SQL intersection

High Performance, Scalable Asynchronous Processing Using Service Broker

Preconference Workshop #09

SQLintersection Fall 2014

Workshop: Monday, November 10, 2014

written and presented by
Jonathan Kehayias
http://www.SQLskills.com

SQLintersection
PRECON09

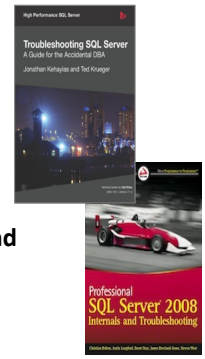
High Performance, Scalable Asynchronous Processing Using Service Broker

Jonathan Kehayias
Principal Consultant
SQLskills.com

SQL intersection

Please silence cell phones

Author/Instructor: Jonathan Kehayias



- **Consultant/Trainer/Speaker/Author**
- **Principal Consultant, SQLskills.com**
  - Email: Jonathan@SQLskills.com
  - Blog: http://www.SQLskills.com/blogs/Jonathan
  - Twitter: @SQLPoolBoy
- **Contributing Author: SQL Server 2008 Internals and**
- **Troubleshooting**
- **Microsoft Certified Master: SQL Server 2008**
- **SQL Server MVP since October 2008**
- **Author of Using Extended Events whitepaper on MSDN**
- **Developed Extended Events Manager SSMS Addin for SQL Server 2008 Open Source project on Codeplex**
- **Member of the Tampa SQL Server User Group and PASS**

SQL intersection

3

Overview

- **Service Broker usage scenarios**
- **Advantages of Service Broker**
- **Basic Service Broker architecture**
- **Conversation architectures**
- **Understanding activation**
- **Performance best practices**

SQL intersection

4

Usage Scenarios (1)

- **Centralized data collection**
  - Multiple sites process transactions locally and use Service Broker to send information to a central office
  - Asynchronous message delivery allows local sales transactions to continue even if a site loses connectivity to the central office
- **Asynchronous trigger execution**
  - Complex logic in DML triggers affect OLTP performance
  - Queuing a message requesting work from a Service Broker service to offload the complex logic into a separate transaction asynchronously

Usage Scenarios (2)

- **Reliable processing of queries**
  - Single transaction is used to read a message, run a query, and return results to the initiating service
  - During a service failure the transaction would roll back, return the message to the queue, and restart processing after recovery
- **Distributed server-side processing**
  - Online ticketing application processes orders using Service Broker to exchange data between payment processing, CRM, and ticket printing applications

Usage Scenarios (3)

- **Batch processing changes**
  - Applications store data to be processed into a queue within Service Broker allowing a program to periodically read and process the data
  - Parallel processing from the queue can handle larger volumes of data quickly and efficiently
  - Conversation groups and dialogs bundle data allowing multiple rows to be handled in a single RECEIVE command

Advantages of Service Broker

- **Database integration**
  - Eliminates the overhead of a separate distributed transaction coordinator
  - The message store and database are maintained at the same transactional point-in-time, eliminating the need for reconciliation processes when one of the two has to be restored
- **Ordering of messages**
  - Messages sent within the Service Broker architecture are received once and only once in the order in which the messages were sent

Advantages of Service Broker (2)

- **Message locking or related messages**
  - Service Broker prevents out of order processing of message within the same conversation group by locking the conversation group during processing
  - Multiple threads can process different conversation groups concurrently increasing scalability and performance

Advantages of Service Broker (3)

- **Loose coupling of applications**
  - The initiating application and target application are loosely coupled, allowing the initiating application to continue processing after sending a message to the target application
  - Loose coupling provides scheduling flexibility, allowing parallel processing
  - Background processing of requests can mitigate high load times against source and target servers through queuing requests for processing

Advantages of Service Broker (4)

- **Automatic activation**
  - Activation allows Service Broker to automatically scale to match the volume of messages arriving in a given service queue
  - Allows programs running inside the database as well as external to the database to take advantage of queue activation
  - Automatically starts a new queue reader when there is work to be performed

Broker and SQL Server Features

- **Event Notifications – watch for Trace/DDL events and respond automatically when they occur**
- **Query Notifications – monitor for changes in results, and leveraged for external queue activation of Service Broker**
- **Database mail – implemented as an external activated application that monitors for messages being queued in msdb**
- **WMI Events – piggy back on Event Notifications to allow integration with Windows Management Instrumentation**

Basic Service Broker Architecture

- **Single database implementation for asynchronous processing**
- **Enable the database for service broker**
- **Create request/reply message types**
- **Create contract specifying request messages are sent by the initiator and reply messages are sent by the target**
- **Create a target/initiator queues/services**

Basic Service Broker Architecture (2)

- **Create the target service specifying the contract to limit conversations**
- **Create the initiator service with no contract**
- **Inter-database Service Broker requires 4 types of objects:**
  - Message types, contracts, queues, and services

Service Broker Components

- **Message types**
  - Define the name of a message type and the type of data that will be contained in the message
  - SQL Server can validate that a message contains valid XML, XML conforming to a specific schema, or no data at all
  - SQL Server can also not validate message data, accepting any data for the message including binary data, XML, or empty messages
  - System message types exist for handling errors and the status of dialogs
  - Default message type is used when a message type is not specified

Service Broker Components (2)

- **Contracts**
  - Define which message types will be used for specific tasks
    - Tasks are defined in Service Broker as initiator (or sender) and target (or receiver)
  - Agreement between services about which message types each of the services will send for a given task
  - Must contain a message type sent by the initiator, otherwise there is no way for the initiator to begin a conversation using the contract

Service Broker Components (3)

- **Queues**
  - Store messages in the database for processing by applications
  - When messages arrive at a service they are inserted into the queue associated with the service
  - Each message is a row in the queue containing:
    - Contents of the message, message type and validation performed
    - Service and contract targeted by the message
    - The conversation the message is associated with
    - Internal information to the queue

Service Broker Components (4)

- **Queues (2)**
  - Applications RECEIVE messages from the queue to get the message for processing within a transaction
    - All messages for a conversation or a conversation group can be processed at once
    - Queues return messages from each conversation in the order the messages were sent
  - May use a stored procedure for internal activation (automated processing)

Service Broker Components (5)

- **Services**
  - Specifies the name of a specific business operation or task implemented in Service Broker
  - Service names are used to:
    - Deliver messages to the correct queue inside of a database
    - Route messages from target to initiator, and back
    - Determine security requirements for new conversations
    - Enforce appropriate contracts for conversations
  - Bound to a specific queue to store incoming messages

Demo

Basic Service Broker architecture

Conversation Architecture

- **Service Broker applications communicate using a dialog conversation or persisted stream of messages back-and-forth between two services**
  - Dialogs provide exactly-once-in-order (EOIO) message delivery using a conversation identifier and sequence numbers
- **Dialogs have two participants:**
  - Initiator – starts the conversation
  - Target – accepts a conversation started by an initiator
  - Messages provide the information exchanged

SQL intersection

Basic Conversation Example

- **Begin a conversation at the initiator specifying (BEGIN DIALOG):**
  - The sending (initiator) service name
  - The receiving (target) service name
  - The contract name for the conversation
- **Sending a message on the conversation (SEND ON CONVERSATION) specifying the message type and the message body to be sent**
- **Receiving the message from the target queue (RECEIVE)**

SQL intersection

Basic Conversation Example (2)

- **Sending a response message from target to initiator (SEND ON CONVERSATION) specifying the message type and the message body to be sent**
- **Ending the dialog on the target (END CONVERSATION)**
- **Receiving the response message on the initiator (RECEIVE)**
- **Ending the conversation on the initiator (END CONVERSATION)**

SQL intersection

Processing Messages

- **The RECEIVE DML statement is used to process and remove messages**
  - Receive a single message into variables with TOP 1 (slowest method)
  - Receive all messages for a single conversation into table variable (faster method)
  - Receive all messages for a conversation group into table variable (faster method)
- **Messages are always received in sequence order, per conversation**

SQL intersection

Processing Messages (2)

- **The RECEIVE statement will lock the conversation using a Service Broker-specific internal lock in SQL Server**
  - This will be a bottleneck if a single conversation is used for all messages
- **Transactions should be used when processing messages, with error handling, to prevent poison messages from occurring**

SQL intersection

Understanding Dialog Conversations

- **Each dialog conversation has a unique conversation handle associated with the conversation**
  - The conversation handle is created on the initiator by the BEGIN DIALOG CONVERSATION command
  - The conversation handle is retrieved by the target when it processes the message(s) with RECEIVE from the queue
  - The target uses the conversation handle to send a response message(s) back to the initiator and ends the conversation with END CONVERSATION

SQL intersection

Understanding Dialog Conversations (2)

- **Certain interactions may require a back-and-forth series of messages on the same conversation with different message types defined by the contract**
  - The initiator should end the conversation when it processes the final message from the target using END CONVERSATION
  - The target should end the conversation based on receipt of a specific message type to avoid orphaning the conversation
- **"Fire and forget" is good for the military but not Service Broker** http://bit.ly/SSB_FireForget

SQL intersection

Demo

Basic intra-database configuration

Dialog Conversation Internals

- **Reliable delivery through receipt-acknowledgement from the target service**
- **Outgoing messages stored in the transmission queue until acknowledged by the target service**
- **If a service is unavailable, messages remain in the transmission queue and retries every minute until the conversation is ended, or the lifetime of the conversation expires,**

Intra-Instance, Inter-Database Messaging

- **Sending cross database messages in the same instance requires:**
  - Configuring security
    - TRUSTWORTHY ON for both databases (easiest but least secure)
    - Configure full dialog security using certificates and a remote service binding (most secure)
  - Creating the broker components in both databases (message types, contracts, services and queues)
  - Use specific naming for services (typically //DBName/ServiceName) to prevent "load balancing" of messages
- **The conversation pattern for intra-instance messaging is the same as for intra-database messaging**

Demo

Intra-instance, inter-database configuration

Inter-Instance Messaging

- **Sending messages between instances of SQL Server requires:**
  - Creating a Service Broker endpoint on each instance
  - Configuring dialog security by exchanging certificates between instances
  - Creating routes in each database and msdb for the services
  - Creating a remote service binding to associate certificate users to the remote service route for dialog security
- **The conversation pattern for inter-instance messaging is the same as for intra-database messaging**

Demo

Inter-instance configuration

Activation

- **Allows automatic processing of messages in a queue**
- **For applications inside of the database a stored procedure can be used to process the queued messages**
  - Referred to as internal activation
  - Stored procedure is associated with a queue and starts automatically
  - Must have an owner and use EXECUTE AS for security context
  - Certificate signing should be used for cross-database access

Activation (2)

- **Service Broker also supports activation outside of the database through an application or service monitoring the queue**
  - This can be written explicitly in the application or service
  - The External Activation feature can alternatively be configured to execute the program or service

Demo

Activation demos

Troubleshooting Service Broker

- **Determine the nature of the problem:**
  - Did the message actually get sent and committed?
  - Check initiator sys.transmission_queue for transmission_status
    - The sender cannot send the message (e.g. security issues, no remote service binding, no route to target service)
  - If the transmission_status is empty – message sent but target does not accept
    - Trace the target server for Broker:Conversation, Broker:Message Undeliverable, Broker:Remote Message Acknowledgement, and Audit Broker Conversation
    - The TextData of the Broker:Message Undeliverable event will point to the problem
  - Message reaches target but target cannot send acknowledgement?
    - Verify return route to the initiator was configured correctly by tracing the Broker:Message Classify event on target server
    - Use GET_TRANSMISSION_STATUS(@handle) to get status of the acknowledgement

SQL intersection

Troubleshooting Service Broker (2)

- **Endpoint TCP port enabled in the firewall**
- **Check certificate expiration dates for dialog and transport security**
  - Certificates should be replaced before expiration to prevent problems
- **SSBdiagnose.exe**
  - Shipped with SQL Server 2008 to help diagnose problems with Service Broker configurations
  - Can validate a Service Broker setup in CONFIGURATION mode
    - Check service names, contracts, routing, and security configuration
  - Actively trace and monitor conversation activity in RUNTIME mode
  - Can output results in XML format or plain text to screen
  - Can be configured to ignore specific errors or only report errors of a specific severity (error/warning/info)

SQL intersection

Demo

Troubleshooting Broker Configurations

Poison Messages

- **All RECEIVE statements are transactional and when a transaction rolls back the message will be placed back on the queue**
- **A poison message is caused by a loop of RECEIVE and ROLLBACK for the same message on a queue**
  - Service Broker will deactivate the queue after five times
  - QUEUE_DEACTIVATION Event Notification can monitor this
  - SQL Server 2008 R2 and onward allow disabling poison message handling
- **Activation stored procedures and external applications should be designed to handle poison messages**
  - Messages must be processed in order meaning a rollback will cause the procedure or application to receive the poison message at next RECEIVE
  - Error handling within the transaction can prevent poison messages

SQL intersection

Demo

Poison messages

Performance Best Practices

- **Configure max queue readers for busy systems to allow parallel processing on different reader threads**
  - When traffic spikes and a thread fails to 'drain' the queue 5 seconds (RECEIVE returns empty), then the activation mechanism will start a new reader thread
  - When traffic reduces and too many listeners are activated, only the minimum required to handle the traffic are kept alive, while the surplus time out
  - When the traffic ends completely for a period, all threads/processes may go away after timing out, but when a new message arrives a processing thread will be activated to process it
- **Disable XML validation on production systems**
  - Appropriate testing should have guaranteed message validity in production deployments

SQL intersection

Performance Best Practices

- **Reuse conversations**
  - Setting up and tearing down a conversation for each message is expensive
  - Guarantees order of messages and changes
  - Multiple messages on the same conversation can speed up processing by as much as ten times on the RECEIVE side
  - Use the 150 trick explained in the Service Broker: Performance and Scalability Techniques whitepaper (http://bit.ly/SSB_Perf) for high volume workloads
- **Receive all of the messages for a conversation handle or conversation group in one operation instead of row by row**

Load Balancing

- **A service with a specific name can be created in multiple database on multiple instances**
  - Each database must have a specific/unique broker instance ID
    - The <service name; broker ID> pair uniquely identify the service within a specific database with the provided broker ID
  - Specifying just the target service name without specifying the broker ID in BEGIN DIALOG leaves the broker ID open for load balancing
  - Requires that all routes specify the broker instance ID for the service name
- **Load balancing doesn't pick a random route, but uses deterministic routing, using a hash of the dialog ID to select a route for the lifetime of the dialog**
  - After the first Acknowledgement is returned from the target, the targets broker ID becomes locked on the initiator side of the conversation to prevent further load balancing of the conversation

Load Balancing Configuration

- **Create the InitiatorService and InitiatorQueue on ServerA, DatabaseA**
- **Create the TargetService and TargetQueue on ServerB, DatabaseB**
- **Create the TargetService and TargetQueue on ServerC, DatabaseC**
- **Create route on ServerA to TargetService on ServerB, DatabaseB**
- **Create route on ServerA to TargetService on ServerC, DatabaseC**
- **Create route on ServerB to InitiatorService on ServerA, DatabaseA**
- **Create route on ServerC to InitiatorService on ServerA, DatabaseA**

- **If default routes have been dropped:**
  - Create local route in msdb on ServerA to InitiatorService in DatabaseA
  - Create local route in msdb on ServerB to TargetService in DatabaseB
  - Create local route in msdb on ServerC to TargetService in DatabaseC

SQL intersection

Demo

Improving Performance and Scalability

Review

- Service Broker usage scenarios
- Advantages of Service Broker
- Basic Service Broker architecture
- Conversation architectures
- Understanding activation
- Troubleshooting Service Broker problems
- Performance best practices

Questions?

Don't forget to complete an online evaluation on EventBoard!

PRECON09

High Performance, Scalable

Asynchronous Processing Using Service Broker

Your evaluation helps organizers build better conferences and helps speakers improve their sessions.

Thank you!