WHITEPAPER

High Availability Embedded Messaging for ISVs

Improving the Availability, Reliability and Fault-Tolerance of Packaged Applications and Hardware Devices

November 2006

PROGRESS SOFTWARE

TABLE OF CONTENTS

1. INTRODUCTION

The growing importance of conducting business 24x7 and the increasing need for real-time operations are driving the demand for reliable and highly available packaged applications and hardware devices. These systems consist of several tiers of servers, e.g. database server, application server, web server and messaging server, each of which have particular availability characteristics and represent possible points of failure. As packaged applications become more loosely coupled and need to interoperate with other applications, message-oriented middleware (MOM) is becoming a standard component. Additionally, MOM is being included on hardware devices to improve reliability and fault-tolerance of software components. However, not all MOMs are alike, especially when it comes to implementing high availability.

Increasingly, end-users of your applications are demanding continuous service availability with no time for outages, whether planned or unexpected. Outages, poor performance, and scheduled interruptions disrupt business operations, raise costs, and damage customer satisfaction. The costs of outages can be seen in productivity losses both in the IT organization and the business units it serves, lost revenue, and penalties, such as regulatory fines.

The cost of an outage is the sum of:

Productivity of affected users = hourly cost of affected users x hours of disruption

+Lost IT productivity = hourly cost of affected staff x hours of lost productivity

+Impact to customer service and credibility

+Lost revenue = lost revenue per hour x hours of outage

+Other business losses incurred

Overtime payments = hourly wages x overtime hours
+ Wasted goods
+ Financial penalties or fines

Typically, lost revenue represents the greatest financial exposure but can be the most difficult to quantify. Certain industries are more dependent on the real-time operation of their enterprise systems. The table below characterizes these costs for several different types of businesses.

Average costs of unplanned outages for U.S. industries[1]

Retail brokerages	$6.45 million per hour
Credit card sales authorization	$2.6 million per hour
Infomercial/800-number promotion	$199,500 per hour
Catalog sales center	$90,000 per hour
Airline reservations	$89,500 per hour
ATM services providers	$14,500 per hour

[1] InternetWeek 4/3/2000 and "Fibre Channel: A Comprehensive Introduction", R. Kembel, 2000, p.8, based on a survey by contingency planning research.

In addition, when system outages are unexpected, user impact is significantly more severe than a planned outage. "Human factors gurus noticed that response time delays to user inquiries of more than 2 seconds broke the inquirer's concentration, requiring a mental reset with lost productivity measured in minutes. System downtime longer than about 20 minutes was actually reflected in users' changing their task at hand, disrupting processes and yielding effective outages measured in hours when seen from the user's perspective."[2]

Finally, system outages can damage not only the reputation of the end-user but also the vendor that supplied the application, exposing both to customer defection and ongoing legal risks such as lawsuits and regulatory penalties.

To differentiate your offering, maximize revenue, and address customer demands, ISVs need to architect high availability into their applications. This paper discusses the benefits of including messaging middleware in your application, the problems associated with current approaches to fault-tolerant enterprise messaging solutions and introduces a new and superior paradigm for companies to improve operational availability, the Sonic Continuous Availability Architecture ™.

2. Forrester Research: March 15, 2004 "An Executive Guide To High Availability" by Bob Zimmerman

2. THE BUSINESS NEED FOR ENTERPRISE MESSAGING

The importance of enterprise messaging infrastructure to an application and device grows as companies seek to improve operational productivity. Even during scheduled systems downtime, it is necessary that other systems, departments, business units, and partners can continue working. The need to connect to systems outside the firewall and to transfer mission-critical data has made it critical to support a reliable, available, and secure exchange of information. Furthermore, in the event of a system crash, it is extremely important that there is no loss of data.

As an ISV or hardware vendor you recognize that you have a choice of building or buying components for your system. The three main options for message-oriented middleware (MOM) are: the application server MOM, an independent MOM, or open source MOM. To determine the right choice for your system, there are several factors to consider, such as:

> How easy is the product to embed into the application or device?

> How easy will it be for an end-user to install, configure and manage?

> Does the embedded middleware support current and future initiatives, such as multi-platforms, open standards, and multiple connectivity options?

> Will it scale as my business and end-user's business grows?

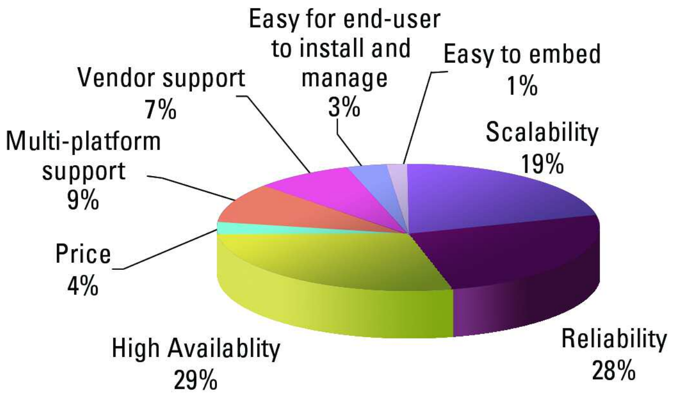
> Will the product increase the availability and reliability of my offering?

> Does the product fit my business model?

> Can I bet my business on this technology and the vendor I select?

Each of these factors is important in their own right, but which one is the most important? A recent survey of over 100 ISVs and hardware vendors regarding the most important factor when selecting embedded middleware for their application showed:

What is THE most important factor when selecting embedded middleware



As high availability and reliability equal 57%, one can conclude that if an application does not consistently deliver expected results, that the other factors are less important – or failure is not an option. But what are some of the typical use cases for high availability embedded messaging? Some of the more common scenarios include:

> Medical equipment companies who require 99.999% availability of their device.

> Financial ISVs whose end-users require no downtime, zero lost trades and in-order messaging

> Communications hardware providers who need highly available and reliable messaging for their Session-Initiated Protocol (SIP) applications, so push-to-talk messages are delivered

> Government system integrators providing mission-critical applications to the armed forces

The rest of this paper presents the technical details of high availability messaging solutions.

3 TECHNICAL ENTERPRISE MESSAGING REQUIREMENTS

To address the business challenges mentioned in the previous section, most companies have implemented an IT infrastructure based on a RPC (remote procedure call) type of messaging system. RPC based systems, such as those provided by application servers have become popular. RPC messaging evolved during the 1990's into CORBA, COM/DCOM, and RMI. Unfortunately, these systems are very brittle, as applications are tightly coupled with synchronous communications. They employ point-to-point connections, and lack a common interface.

An RPC type messaging system requires that enterprise client applications be developed along very constrained guidelines. The synchronous nature of the system requires that all systems and applications be available when they are needed. Each client application requires an intimate knowledge of the API's associated with other applications. As applications change or new ones are integrated into the system, the number of interfaces required grows exponentially. While connecting a limited number of applications together using this approach may be reasonable, a larger heterogeneous environment could suffer from these tightly-coupled synchronous interactions.

In the 1990's, message-oriented middleware (MOM) systems evolved as a more scalable approach consisting of a loosely-coupled, asynchronous architecture. A MOM-based system manages functions such as guaranteed delivery of messages and error handling through a standardized messaging interface. Since applications can communicate asynchronously with one another, there is no requirement that all systems be running in order to maintain the health of the application network. Overall, MOM represents a significant improvement over the RPC model of communication as there are fewer network connections, increased flexibility in deployments (less brittleness), and less coding required to support configuration changes.

3.1 Java Message Service

In 1998, Sun Microsystems released the Java Message Service (JMS). The JMS API, developed by Sun in close cooperation with leading enterprise messaging vendors, combined key elements of RPC and MOM. Enterprise messaging is now recognized as an essential tool for building enterprise client applications. Detailed information regarding JMS may be found at:
http://java.sun.com/products/jms/.

In 1998, Progress Software developed an enterprise messaging product, SonicMQ®, based on JMS that is acknowledged as the industry's most robust and resilient standards-based enterprise messaging system . Information regarding SonicMQ can be found at:
http://www.progress.com/sonic

Enterprise messaging and JMS in particular provides a reliable, flexible service for the asynchronous exchange of critical business data and events in an application and throughout an enterprise. JMS enables client applications to communicate with each other using a well defined and loosely coupled messaging protocol. The JMS API adds to this a common API and provider framework that enables the development of portable, secure and reliable message based applications.

Many enterprise client applications cannot tolerate dropped or duplicated messages. It is critical for many JMS applications to ensure delivery of a message once and only once. This level of message service delivery is referred to as exactly once message reliability.

JMS defines several message reliability mechanisms. The most reliable way to produce a message is to send the message as a persistent message within a transaction. The most reliable way to consume a message is to receive the message from a queue or durable subscription within a transaction.

3.2 Message Failures

In normal operations, JMS provides exactly once message reliability. However, none of the JMS reliability mechanisms can provide exactly once reliability in the event of hardware, networking or operating system failure. There are four classes of message failures that can occur which have a critical impact on business operations.

1. Trapped messages

After system failure, messages are lost in the failed messaging system and never received by the client application.

2. Duplicate messages

After system failure, messages are left in an unknown state. Upon failover, this may result in duplicate messages being sent by either the client application or the messaging broker, e.g. a financial service debit order is sent twice.

3. Out-of-order messages

After system failure, message delivery and transit is left in an in doubt state. Upon restart, this can result in out-of-order messages being received by the client application, e.g. a "Cancel" or "Update" order is received before the initial order.

4. Broken transactions

After system failure, transactions are left in an incomplete state. Upon restart, the broken transactional messages must be rolled back and discarded.

These failures can cause a significant delay in the completion of operational business processes. In all cases, a "forensic" team is required to intervene and reconstruct the message state. This involves the rollback of transactional messages, recovery of trapped messages, identification and elimination of duplicate messages and the reconstruction of out-of-order messages. If this reconstruction of the message state is not completed, then enterprise client applications will receive corrupted message streams resulting in incomplete or inaccurate transactions.

In order for an enterprise messaging system to provide exactly once message reliability in the face of hardware, network or system failures, IT must provide system redundancy. This redundancy is provided through a primary and secondary (backup) broker pair, which must support the following fault tolerant requirements:

1. Full-state recovery:

In the event of a system failure, the secondary broker must be able to assume the role of its failed primary partner. The secondary broker must gracefully recover the full message state prior to the failure.

2. Hot-failover:

In the event of a system failure, the secondary broker must transition to an active state with minimum failover latency. Minimum latency ensures that client application buffers will not overflow or applications fail.

3. Client Failure Transparency:

In the event of a system failure, enterprise applications must transparently failover to the secondary messaging broker. The enterprise application connection to the message system stays warm until the transition to the secondary broker is complete.

4 TRADITIONAL FAULT TOLERANT ARCHITECTURES

When a message travels from a client to a destination, it may traverse many networks, systems, and applications.   Higher levels of availability can be achieved by hardening the various components, such as clients, middleware, and databases. Fault tolerance for the messaging infrastructure has traditionally been based on combining two or more servers which are tightly coupled and centrally managed. The servers are configured to provide an enterprise client application with a secondary server in the event of any system failure. This is referred to as "failing over" from one server to another.

A high-availability/fault tolerant system appears to users and applications as a single environment. In addition to providing increased application availability, clustering technology can also be used to increase system capacity and provide administrative efficiency.

High availability (HA) solutions have been available since the 1980s when used in DEC's VMS systems. IBM's sysplex is an HA approach for a mainframe system. Microsoft, Sun Microsystems, and other leading hardware and software companies offer HA packages that are said to offer scalability as well as availability. As traffic or availability assurance increases, all or some parts of the system can be increased in size or number.

High availability architectures ensure that an operating system crash does not cause a lengthy application outage, and it provides an environment that supports online maintenance and upgrading of the individual computer systems that compose the system. High availability can be implemented through both hardware solutions and software solutions.

4.1 Hardware High Availability Architectures

Hardware high availability (vertical scaling) is a hardware-based method in which a group of servers act like a single system. In common practice, this architecture is created by installing a number of blade servers on the machine that will control the system. Each of the blade servers functions independently of the others, although they all respond to the same requests. The operating system of the controlling server is responsible for monitoring the system and performing administrative tasks, such as deciding when failover is necessary and assigning the load of a failed node to a functioning server.

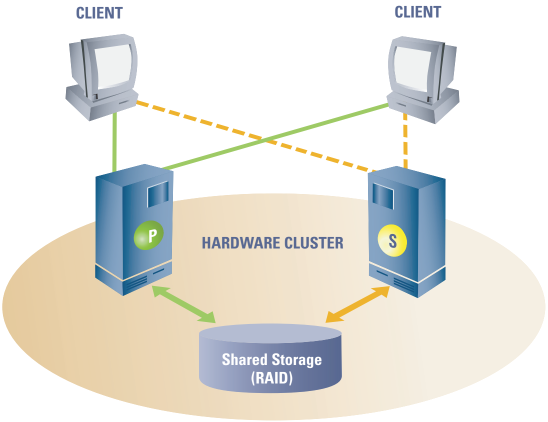
Hardware high availability architectures may be active-passive, in which case some redundant servers are reserved for failover duty and do not run any applications of their own. It can also be active-active, in which case all servers run their own applications but also reserve resources to allow them to perform failover duty for each other. Hardware fault tolerance involves the purchase and configuration of expensive specialized hardware; therefore, it is used predominantly in high-end enterprise systems.

4.2 Software High Availability Architectures

Software high availability (horizontal scaling) is a method of turning multiple servers into a fault tolerant solution. Operating system clustering software is installed on each of the servers in the system, and typically the primary and secondary servers have mirrored applications. Each of the servers maintains the same information and collectively they perform administrative tasks such as load balancing, determining node failures, and assigning failover duty.

Because servers can easily be added or removed from the cluster as needs dictate, software fault tolerance is a scalable solution. However, the typical software fault tolerance architecture (Figure 1) for enterprise messaging does require the use of a shared database to maintain current and complete message state information. Two messaging servers (primary and secondary) designate this messaging database at a network accessible location. Only the primary server will have read/write access until failure. A RAID array database system is used to ensure that the database doesn't expose a single point of failure.

Figure 1. This diagram shows a traditional fault tolerant solution. Upon failure of the primary server, the secondary (backup) server starts up and begins the recovery process from shared storage. Messaging problems may arise such as trapped messages on the failed server, duplicate messages sent and received, out-of-order messages and broken transactions



In the event of a primary server failure, the third-party system will initiate a fail-over process to the secondary server. The secondary server begins reading message state information from the shared database and initializing internal logs and files. This failover process may take several minutes or longer. The fault tolerance system also notifies enterprise applications of the failure and provides re-connection information. The client applications connect to the secondary server and complete the recovery process.

Despite this complex process, the failover and re-initialization process still does not provide exactly once message reliability. Enterprise client applications may still encounter trapped, duplicated, out-of-order and broken transactional messages. All four messaging failure classes can impact the operational system.

5 SONIC CONTINUOUS AVAILABILITY ARCHITECTURE™

Sonic has a unique messaging architecture, the patent-pending Sonic Continuous Availability Architecture (CAA). Superior to traditional high availability solutions provided by current enterprise messaging vendors, Sonic CAA provides users with significant benefits of higher levels of operational availability as well as reduced development, deployment and administration costs. Sonic CAA provides high availability for the messaging layer, including the Sonic message brokers, Sonic clients and the communications among clients, brokers, and destinations, guaranteeing exactly once message reliability under both normal and failure conditions. Sonic CAA eliminates the requirement for expensive RAID, OS clustering software or third-party HA frameworks in the messaging layer. No matter how complex, in-process transactions continue to their destinations without any costly roll back or recovery time.

Sonic CAA (Figure 2) is based on primary/secondary broker replication through backchannel synchronization and fault tolerant client connections. This architecture supports the extended business enterprise in a way not possible until now, and fully complements existing high availability solutions, such as those for the database, application server, and web server.

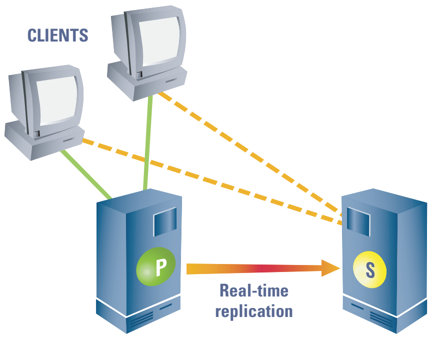


Figure 2. This diagram illustrates the Sonic CAA architecture, which provides real-time replication between primary and secondary servers without the risk of trapped, duplicate, out-of-order messages or broken transactions.

5.1 Broker Replication

The central concept of Sonic CAA is backchannel broker replication. The primary broker provides message services to enterprise client applications. Concurrently, the primary broker uses a backend replication channel to stream messaging state to a secondary broker. This replication channel is supported on a private network dedicated to the synchronization of the broker state and messaging data. The primary/secondary broker pair uses the replication channel to routinely seek the heartbeat of the other and watch for any interruption in the data flow or connection. The secondary broker accepts no client connections while in its hot standby role, but is prepared to immediately transition to the active role should the primary broker become unavailable.

In the event of a system failure, the secondary broker assumes the active role. All client applications failover from the primary broker and reconnect to the designated secondary backup broker. This hot-failover process is immediate and transparent to the client applications. The secondary broker in the active role is sensitive to re-establishment of the replication channel. This reconnection may come from a recovery of the primary broker or from a replacement primary broker. Once reconnected, the brokers actively replicate to achieve synchronization at which point the primary broker is again prepared to resume the active state.

State Diagram

The behavior of the broker is determined by the broker's replication state. A state diagram (Figure 3) serves to illustrate the roles, states and events associated with the fault tolerant broker pair.

The broker states are grouped into two main roles, the active role and the standby role. A broker in the active role is providing messaging services to enterprise application clients. The broker in the standby role absorbs replicated state information and is continuously prepared for a hot-failover.

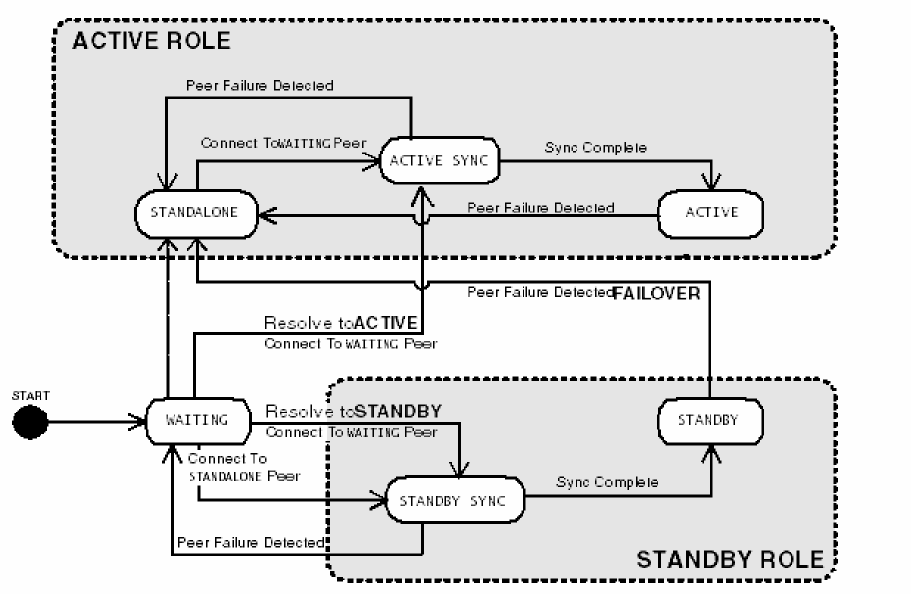


Figure 3. State diagram showing roles, states, and events of fault tolerant broker pair

Upon startup, a broker enters the **WAITING** state and does not accept client connections. A broker will change state if one of the three events occurs:

- A broker in the waiting state may transition directly to **STANDALONE** state in response to administrative intervention, or if it is configured to do so.
- If a replication connection is established, and the other broker is in the **STANDALONE** state, the broker will transition to the **STANDBY SYNC** state and begin runtime synchronization.
- If a replication connection is established, and the other broker is also in the **WAITING** state, the brokers choose roles based on their previous role, synchronization state and administratively defined preferences.

A primary broker is in the **STANDALONE** state if it is actively servicing client and cluster operations but no secondary broker is running. While in this state, if a replication connection is established, and the other broker is in the **WAITING** state, the primary broker will transition to the **ACTIVE SYNC** state.

In the **ACTIVE SYNC** state the broker is synchronizing state with the standby broker while at the same time serving client applications. Completion of the runtime synchronization protocol causes a transition to the **ACTIVE** state.

Once the primary broker has completed synchronization, it transitions to an **ACTIVE** state and begins actively streaming messaging state and messaging data to its corresponding secondary broker which is currently in the **STANDBY** state. At this point if there is a failure of the primary broker, the secondary broker will immediately transition to the **STANDALONE** state.

When a secondary broker enters the **STANDBY SYNC** state it dynamically synchronizes its state with the active broker that is in the **ACTIVE SYNC** state while absorbing the live replication stream. Completion of the runtime synchronization protocol causes a transition to the **STANDBY** state

Once the secondary broker is in the **STANDBY** state it absorbs the live messaging state and data from the primary broker that enables it to immediately assume a **STANDALONE** role if a failure of the primary is detected. The transition from **STANDBY** to **STANDALONE** transition is referred to as broker failover.

5.2 Replication Connections

Sonic CAA has been implemented using a unique replication protocol that supports dynamic synchronization and live state streaming between a fault-tolerant broker pair. Sonic CAA supports and encourages the use of multiple replication connections in order to provide redundant network support between the primary and secondary brokers. Only one connection is used to handle replication traffic at any time, but in the event of a connection failure the broker pair will automatically switch to the next replication connection without interrupting the replication process.

Multiple replication connections (Figure 4) increase the resiliency of the fault-tolerant broker pair. If all replication connections fail, it is assumed that the primary broker has failed and the secondary broker will transition to an active role. This redundancy is important because under no circumstances should the primary and secondary broker go into active state concurrently—a condition referred to as a *dual active partition.*

Each replication connection is assigned a weight that is used to determine its precedence in handling replication traffic. Replication connections with a weight of 0 will not handle replication traffic and will only used for heartbeating to avoid false partitions. Typically a 0 weight connection would be created on the public network to protect against a failure of the private network causing a partition.

If a replication connection fails and there is a lower weight replication connection available, replication will continue, uninterrupted, on that connection. If the higher weight connection is re-established, replication traffic will be resumed on that connection, giving administrators flexibility on choosing the highest bandwidth network path for replication traffic.

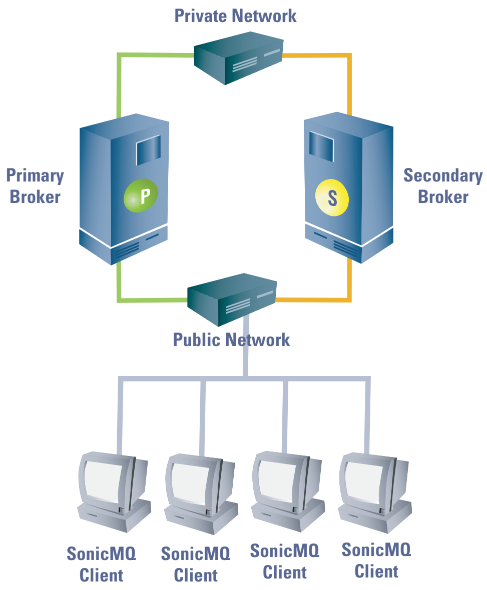


Figure 4. Multiple replication connections provide continuous availability of network communications

5.3 Continuously Available Client Connections

Sonic Continuous Availability Architecture provides resilient connections for enterprise client applications. A standard JMS connection is immediately dropped when the broker or network fails. The dropped connection forces the client application to explicitly deal with this external event.

Under the covers, a fault-tolerant connection will attempt to immediately reconnect when it encounters a broker or network failure. The client reconnection protocol responds in several different ways depending on how the broker is configured and the nature of the failure.

> If the network experiences a transient failure, the fault-tolerant connection will repeatedly try to recover the connection until the network returns to a normal state.

> If the client application has redundant network pathways to the broker and one pathway should fail, the fault-tolerant connection will use the other pathways to resume the connection.

> If the client application is connected to a secondary broker that fails, the fault tolerant connection will repeatedly try to reconnect to the broker, until it is recovered and restarted.

> If the client application is connected to a fault-tolerant broker pair and the primary broker fails, the fault-tolerant connection will immediately connect to the secondary broker.

When the client application successfully reconnects, it immediately executes several state and synchronization protocol exchanges, allowing it to resynchronize client and broker state and resolve in-doubt messages. While the connection successfully resynchronizes client and broker state the client applications continue operations.

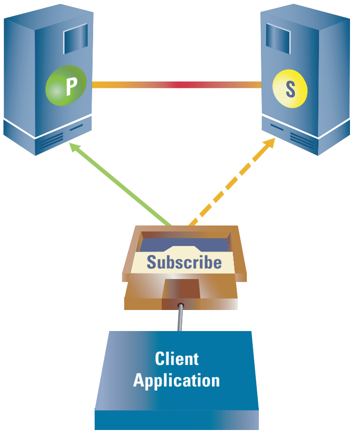


Figure 5. Fault tolerant client connections

5.4 Message Reliability

Sonic CAA provides different levels of message reliability for client applications connected to a message broker. These qualities of service describe the number of messages a client application will receive in the event of a system failure. These three service levels include: “at most once” (AMO), “at least once” (ALO) and “exactly once” (EO). These service levels are controlled by the client applications. The table below provides a mapping of these message service levels.

> Applications may choose to use publish-subscribe or point-to-point messaging.

> Client applications producing messages may be persistent or non-persistent.

> Client applications consuming messages may select durable or non-durable subscriptions.

> Client applications may connect to a fault-tolerant broker pair.

> Client applications may select standard or fault-tolerant connections.

Message Reliability

Message Producer		Message Consumer			
Connection Type	Delivery Mode	Standard Connection		Fault Tolerant Connection	
		Topic	Topic (Durable Subscription) or Queue	Topic	Topic (Durable Subscription) or Queue
Standard Connection	DISCARDABLE	At most once[1]	At most once[1]	At most once[1]	At most once[1]
	PERSISTENT	At most once[2]	At least once[1, 2]	At most once[1]	Exactly once[1]
	NON_PERSISTENT	At most once[1]	At most once[1]	At most once[1]	At most once[1]
Fault-Tolerant Connection	DISCARDABLE	At most once	At most once	At most once	At most once
	PERSISTENT	At most once[3]	At least once[2]	At most once	Exactly once
	NON_PERSISTENT	At most once	At most once	At most once	At most once

This table shows that when both the producer and consumer client applications use standard connections, the messages will be received at most once, except for persistent/durable messages. In the case of persistent/durable messages, the message will be received at least once, but with possible duplication. Therefore, a message can not be guaranteed to be received exactly once when using standard connection.

In the case of producer and consumer client applications which use a combination of standard and fault tolerant connections, the quality of message reliability only slightly improves. Three special cases exist:

> 1. In the case of a standard connection failure, if the last message sent was in doubt, the producer client application may attempt to resend the message. This causes the generation of a duplicate message if the broker had received the original message. According to JMS Specification, this is not a redelivery since the message was delivered from a new session. This ambiguity is resolved with a fault-tolerant condition: PERSISTENT messages are exactly-once; DISCARDABLE and NON_PERSISTENT messages are dropped in a failure.

> 2. In the case of a standard connection failure, the acknowledgement for the last message may be lost. In this case the broker will redeliver the message in accordance with the JMS Specification.

> 3. If a consumer client application reconnects using a standard connection at the same time a fault-tolerant producer is reconnecting, it is possible that the producer will resend a message that had been delivered to the previously connected client. According to the JMS Specification, this is not a redelivery since the message was delivered to a new session.

Exactly once message delivery can only be guaranteed when both the producer and consumer are using fault-tolerant connections and messages are sent as persistent/durable.

5.5 Transaction Reliability

Sonic CAA also adds an additional layer of reliability to transactional messaging. In the absence of fault tolerance, a client application must be able to handle the ambiguity associated with a failure during a transaction commit. If the broker or connection fails, the result of the commit operation is in doubt; it is unclear to the application if the transaction was committed or not.

This ambiguity is lifted for a client that is using a fault-tolerant connection. During reconnect this ambiguity is transparently resolved and the commit succeeds. Sonic CAA also presents a great advantage to clients that experience a failure part way through a transaction since the transaction can continue uninterrupted after a failure. For a non-fault tolerant client all work that had been done up to the point of failure would have been rolled back, and the transaction would need to be restarted.

5.6 Flexible Deployment Solutions

Sonic Continuous Availability Architecture adds high availability and resilience to the messaging infrastructure. To provide continuous availability in large scale and diverse deployments, fault-tolerant broker pairs can be configured across heterogeneous hardware platforms. It is not a requirement to have identical hardware for primary and secondary servers. In addition, no special clustering or high-availability software management software is required. This flexibility enables fault-tolerant broker pairs to be configured to increase the utilization of computer resources. The three machine configuration shown in Figure 6, hosts three fault-tolerant broker pairs providing a robust and powerful enterprise messaging system.

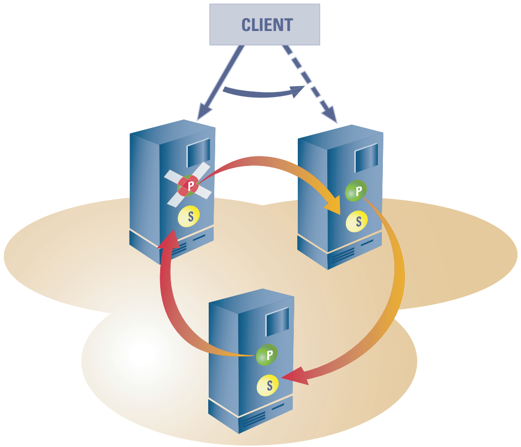


Figure 6. More efficient resource utilization and higher cluster performance is achieved by distributing fault-tolerant broker pairs across machines

In addition, with Sonic's advanced clustering and Dynamic Routing Architecture® capabilities, high availability can be extended across the enterprise providing continuous availability to remote locations and business partners. (Figure 7)

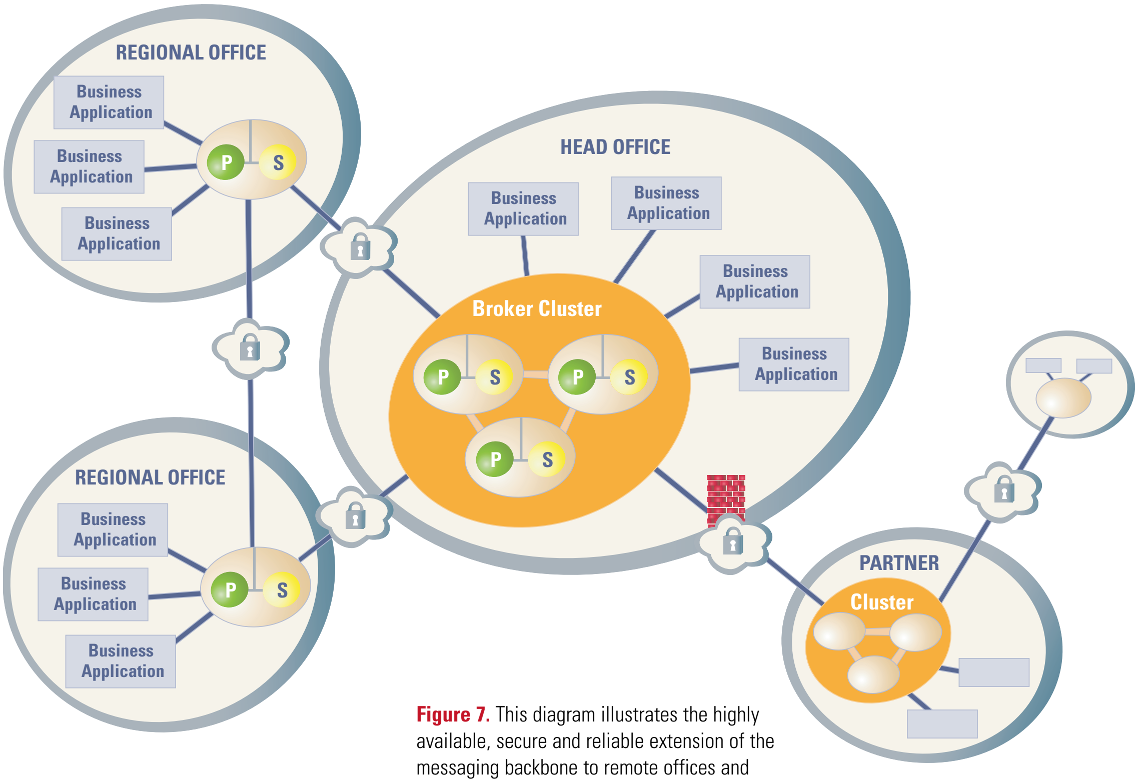


Figure 7. This diagram illustrates the highly available, secure and reliable extension of the messaging backbone to remote offices and business partners

5.7 Benefits for ISVs

For the ISV, this means that high availability messaging becomes an embedded component of your application. You and your customers no longer need to build a complex set of add-on fault tolerant solutions. With improved levels of availability, not only do you have a significant product differentiator, but your customer satisfaction will increase.

6 SUMMARY

Increasingly, your customers are demanding continuous availability with no time for outages, whether planned or unexpected. To differentiate your offering, maximize revenue, and increase customer satisfaction, ISVs need to architect high availability into their applications.

Sonic raises the bar for high availability and fault-tolerant messaging, reducing operational risk as well as the development time and administration complexity of high availability solutions. The patent-pending Sonic Continuous Availability Architecture (CAA) addresses the problems caused by messaging system failure, so packaged applications and hardware devices continue to operate. Sonic CAA provides high availability for Sonic message brokers, Sonic clients and the communications among clients, brokers, and destinations, thereby eliminating the requirement for expensive RAID, OS clustering software or third-party HA frameworks in the messaging layer. No matter how complex, in-process transactions continue to their destinations without any costly roll back or recovery time.

“The potential impact of system downtime in our messaging infrastructure is simply enormous, with almost all of our trades valued in millions of dollars. The high availability solutions that are out there fall short because even when the brokers come back online, in-flight trades may be lost. Sonic solves this problem,” said John Brann, chief architect at FXall, the leading foreign exchange trading portal. “Configuration was exceptionally easy with Sonic. We literally had our first successful tests completed in less than a day.”

For more information on how you can improve the availability of your messaging architecture, or to download an evaluation copy of Sonic products visit http://www.progress.com/sonic .

About Sonic and Progress Software

Sonic™ products from Progress Software help IT organizations achieve broad-scale interoperability of IT systems and the flexibility to adapt these systems to rapidly changing business needs. The Sonic products include SonicMQ®, the industry's only continuously available JMS enterprise messaging system, and Sonic ESB®, the world's first and market-share leading ESB. Sonic products simplify the integration and flexible reuse of diverse and often proprietary business systems by manipulating them as modular, standards-based services which can be rapidly combined to serve the business in new ways.

Progress Software Corporation (Nasdaq: PRGS) provides application infrastructure software for the development, deployment, integration and management of business applications. Our goal is to maximize the benefits of information technology while minimizing its complexity and total cost of ownership. Progress can be reached at www.progress.com or +1-781-280-4000.