. For messages flowing from the TFM through the concentrator to the participants, a single queue for each message type was defined that mapped to separate queues by message type and participant on the participant system enabling the testing of queue configurations for both aggregating and fanning out messages.

Chart #3: GSTP Test Configuration



Two queue managers per NT server were installed to service the participant and TFM message queues. Under actual circumstances, the participants would be spread over many servers with each server having its individual queue manager. The required message volumes were still attained with the four queue managers.

The concentrator was deployed in one MQSI broker with one queue manager. Multiple brokers could be deployed to support higher message volumes.

Test Drivers. To drive the message workload, the IBM sample program (MQSIPUT2) was modified and used to insert the messages outbound to the concentrator. A Java program, MQSIARM, was written to read the messages inbound from the concentrator. MQSIARM used the Java Message Service APIs to interface with MQSeries. These drivers included diagnostic code to collect performance statistics as well as configuration options to change the message rate. Messages were generated by a Java program using templates defined in XML. Chart #4 shows the test flow with message generation, test specification, test execution and performance data collection. All tests were defined in a database.

Both drivers were deployed on each NT system, one driver executed per message queue. Each driver was configured to generate message rates proportional to the mix of the basic trade messages by type. XML was used as the format for most of the messages with one fixed field implemented to compare XML and fixed field. Persistent message queues and transaction integrity were used in keeping with the financial nature of GSTP processing.

DB2. The application database was installed on a separate 8-way NT system using DB2 Version 7.1. Message flows accessed this database to validate fields in the message and save the state data. Each message flow execution accessed from one to five rows and inserted one row for the state data. The high message volume caused a corresponding high volume of database inserts that stressed DB2. We found that after 300 inserts per second, the insert time was approaching 250 milliseconds. Additional tuning, like using table partitioning with DB2 Extended Enterprise Edition (DB2 EEE), could alleviate this contention.

Test Execution

Tests were executed in a stand-alone system. The system was carefully reinitialized between each test run to ensure repeatable and consistent results. Tests were executed for three to five minutes, with results captured only after the system reached a steady state. The message flows were not primed, as it was discovered that they took as much as 30 seconds to start up, thus skewing the results.

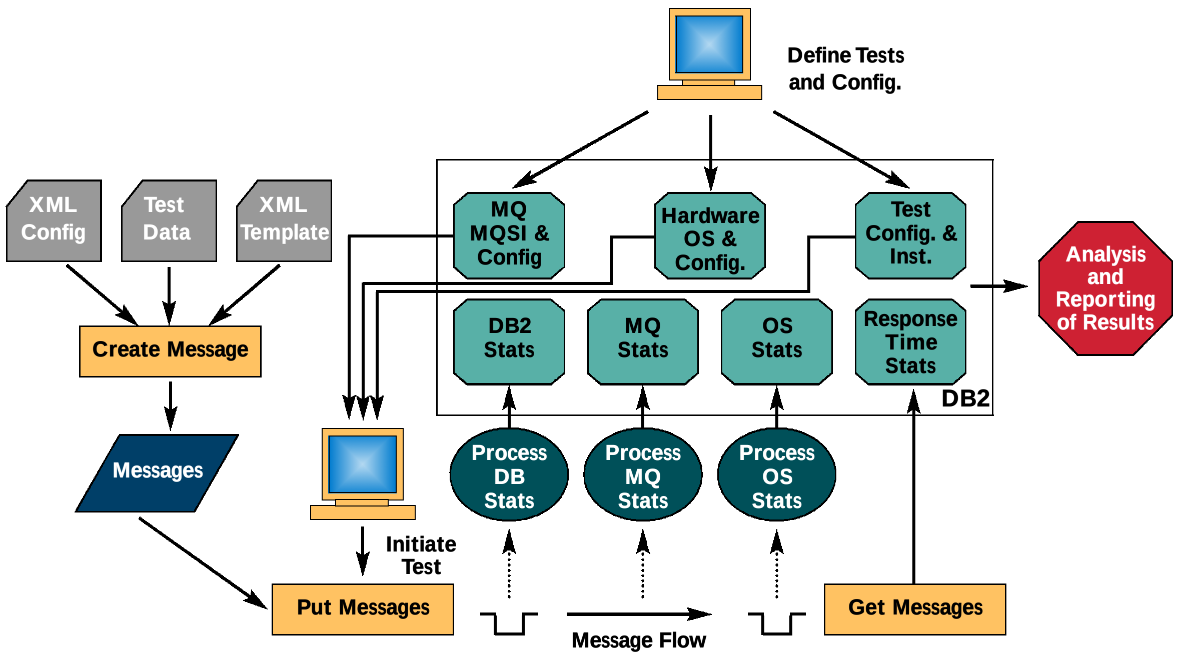
Most of the testing was done with MQSeries Version 5.2 and MQSI Version 2.0.1. Toward the end of the testing, MQSI Version 2.0.2 became available and selected tests were rerun to compare the performance between releases.

The system was extensively instrumented to capture performance data from MQSI, UNIX, DB2, and the test drivers. This data included CPU utilization, I/O rates, queue depth, response time, and DB2 statistics. All performance results were saved in a DB2 database. Each test run was assigned a unique ID that linked a specific test to its description and the performance results.

Processing times were measured for the execution of the MQSI message flows by having each compute node insert the current timestamp into the output message. MQSIARM extracted these timestamps and inserted them into the performance results database. These timestamps were used to calculate the response time for each compute node in the message flows.

Performance metrics were derived for key MQSI operations by building a set of message flows with varying workloads. Processing times were measured for individual nodes by using the IBM Execute Time plug-in. All measurements were also stored in the performance results database. These message flows were run as a single task.

Chart #4: Test Drivers and Performance Monitoring



The Results

Test results were analyzed in the context of the GSTP system, or in general, around the issues faced when implementing a STP solution. The goal was to develop practical, objective performance metrics and best practices to guide designers who would be using message queuing and brokering tools, and in particular, MQSeries and MQSI.

The results are documented under the following categories:
- Connectivity
- Systems Integration
- Batch Processing
- Transaction Integrity
- Scalability
- Performance
- Process Management
- Operations Management

Except where noted, all results were obtained in testing with MQSI Version 2.0.1.

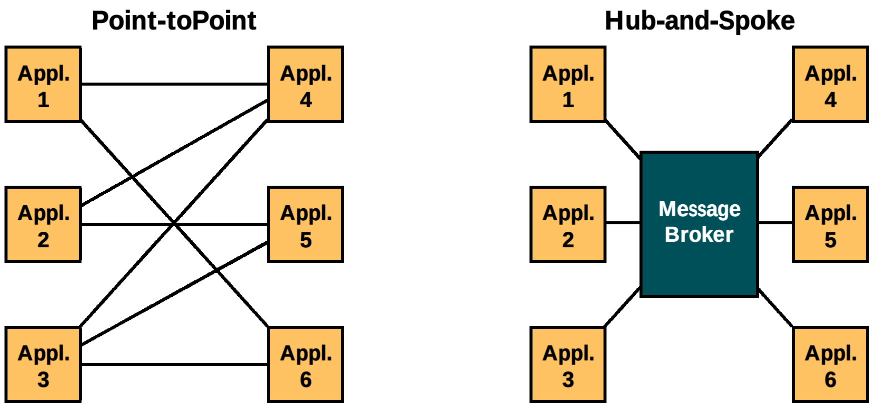
Connectivity

Historically, FSIs have built separate, independent, back-office and front-office systems aligned by product area, service offering, or application. Systems were frequently connected point-to-point, making them dependent on each other and expensive to connect and maintain. Further, these systems were often implemented using different technologies, making it particularly challenging to connect them. The GSTP prototype outlined in this paper used a combination of MQSeries and MQSI services to connect the participants and the TFM to the concentrator in a hub-and-spoke topology. This topology allows the systems to inter-operate freely while retaining system autonomy. Simplifying network configuration and maintenance was an additional objective. MQSeries provides the physical transmission services while MQSI supplies the message broker and application routing services.

Hub and Spoke Topology. The hub-and-spoke topology has the advantage that every 'spoke' can communicate with every other 'spoke' via the hub without having to build individual point-to-point links. Chart #5 shows the difference between a point-to-point network and a hub-and-spoke network. The participants and TFM were the 'spokes', while the concentrator was the 'hub'. MQSeries channels and queues were used to physically connect the participants and TFM, and to accomplish the physical message routing. The asynchronous message queuing model used by MQSeries provides integrity without tightly coupling systems within a single transaction unit of work.

Chart #5: Network Topology

Hub-and-spoke topology is advantageous in that each node only has to connect to the hub and then it can communicate to all other nodes.



MQSI is the key component in the hub topology. It serves as a message broker to transform and route messages from one system to another, eliminating any direct dependency. The actual routing is performed by the application logic built into MQSI, using the underlying MQSeries queues for the physical message transmission. Applications systems can connect to other systems without having any knowledge of the physical network or the other system. Translation and routing logic was implemented in the MQSI message flows.

*An **adapter** is a piece of code that transforms a representation of one business function to another. Adapters sit between the message broker and the application, and are usually application specific. An adapter may be used between the input application and message broker or between message broker and the output application, or both.*

*A **message broker** acts as a "hub", sitting between all the participating applications. It manages the exchange of information between applications, by routing messages, transforming formats, and applying rules. Message delivery is usually provided by messaging software through asynchronous message queues. Messaging can be bi-directional or follow a publish/subscribe protocol.*

Dynamic Message Routing. In the prototype, it was desirable to dynamically route messages between applications to minimize the need to hardcode queue names in the message flows. CSC envisions that GSTP or STP solutions could have hundreds or even thousands of queue names. A dynamic message routing scheme was employed by using a combination of a routing table and the MQSeries distribution list. The routing table contained a mapping of each participant agent ID and function to destination queue manager name and queue name. A compute node was used to look up the queue name in the routing table and place it in the destination list. A state table, which was updated each time a message was processed for a trade, was also implemented. Although the logic was not employed, this state information could have been used in the routing of messages.

The routing table keeps the routing data independent of the message flows, for easy management. Maintenance is performed by updating the routing table, which can be accomplished on demand. The only disadvantage to using the routing table is the cost of creating the table look-up. Allocating a large DB2 buffer pool allows the routing table to be kept in memory. The table used in CSC's GSTP prototype included 3,000 entries. Performance was acceptable even with the table look-ups.

A network was simulated as if there were 100 physically separate broker/dealers, 100 asset managers, and 10 custodians - even though physically all the participants were on one NT system. Two major options were explored for configuring message queues: 1) having one queue per participant for all messages, and 2) having a separate queue per participant per message.

MQSeries Large-Scale Network Configurations. MQSeries offers a number of options for configuring large-scale networks that can simplify the definition and management of the queue objects. Using the cluster feature (a grouping of queue managers) can reduce the number of objects to define and administer. Queue managers can be linked together in a hierarchy of smaller networks with gateways for passing messages through the system. With careful design, this approach can minimize the number of actual physical connections and MQSeries channels. It allows messages to be sent indirectly to queues without having to define the queue names on the local system. A complex network was not implemented but there is great potential for using MQSeries in this manner.

External Gateways. For STP environments, MQSeries and MQSI should provide an effective and flexible solution for connecting internal systems. This same approach may also be used when connecting to external systems, although other interfaces, like SWIFTNet, may also be required. Adapters are available to easily integrate MQSeries and most other protocols making for seamless communications. For example, SWIFT offers a MQSI host adapter for the SWIFT Alliance Gateway. In the CSC prototype, messages intended for the TFM were placed in a queue that --in a production system-- would have been passed to the SWIFT Alliance Gateway, via the SWIFT adapter. The TFM processing was outside the system-under-test.

System Integration

FSIs typically integrate systems by building custom code -- which is expensive to create and maintain. The key to effective system integration is translating messages between systems so that all systems can interface to all other systems. Without an integration tool, each application has to be aware of the specific interface of the connecting application and be able to translate messages between systems.

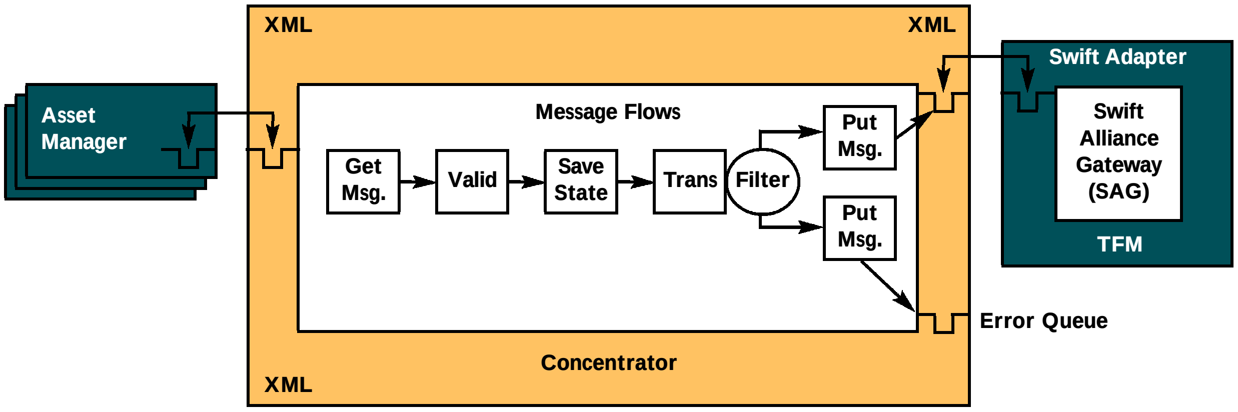
With MQSI, system integration occurs at two levels: adapters and message flows. Adapters interface between an application and MQSI, passing data back and forth without having to change the application to access MQSI directly. Adapters are usually specific to an application or technology, and encapsulate the business function. Message flows are used to transform a message from one format to another as it flows through MQSI.

Both levels of integration are often required. Adapters move information in and out of MQSI, eliminating the need for sending and receiving applications to interface with MQSeries. MQSI message flows transform and route messages.

Adapters. MQSeries and MQSI work with an extensive set of adapters. These range from adapters specific to an application, such as SAP, to adapters specific to a technology (i.e. DB2), see Chart #6.

CSC's prototype did not employ an adapter. However, in actual circumstances, participants would have acted as the interface between the local participant systems and MQSeries, to transform messages between the application-specific format and XML. On the TFM side, CSC envisioned the use of an adapter like the SWIFT Alliance Gateway MQSeries adapter to interface to the SWIFTNet. This adapter translates messages between XML and the SWIFT message format by calling the SWIFT APIs.

Chart #6: Use of Adapters to Interface with MQSI



Message Flows. Once the message is in MQSI, message flows should be used to transform messages between input and output formats. As a message broker, MQSI has a full range of data transformation and enrichment services that can be implemented through ESQL in message flows. Data being processed by message flows is kept in a canonical format, which bridges the sending and receiving formats. This architecture provides an open platform for any new system to immediately 'plug in', and use the same transformation code -- as long as the field names are the same.

This capability was demonstrated by having the same message flows process both a fixed-field message and a XML message. Fixed-field messages require that the appropriate message descriptors be defined.

XML. Generic XML was used for most of the messages implemented for the GSTP prototype, as XML is the direction set by SWIFT and FLP with the ISO 15022 XML standard. By doing this, the unnecessary work of having to define message descriptors was avoided and a set of standard field names was established.

XML will work only if all the applications agree on the field names and the data semantics. In the CSC prototype, it would be the task of the adapters to tranform the messages from the application specific format to XML.

XML can be predefined in MQSI and it is very easy to manipulate - an advantage that message formats do not enjoy. However, document type definitions (DTD) are not currently used by MQSI, requiring that the message formats be validated in the message flow. IBM will support DTDs in a future release. During this evaluation, CSC edited all fields in the message flows using compute nodes with ESQL statements.

Data Transformation. A range of transformations in message flows was tested - from simple translations of message formats to changing the semantic content of messages. Translating from one message format can usually be accomplished by just moving fields from the input to output message. In this prototype, a fixed-field message was converted to XML. More complex transformations can be made via ESQL. With ESQL, DB2 tables were accessed to add content to messages. Performance can be impacted depending on the number of fields being transformed or the amount of messages being formatted.

Data Validation. Message flows can be used to validate fields. These validations can be coded as ESQL IF statements or as filter nodes. Using SQL, database tables can be referenced to validate fields against a database.

All three approaches were tested and findings indicated that each has unique uses. For example, the IF statement is best for simple edits, as they may be contained within one compute node. The filter node should be considered when a potential edit would change the message flow. Filter nodes are limited to a binary decision, making the implementation of multiple decision paths challenging. Edit errors were stored by adding them to the output message. This approach allowed continuous editing of fields, even in the event that an error was found.

Data Enrichment. Message flows can use ESQL to add information or to 'enrich' the data. An example of data enrichment is replacing a code, such as a product code, with the full name. Values may be hard coded within the ESQL or retrieved from a database. Both options were tested.

Business Logic. Beyond transforming data, almost any business logic can be implemented in ESQL. Nearly any processing can be implemented with the IBM supplied primitives. Although ESQL is fairly flexible, large sets of code can be difficult to follow and should be structured. MQSI has limits on the number of EQSL lines possible in a message flow based on the repository column size definition.

Reusable Code. In coding message flows, findings indicated that some functions were common to many flows. For example, many fields were processed by multiple message flows, necessitating coding the same edits for each flow.

CSC recommends using sub-flows as a means to reuse code. A sub-flow is a self-contained grouping of message nodes that can be linked into any number of message flows as an entity. Thus, a function only has to be coded once for reusability. Using sub-flows increases the number of compute nodes, which can negatively impact performance.

Sub-flows should be used as much as practical to simplify the coding effort and minimize maintenance. Maximizing the use of sub-flows requires that message flows be designed before they are coded. Designers must be aware of the performance impact of increasing the number of compute nodes and design sub-flows accordingly. Having too many compute nodes can severely impact performance.

Changing the ESQL in a sub-flow requires that all message flows be re-deployed to enable the change.

Error Handling. MQSI provides a range of options for trapping and handling errors, including: using the filter node to redirect the message flow; and using the node connectors. Before coding message flows, CSC designed a common approach for handling errors. Integral to this approach was establishing a set of error queues where all error messages would be routed, depending on the type of error.

In addition to these options, MQSI offers a way to throw exceptions and provides a trace facility. The trace facility was used extensively to debug ESQL errors.

Message Flow Architecture. Message flows can get fairly complex with a large number of processing nodes, as well as lengthy ESQL procedures. Just like any application system, designers should establish a message flow architecture to guide the development and ensure a structured, efficient design. Performance will usually be a major factor in message flow architecture and design.

Elements of the architecture include:

- Message flow structure
- Editing techniques
- Node usage guidelines
- Code reusability
- ESQL coding guidelines
- Message queue names
- Database interfaces
- Error processing

Batch Processing

One of the greatest challenges in implementing STP is the ability to interface with legacy batch systems in the back-office. These systems will have to be changed to process messages in near-real time and to commit work at the transaction level. Since rewriting all of the systems at one time is usually unfeasible, IT professionals will have to find practical ways to modify these systems to benefit from STP.

Although custom solutions will have to be crafted for each specific batch system, there are some general approaches that will serve as a starting point. CSC did not test these with our prototype.

The approach for converting a batch program to near-real time processing will vary depending on whether the batch program is reading or writing messages. For programs that read messages from a file, the program will have to be vastly changed to be interactive and immediately process messages as they arrive. Arrival rates will be random. One approach is to implement a message driver that waits on the input queue and can immediately read a message when received. Programs will have to be modularized and callable so that the message driver can call the appropriate code and process the message as an independent unit of work. Multiple message drivers may be needed to ensure near-real time processing depending on the message arrival rate.

For programs that write messages to a file, one approach may be simply to replace all the write calls with calls to the messaging queuing system. These calls may have to be achieved as a separate MQSeries unit of work to ensure that the message is immediately transmitted.

In any of these approaches, MQSI message flows or an adapter could be used to transform data formats.

Transactional Integrity

Due to the financial nature of the STP applications, the processing of critical messages requires transactional integrity. All the GSTP messages were processed as transactions to demonstrate the feasibility of using transaction integrity for high message volumes.

MQSI uses underlying MQSeries services to provide transactional integrity and, in a traditional sense, is acting as a transaction manager. The scope of a transaction is the message flow-- from receiving the input message to writing the output message. Transactional integrity requires that all message queues be defined as persistent. The failure of a message flow results in having all the work rolled back and the input message remaining on the queue.

The greatest impact of executing messages as transactions is the overhead incurred by having persistent queues. Throughput numbers show that, with the appropriate tuning of MQSeries and the message queues, high message rates are still practical although will be limited by the I/O rate of the persistent queue.

XA. Extending the unit of work to include an external DBMS requires the X/Open XA interface, in addition to transactional integrity and persistent queues. The XA interface is an industry standard that allows multiple resource managers, like a DBMS, to coordinate update processing in a single unit of work using the two-phase commit protocol. This ensures that all changes either all fail or all complete.

For MQSI, database changes will be under the same unit of work as the message queue changes, with MQSI acting as the unit of work coordinator. If either the message flow or database fails, both the message queue and the database changes would be backed out together. XA would be used when the message queue content has to be coordinated with database changes.

XA was tested using DB2 as the DBMS. XA had a significant impact on throughput, cutting messaging rates by up to 40% in some tests. From an operational perspective, XA introduces more complexity, which makes it more difficult to manage.

Use of Transactional and XA Options. CSC recommends using MQSI transactional integrity only if needed as it does increase processing time. With the appropriate tuning, the overhead of using transactional integrity is manageable, as demonstrated by attaining the target GSTP workload of 330 messages per second. Using transactional integrity requires the use of persistent queues.

Using XA adds significantly to the overhead of the MQSI transaction processing and its use should be limited to situations where there are no other design alternatives and where performance is acceptable. Not only does the use of XA significantly reduce throughput, it also greatly complicates system administration and management. The performance impact of using XA is covered in the next section.

Scalability

Given the volatility in trade volumes, STP systems will have to scale to provide consistent response times, even as transaction volumes abruptly increase. Although they execute in near-real time, these systems still require a guaranteed level of service in regard to response time.

A comprehensive set of tests was executed to understand the scalability of MQSI in preparation for the 330 message per second GSTP workload. These tests benchmarked the performance characteristics of MQSeries and MQSI as options were tuned and application code was changed. Scalability was assessed by measuring the change in throughput and response time as the message rates were increased. All tests were conduced using a single MQSI broker. With the appropriate message design, additional brokers could be deployed to horizontally scale a workload.

These results must be understood in the context of the workloads tested, as factors like message size, database accesses, and message flow logic will affect performance and throughput.

Tuning MQSeries. The first hurdle in driving high message rates through MQSI is tuning MQSeries so the queue manager can deliver the desired message rate. CSC focused its efforts on tuning MQSeries using persistent queues. The tuning effort for non-persistent messages is fairly straightforward.

Using persistent message queues complicates the MQSeries tuning effort due to the fact that the queue, queue manager, logging parameters, and application queue processing must all be configured appropriately for the expected message rate. Using MQSeries V5.1 was crucial to increasing the persistent logging rate as IBM increased the log buffer size and implemented a kind of 'group commit'.

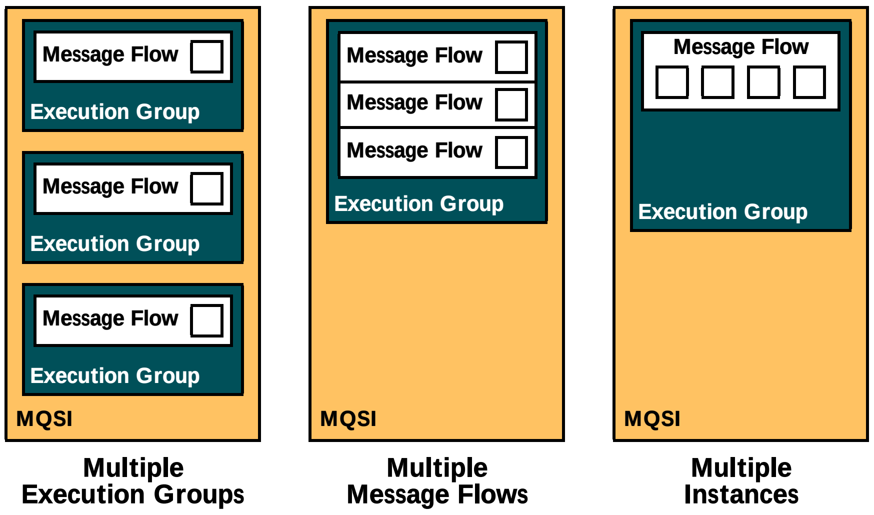
From an application perspective, the MQSeries connect, open, put, and commit options should be optimized. In particular, multiple puts should be processed per open and connect. Message size affects maximum message rates, especially as messages exceed 4KB. The messages tested in the GSTP prototype tended to be smaller than 4KB and there was no need to optimize for message size. The log file can be a huge bottleneck, constrained by the physical I/O rate. Findings indicated that IBM's Enterprise Storage Server (ESS), with its large cache, can substantially increase I/O rates. The message batch size was carefully matched with the application insertion rate to optimize processing per MQSeries synchronization point.

Tuning the message queues and the MQSeries is an iterative approach because of the interaction of parameters and the dynamic nature of a queued messaging system. MQSeries and MQSI also affect each other, sometimes in unexpected ways. Message queues must be monitored in order to locate possible constraints.

MQSI Message Flow Execution. MQSI offers three major configuration options for executing message flows: execution groups, message flows, and multiple instances. Chart #7 shows how these execute from an operating system perspective. Each of these options scales differently, and each has a set of advantages and disadvantages. CSC spent considerable time testing these options to understand how to configure MQSI and MQSeries for optimal response time and throughput depending on workload. The IBM 12-way S80 allowed for extensive scalability testing.

Chart #7: MQSI Execution Architecture

Execution groups are implemented as operating system processes. Message flows and instances execute as threads.

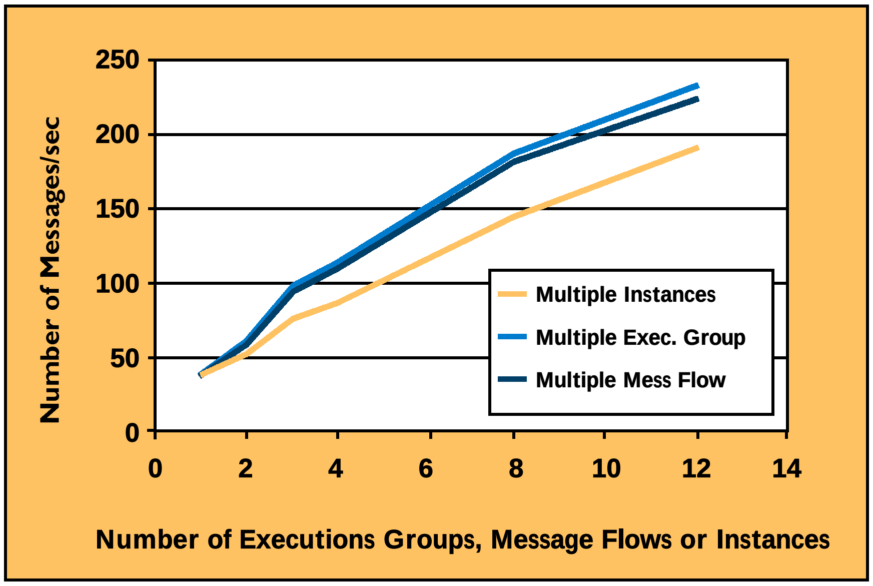


Execution groups scaled best, with message flows second, and multiple instances last. The difference in scalability reflects the dispatching efficiencies and contention within MQSI and the operating system resource usage. Execution groups are implemented as operating system processes-- require more memory while instances are implemented as operating system threads and use less memory but more CPU.

Although multiple instances performed the worst of the three options, it has the advantage of being the easiest to deploy and manage. Chart #8 summarizes the results. In general, CSC recommends using multiple instances, with its ease of deployment, because testing indicated that the throughput rates were reasonable up to 20 or 30 instances. If performance is an issue, multiple execution groups are recommended.

Chart #8: Comparison of the Scalability of Execution Groups, Message Flows, and Instances

Execution groups scale best but are more difficult to manage. Multiple instances are preferable unless performance is an issue. Scalability depends on the message flow workload (tests were executed on a single MQSI broker).



Throughput can be increased by adding more execution groups or instances, as long as resources are not constrained. In CPU-intensive message flows, the processors will be the constraint, limiting the scalability. In message flows with external calls, like to a database, the constraint may shift to the external system. Because these message flows are not as CPU-bound, they may be able to scale to higher workloads as long as the external resource does not become a constraint.

Message flow workload has a major impact on message throughput. ESQL is an interpretive language and can be CPU-intensive depending on the number of statements, fields, and nodes. Architects must consider the impact of CPU and external resource usage on throughput rates and scalability. Message flows that will have high message rates must be carefully designed to minimize CPU usage.

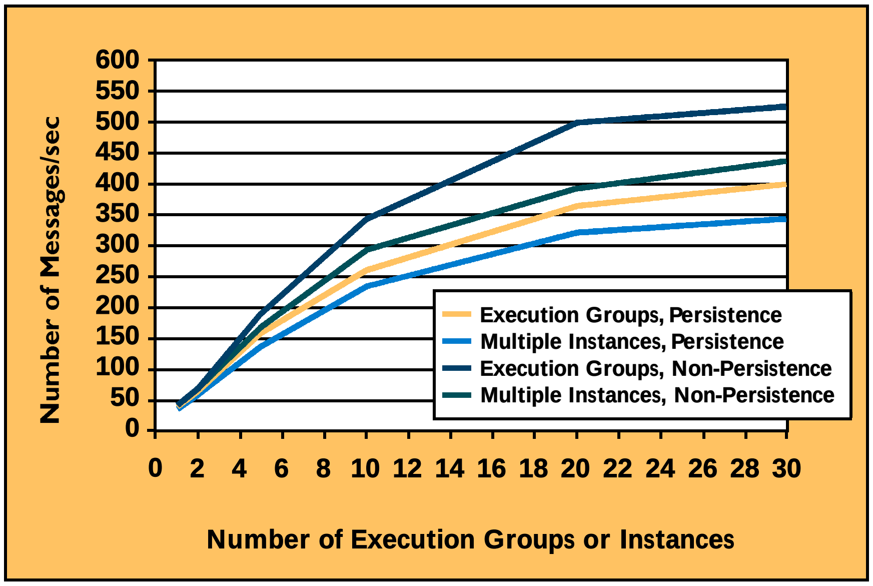
Impact of Persistent Message Queues. Persistent message queues add signifcant overhead to MQSeries because each message has to be physically logged as part of the unit of work. The message rate for inserting messages is dependent on the physical I/O rate that messages can be logged. Considerable tuning of MQSeries was needed to optimize this rate. The IBM ESS disk with its immediate write feature was paramount to this process.

Chart #9 compares the message throughput with and without persistent queues, using a modified message NOE message flow and one queue manager. The state table insert was removed to reduce database contention.

With 30 execution groups almost 525 messages a second were achieved compared to about 425 messages a second with 30 instances. The comparable message rates with persistent queues were about 400 messages per second with 30 execution groups and about 350 messages per second with multiple instances.

Chart #9: Comparison of the Scalability using Persistence

The use of persistent queues decreased throughput rates for both execution groups and instances. With all four tests, throughput rates increased up to 30 threads. Scalability depends on the message flow workload (tests were executed on a single MQSI broker).



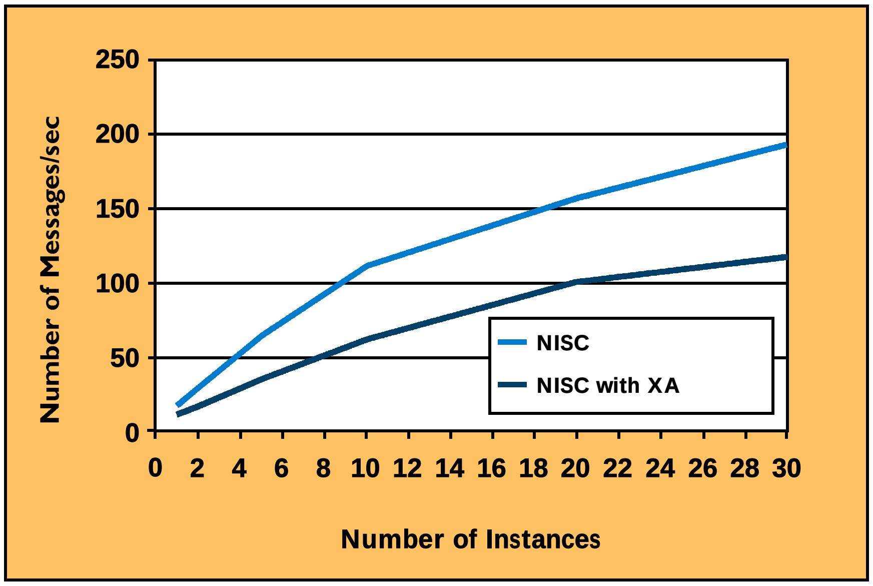
Throughputs can be scaled beyond the logging constraint by adding another broker and a queue manager. Since each queue manager has its own physical log, the I/O rate is effectively doubled.

These impressive message rates demonstrate that the processing capacity of MQSI is suitable for the GSTP workload and STP-like applications. Even with persistent queues, a rate of over 350 messages per second was attained. High message rates, such as these, depend on the message flow workload and require careful tuning of both MQSeries and MQSI.

Impact of XA. The use of XA decreased message throughput by 35% to 40%, as illustrated in Chart #10. The peak message rates with XA were 123 messages per second at 30 instances compared to 191 without XA. Still, attaining a processing rate of 123 messages per second is impressive and would support many applications. These tests were run using the NOE workload. The message flow inserts a row in the state table, which caused some contention and a flattening of the scalability. The average CPU utilization on the 12-way S80 was 35% for 30 instances.

Chart #10: Comparison of Scalability using XA

XA significantly reduced throughput by up to 40% with the NOE Message Flow (tests were executed on a single MQSI broker).



Achieving the Target GSTP Message Volume. CSC achieved the targeted processing volume of 10 trades per second with the GSTP workload, which was approximately 310 messages/second. This achievement demonstrates the suitability of using MQSI for GSTP and STP. The message volume was constrained by database contention on the insert processing of the state table. As the S80 was only about 50% utilized, a higher message rate could have been attained by easing the insert contention. Based on scalability testing, CSC would expect that rates in excess of 400 messages per second would be possible before having to deal with constraints in MQSeries and persistent logging.

Table #1 summarizes the results by queue name. The participant-side message flows are more process intensive, with more complex validations and database selects. The TFM-side message flows do less processing and database selects but use the routing table to achieve dynamic message routing. Queues for all messages are persistent and all message flows update the state table.

Each message type was placed in its own execution group with 20 thread instances defined per group. A total of 13 execution groups and 260 thread instances were used. The high number of instances was required due to the elongated elapsed times caused by the database insert contention. Without the database insert contention, the GSTP workload could have been processed easily by 30 message flows as demonstrated by Chart #9. The high number of inserts was causing a wait time of almost 250 milliseconds per insert.

Based on scalability testing, CSC found that 20 thread instances per execution group yielded reasonable performance while minimizing the number of execution groups to deploy and manage.

Table #1: Summary of the High-Volume GSTP Test

Over 300 messages a second with persistence were achieved with an average elapsed time less than .250 seconds. This equates to about 10 GSTP trades per second (tests were executed on a single MQSI broker).

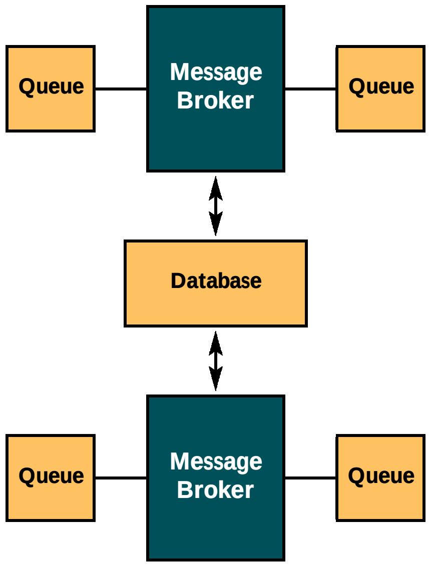
Message Queue	Number of Message Types	Exec. Groups	Message Flows	Instances	Test Drivers	Target Message Rate per Second	Actual Message Rate per Second	Average Process Time
BON	1	1	1	20	1	10	9.93	0.230
NOE	1	1	1	20	1	10	9.92	0.248
ALLOC	1	1	1	20	1	10	8.98	0.222
NP	1	1	1	20	1	20	17.69	**0.209**
SetDet	1		1	20	1	20	17.76	**0.245**
Part Total	5	5	5	100	5	70	64.27	
Success	5	2	2	**40**	7	70	75.29	**0.225**
TMS	1	1	1	**20**	2	20	19.92	**0.233**
All_Not	1	1	1	**20**	2	20	20.05	**0.228**
Stat_Chg	5	2	2	**40**	5	100	78.49	**0.167**
NP MS	1	1	1	**20**	3	30	30.20	**0.224**
SDS	1	1	1	**20**	2	20	20.25	**0.224**
TFM Total	14	8	8	**160**	21	260	244.20	
Total	19	13	13	**260**	26	330	308.47	

Scaling Beyond One MQSI Broker. MQSI workloads can be scaled over more than one MQSI broker if designed appropriately, see Chart #11. CSC designed the GSTP prototype to be scale horizontally by storing state data in a DB2 database. This architecture made the state data independent of MQSI, enabling the message flows to be spread over multiple MQSI brokers, each accessing the common state table. Each broker could be deployed on its own server, if needed, given that they all connect to the same state table. Additional brokers could be added providing that the DB2 database was not constrained. If the database became a constraint, it could be moved to a larger server or partitioned using DB2 EEE.

A collection of brokers could be deployed as a MQSeries cluster to have MQSeries manage the workload distribution. Spreading a message workload over a cluster provides some degree of redundancy and fail-over capability, which would be recommended for a financial application.

Chart #11: Deploying Multiple MQSI Message Brokers

Workloads can be scaled over multiple MQSI brokers, each accessing the same database for state information. Brokers may be installed on multiple computers.



Scaling Improvement with MQSI Version 2.0.2. Both MQSI Version 2.0.1 and Version 2.0.2 were tested. In Version 2.0.2, IBM made significant performance enhancements, especially in message parsing and node processing. The throughput of the modified NOE workload increased by 42% for one instance thread to 32% for two threads, and only 8% for 30 threads. See Chart #12 for a graph and Table #2 for percentage improvement in throughput. The improved throughput decreases with volume because of the inefficiencies of having so many threads.

Chart #12: Comparison of MQSI V2.0.1 and V2.0.2 Throughput Rates

Performance was significantly improved in MQSI V2.0.2 resulting in a modest increase in the throughput rates, depending on the message flow workload (tests were executed on a single MQSI broker).

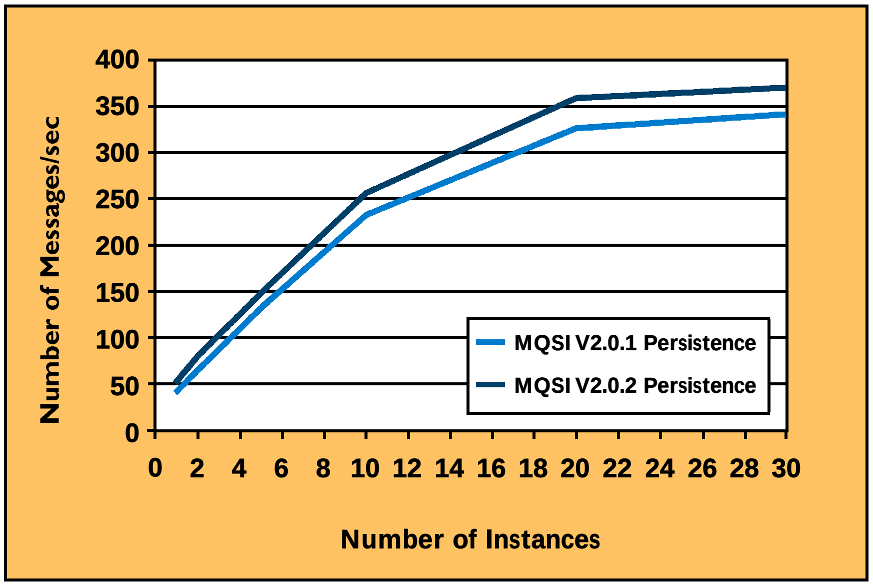


Table #2: Percentage Throughput Improvement between MQSI V2.0.1 and V2.0.2.

The percent of throughput improvement decreases as the number of instances increases because of contention between instances. All messages were executed with persistent queues.

Instances	Msgs./second with V2.0.1	Msgs./second with V2.0.2	Percent Improvement
1	32.88	46.90	42.66%
2	59.00	77.92	32.06%
5	131.73	152.43	15.71%
10	229.39	258.76	12.80%
20	321.50	360.03	11.98%
30	344.75	372.63	8.09%

Even more significant is the decrease in CPU time as measured in average CPU utilization over the running of the test. CPU time decreased by an average of 18%. See Table #3 for the percentage of CPU improvement for each test.

Table #3: Comparison of CPU Utilization between MQSI V2.0.1 and V2.0.2.

Average CPU utilization is significantly reduced with V2.0.2 showing that although throughput increases are modest, V2.0.2 uses less CPU.

Instances	CPU Utilization with V2.0.1	CPU Utilization with V2.0.2	Percent Improvement
1	6%	5%	-16.67%
2	11%	9%	-18.18%
5	22%	18%	-18.18%
10	42%	35%	-16.67%
20	69%	57%	-17.39%
30	76%	58%	-23.68%

Performance

The GSTP message flows were designed to perform in near-real time, which is a key requirement of GSTP. STP message processing can be challenging due to the complexity of format transformations, data validations, data enrichment, message routing, and database accesses.

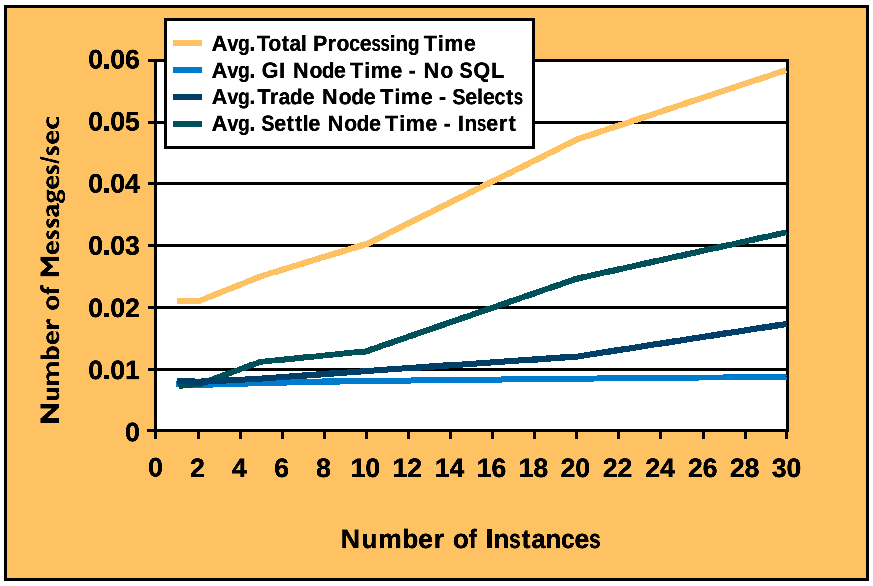
Response time is comprised of CPU and wait time. In MQSI, CPU time is most impacted by ESQL, message size, and the number of message nodes. Although a powerful language, ESQL can be CPU-intensive as it is interpreted. Message size affects CPU because of the cost of parsing and navigating the internal data structures. This cost is dependent on the number of fields and nested levels. Messages with 'deeper' structures are more efficiently accessed than those with 'wider' structures. Since messages are only parsed once per message flow, it is most efficient to have more nodes per message flow.

Synchronous calls to external resources, like a DB2 database, I/O or the message queue causes a wait in MQSI processing. MQSI minimizes the database processing time by enabling connection pooling and reusing DB2 plans. In the GSTP workload, the database itself represented the most significant challenge because of the heavy insert contention, which significantly elongated elapsed times. Under full load, average elapsed time for all the GSTP message flows was about .250 milliseconds, still acceptable for near-real time processing, but 10 times longer than the average for a single instance.

Chart #13 shows an example of the elapsed times by compute node, showing the impact of the database processing. The average elapsed time for the general information (GI) node, which does not access the database, is fairly constant with the processing rates and number of instances. The elapsed time for the trade node, which does five selects, increases more. The settlement node, which inserts a row, more than doubles as the message volume increases, because of the DB2 insert contention. Total processing time is measured from the beginning of the message flow processing to the last compute node.

Chart #13: Elapsed Time of Compute Nodes

NOE message flow with one execution group and 1 to 30 instances. The peak message rate was 191 messages per second.



Design for Performance. Architects can control message flow processing time by careful design and by managing the workload. Some ESQL features are more CPU-intensive than others. In this testing of the GSTP workload, CSC identified a set of best practices for designing message flows. Message flow structure can affect performance, especially the number of nodes, ESQL statements, and message fields. A series of simple message flows were executed to measure CPU by ESQL function and derive performance metrics. These metrics can help designers estimate performance and make design decisions.

Performance Metrics. Performance metrics for key MQSI operations were derived by structuring series of tests with varying workloads. Designers can use these metrics to predict performance when making design choices. Examples of performance metrics are the message flow initiation cost and filter node execution cost.

MQSI V2.0.2 Performance Improvements. Performance improvements between MQSI Version 2.0.1 and Version 2.0.2 were compared for a range of ESQL processing. IBM significantly improved the performance in Version 2.0.2, especially for the message parsing and node processing.

To measure the processing time difference, 19 message flows were timed under V2.0.1 and V2.0.2. These message flows were designed to isolate the processing costs of specific ESQL functions. CPU time deceased an average of 34%. The improvement in lapsed time will be less because CPU time is just a small portion of the total lapsed time.

Process Management

As STP messages flow through the various financial systems, state must be managed in order to direct and administer the process flow. Having so many autonomous and physically separate systems makes implementing a process management system technically challenging. A balance must be achieved to provide a reasonable amount of process management while not overly complicating the processing.

Some basic process management was implemented in CSC's GSTP prototype to demonstrate how to track trade status and direct the process flow. Although it is not as robust as a full workflow manager, MQSI does offer some process management services that were adequate for testing purposes.

The heart of CSC's process management is the state database and the routing code in the message flows. Managing workflow over a distributed system can be very complex and requires that workflow software be installed on all systems. CSC's solution is simple, as it is based only on the concentrator, through which all messages must pass. Storing the state data in a DB2 database provided easy access to the information through SQL.

The advantage of using MQSI for workflow is that although the capability is modest, it supports high-volume message rates. Using MQSI message flows, this workflow framework could easily be enhanced to offer more functions. For example, MQSeries trigger queues could be used to start 'events' and the publish/subscribe mechanism used to broadcast messages based on interest. Those applications needing more workflow functions should consider a workflow manager product.

Operations Management

STP applications must be highly available, reliable and secure. While providing these requirements at the individual system level is standard in today's financial services industry, implementing these features across a production, distributed environment will be more challenging.

Reliability. CSC found MQSI to be extremely reliable and available. The code was consistent, as we encountered no major product problems during the evaluation. Reliability was tested by executing a steady state of 5 messages per second through the concentrator for up to 12 hours at a time.

Operational Requirements. The message queuing model of MQSeries insulates applications from local system failures. But operating and managing a large number of distributed applications requires adequate monitoring, operational procedures, and management tools.

Near-real time processing requirements of STP require that problems be immediately detected and rapidly resolved. All operational aspects of system management will have to be extended to support these distributed systems, from normal operations to disaster recovery. Appropriate tools and procedures will be required, each of which will have to be tested and proven for production environments. MQSeries and MQSI will play a major role in this environment.

MQSI provides a central control point with the control center. Through this component, multiple MQSI brokers can be controlled from a single point. CSC found it to be robust with indicators for the current status of deployed objects. The control center is also used to deploy execution groups and message flows to brokers, although CSC found this process and the management functions to be cumbersome. IBM is enhancing the control center to make it easier to use.

Security. Security in MQSI is dependent mostly on the underlying operating system and MQSeries services. These features are limited and applications may have to develop their own security services if they have more stringent security requirements. For example, message encryption requires a MQSeries channel exit program.

Backup and Recovery. Backup and recovery procedures must encompass all resources used by the application, in particular MQSeries message queues and any databases updated. Appropriate procedures should be built using the native backup and recovery facilities for each product. The procedures must be coordinated across all resources to ensure a point of consistency within the semantics of the application. Disaster recovery requirements must also be addressed.

Conclusions

CSC successfully built and tested a prototype of the GSTP concentrator to demonstrate how message queuing and brokering tools can be used to implement STP solutions. These tools are the key enabling technology for STP solutions, providing message queuing, data transformation, data enrichment, and data routing services. STP will be critical to help FSIs reduce operating costs and improve back-office processing.

Message rates equaling those CSC would expect to reach in the single largest production systems were attained in testing. This volume was achieved with transaction integrity.

Key findings include:

- CSC's GSTP prototype demonstrates that message queuing and brokering products, like MQSeries and MQSI, can be used together to implement STP like solutions and support high message volumes between internal and external systems.
- A message broker with a "hub and spoke" topology provides an efficient network, with application independence and high flexibility when integrating a large number of applications.
- High message rates using persistent messaging and transactional integrity, in excess of 300 messages per second, are practical on a single message broker using MQSeries and MQSI with the appropriate design, tuning, and hardware resources.
- Using persistent queues can be the gating factor on message throughput using a single MQSI message broker.
- MQSI performance and scalability depends on the message flow workload. The number of message flow nodes, ESQL statements, and message fields will impact performance and must be appropriately designed. Accessing external resources, like a database, can also have a major impact especially if there are resource constraints or contention.
- Building messaging systems with high-volume message flows requires careful design and tuning of the underlying messaging software and the message flows, like MQSeries and MQSI.
- Tuning messaging queuing systems can be particularly difficult given the nature of queuing systems and the unpredictable performance behavior - especially when multiple, autonomous systems interact.
- Scalability in message queuing and brokering can be extended by building systems with 'horizontal' architectures and using multiple MQSI brokers. For maximum processing power, each broker can be deployed on a separate computer.
- Achieving STP in a legacy environment poses the challenge of converting batch systems to process messages in near-real time.
- XML, as the emerging communications standard, will facilitate the integration of internal and external systems for STP.
- The X-Open XA interface, while slow, is practical to use sparingly when messaging queuing must be in the same unit of work as the database processing.
- An effective workflow management scheme can be implemented using a message broker that, while not being as robust as a workflow product, may be adequate and more straightforward.
- Administering and operating a distributed set of loosely coupled applications will require the appropriate tools and procedures, especially to maintain high-availability and near-real time processing.
- MQSeries Version 5.2 and MQSI Version 2.0.2 should be used, as IBM has made major performance enhancements.

References

Axion4gstp Ltd, www.axion4.com

Global Straight Through Processing Association, www.gstpa.org

International Standards Organization 150222, www.iso15022.org

GSTP Executive Summary and Overview, Axion4GSTP, Version 1, July 31, 2000

GSTP TFM Message Manual: General Volume, Axion4GSTP, Version 1.3, December 8, 2000

GSTP TFM Message Manual: Broker/Dealer Messages, Axion4GSTP, Version 1.2, December 8, 2000

GSTP TFM Message Manual: Investment Manager Messages, Axion4GSTP, Version 1.1, December 8, 2000

IA0A: MQSeries Integrator Execute Time Plug-In, Neil Kolban, IBM Dallas Systems Center, March 9, 2001

MQSeries Integrator Version 2.0 Technical White Paper, IBM

MQSeries Integrator Administration Guide Version, 2.0.1, IBM, November 2000

MQSeries Integrator Introduction and Planning, Version 2.0.1, IBM, August 2000

MQSeries Integrator Administration Guide, Version 2.0.1, IBM, November 2000

MQSeries Integrator Programming Guide, Version 2.0.1, IBM, August 2000

MQSeries Integrator Using the Control Center, Version 2.0.1, IBM, November, 2000

MQSeries Integrator for AIX V2 Performance Report Version 1, Tim Dunn and Dan Jones, IBM UK Laboratories, September 8, 2000

MQSeries Integrator for AIX V2 Performance Report Version 1.1, Tim Dunn and Dan Jones, IBM UK Laboratories, June 15, 2001

MQSeries Integrator for AIX V2 Designing Message Flows for Performance, Dan Jones, IBM UK Laboratories, May 8, 2001

MQSeries for AIV V5.2 - Performance Highlights Version 1.1, Tim Pickrell, IBM UK Laboratories, February 13, 2001

MQSeries for Windows NT and Windows V5.2 - Performance Highlights Version 1.1, Richard Crockford, IBM UK Laboratories, February 13, 2001

Acknowledgements. This paper represents the collective efforts of the CSC project team members who worked tirelessly to implement and test the GSTP prototype. Special acknowledges are due to Colin Eby, Randy Hill, Phu O, Tara Pervier Richards, and Khalid Rizvi. Analysis and input critical to the business focus of this white paper were provided by CSC's financial services industry experts on STP and GSTP, particularly Mike Brennan, Kevin Schlumpf, and Ted Stanley. Additionally, we acknowledge the support and contribution of our CSC colleagues Glenn Davidson, Darrow Lebovici, Robert Mansmann, Carol Murphy, Jane Peugh and Robert Switzer. Finally, we would like to acknowledge the exceptional support we received from our partners at IBM - especially David Judge, Craig Heath, David Cho, Rick Bowman, Martha Mora, and Tim Dunn.

Computer Sciences Corporation

Financial Services
275 Second Avenue
Waltham, Massachusetts 02451
United States
+1.781.290.1525

Worldwide CSC Headquarters

The Americas
2100 East Grand Avenue
El Segundo, California 90245
United States
+1.310.615.0311

Europe, Middle East, Africa
279 Farnborough Road
Farnborough
Hampshire GU 14 7LS
United Kingdom
+44(0)1252.363000

Australia/New Zealand
460 Pacific Highway
St. Leonards NSW 2065
Australia
+61(0)2.9901.1111

Asia
139 Cecil Street
#08-00 Cecil House
Singapore 069539
Republic of Singapore
+65.221.9095

About CSC

Computer Sciences Corporation helps clients achieve strategic goals and profit from the use of information technology.

With the broadest range of capabilities, CSC offers clients the solutions they need to manage complexity, focus on core businesses, collaborate with partners and clients, and improve operations.

CSC makes a special point of understanding its clients and provides experts with real-world experience to work with them. CSC is vendor-independent, delivering solutions that best meet each client's unique requirements.

For more than 40 years, clients in industries and governments worldwide have trusted CSC with their business process and information systems outsourcing, systems integration and consulting needs.

The company trades on the New York Stock Exchange under the symbol "CSC."