IBM MQ as a Reactive System

Lee Wheaton

September 25, 2018 15:45
September 26, 2018 15:30

IBM MQ as a Reactive System

- Disclaimer
  - The web sites listed in this presentation belong to their respective owners, and as such, all rights are reserved by these owners. These web sites are provided for informational purposes only and are provided “AS IS” without warranty of any kind. It should not be construed that the author of this presentation or MQTC is endorsing or recommending these websites or the companies and products listed therein. Any visits you make to these web sites are done strictly at your own risk.

  - As of September 20, 2018, these web links were still valid URL’s.

Session Topics

The following topics will be leveraged in this session to highlight the features, flexibility and capabilities of IBM MQ as a Reactive System:

- Transaction & Software Pipeline Principles
- The Reactive Manifesto
- IBM MQ as a Reactive System
- IBM MQ Reactive Programming

Transaction & Software

Pipeline Principles

Pipelines and Enterprise Architecture

Enterprise Architecture principles and concept of Interchangeable Pipes (or plug ‘n play):

- “Principle 16: **Technology Independence**
  Statement: Applications are independent of specific technology choices and therefore can operate on a variety of technology platforms.
  Rationale: Independence of applications from the underlying technology allows applications to be developed, upgraded, and operated in the most cost-effective and timely way. Otherwise technology, which is subject to continual obsolescence and vendor dependence, becomes the driver rather than the user requirements themselves. Realizing that every decision made with respect to IT makes us dependent on that technology, the intent of this principle is to ensure that Application Software is not dependent on specific hardware and operating systems software.”
  Implications: Middleware should be used to decouple applications from specific software solutions.
  Source: TOGAF: 20.6.3 Application Principles

- “Loose coupling permits a clean separation of concerns (temporal, technological, and organizational) between the parts in a solution, enabling flexibility and agility in both business processes and IT systems.”
  Source: IBM developerWorks: Exploring the Enterprise Service Bus

Transaction and Software Pipelines: Background (1)

- Chip manufacturers used to be able to double the speed of sequential processors around every two years (see Moore's law). As this paradigm was becoming unsustainable, chip manufacturers changed their approach to produce multicore processor chips to allow them to maintain the ability to increase processor performance.
- However, to use multicore processors to their fullest, applications need to employ parallel processing at the processor, task and data level to derive the best benefits, particularly for environments with high traffic volumes requiring low response times.
  Reference: Intel: Parallel Computing: Background
- "The challenge of finding a business-oriented approach to parallel processing is answered by Software Pipelines. The architecture is highly scalable and flexible. It executes business services independent of location, and in such a way as to maximize throughput on available computing resources, while easily meeting a vast array of complex business application requirements."
  Source: Cory Isaacson: Software Pipelines and SOA
- The Reactive Manifesto, discussed later, expands further on pipeline principles and reactive architecture. However, it does not go into the same level of detail on multicore parallel processing as noted in Cory's book.

Transaction and Software Pipelines: Background (2)

- Cory Isaacson's book "Software Pipelines and SOA describes several techniques for bringing parallel computing to mainstream software development.". Unless otherwise noted, the remaining content for this section of the presentation will contain paraphrased excerpts from Cory's book.

- "a **pipeline** is a control mechanism that receives and performs delegated tasks, with the option of then delegating tasks in turn to other pipelines in the system as required."

- Note-water molecules normally run in only one direction through a pipe which is analogous to how asynchronous transactions behave.

- "You can group multiple pipelines into fully distributed, peer-to-peer pools; each pipeline processes a portion of an application or process. And because you can configure each pool to run on a specific local or remote server, the system can execute tasks anywhere on a network."

Transaction and Software Pipelines Principles (1)

InputRate = OutputRate

InputRate and OutputRate will always be the same. A software system can't accept more transactions than it can process.

InputRate must be <= ProcessRate

The InputRate must be reduced to accommodate the ProcessRate/capacity of all downstream components or processes. Otherwise backpressure can occur or a downstream component may fail.

OR ProcessRate must be increased to match the InputRate

All downstream components or processes must be able to accommodate the input rate of any upstream process or component. If the ProcessRate is insufficient, then additional pipelines (or other solutions) are needed to service the InputRate.

Friction and restriction limit the flow

Any restriction in a pipe limits the overall flow of fluid through the system and restricts and reduces both inflow and outflow. Restrictions are caused by crimps and bends in a pipe. In IT terms, each technology component in a pipeline must be restriction free.

Transaction and Software Pipelines Principles

Pipeline Distributor capacity must be >= InputRate

A pipeline distributor directs traffic coming from one pipe into 2 or more outbound pipes.
The distributor provides a way to add more backend capacity/pipelines to match the InputRate and can allow traffic to be selectively siphoned off and rerouted.
e.g. tee fitting or cross fitting with optional reducer fitting. In IT terms, think load balancing device/software.

Formula to Determine the Optimum Number of Downstream Pipelines

numberofpipelines = DistributorTPS (rate) / ProcessTPS (rate)
Add a fudge factor to account for future growth.
e.g. A pipeline distributor can process 2000 TPS and feeds into four pipelines that each process 500 TPS.
NOTE-Rightsize pipelines to avoid wasting resources and incurring unnecessary costs.

Buffer Area

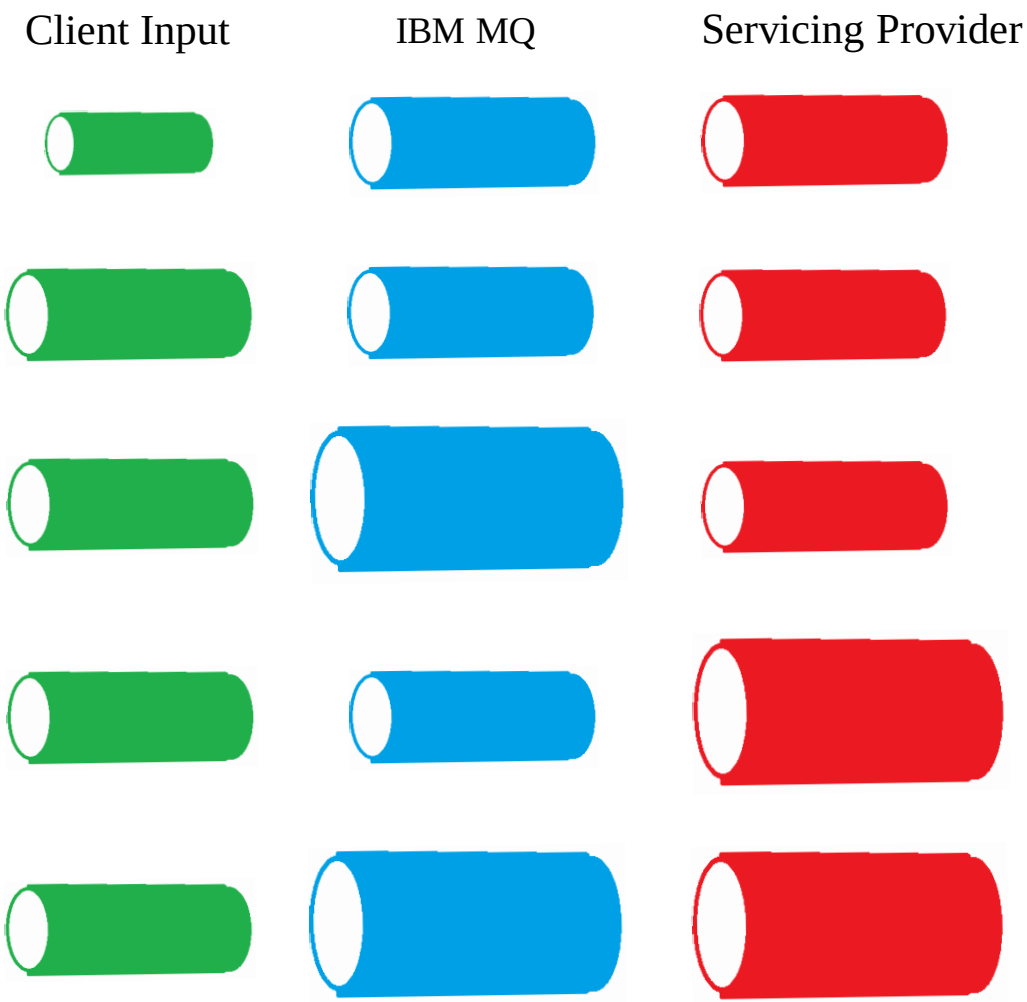
Plumbing systems can have overflow pipes or reservoirs that take the pressure off the pipeline and feed the overflow back into the pipeline when conditions permit. If there's no buffer in which transactions can queue up before they get to 'their destination', you can lose transactions. Note-MQ queues have buffer capabilities.

The Response Time and Throughput Factor

Note-As the response time goes down…the throughput rate (capacity) requirement$ normally go up.
If it takes a 1" pipe two hours to fill a swimming pool, then ideally, a 2" pipe will do it in one hour.
ie adding capacity. All components in a pipeline need to optimize their throughput (GIGO).

Transaction and Software Pipelines Principles

Figure: Compatible Traffic Pipelines

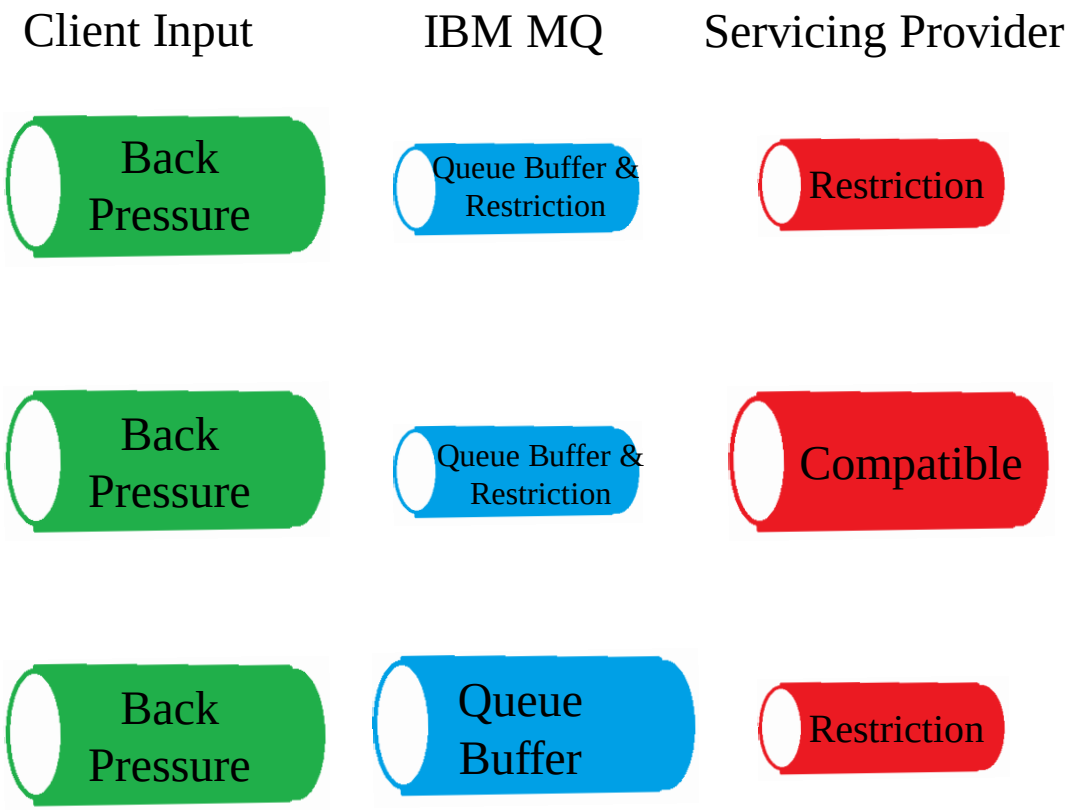


InputRate is <= ProcessRate

Source: Lee Wheaton@MQTC 2018

Transaction and Software Pipelines Principles

Figure: Mismatched Pipelines
InputRate is > ProcessRate



Source: Lee Wheaton@MQTC 2018

MQ Queue Manager and/or Servicing Provider may be swamped and suffer an outage*. MQ buffer shelters Servicing Provider. Client experiences increased response times or receives no responses. Too much back pressure and Client fails.

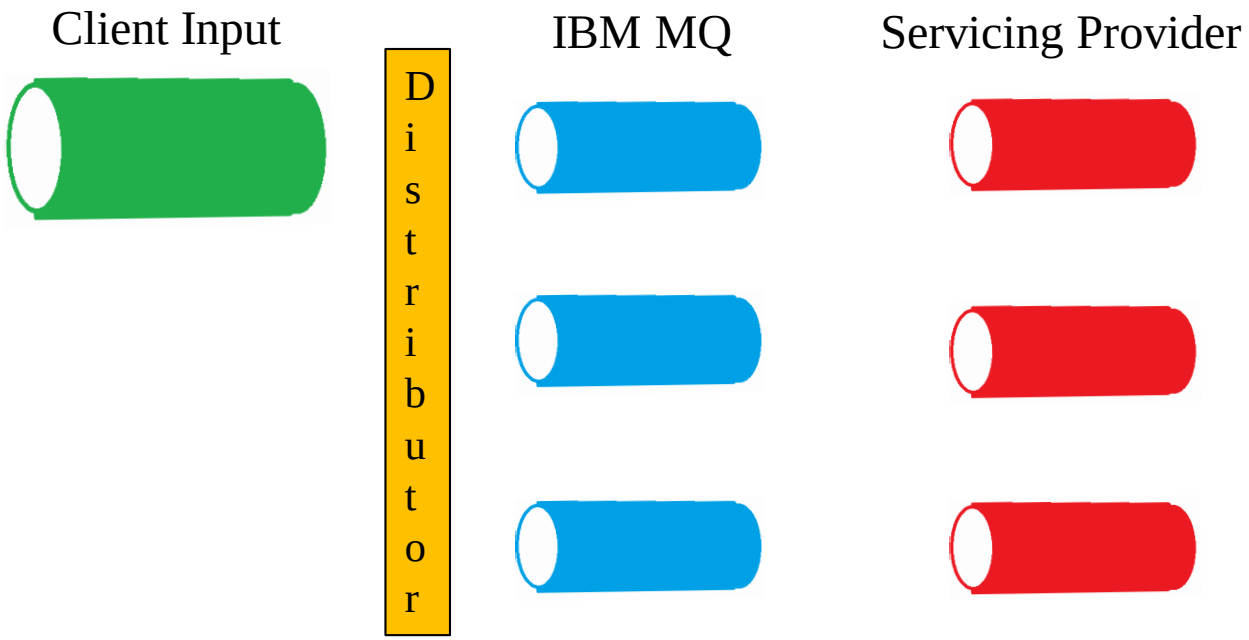
MQ Queue Manager either holds up or not*. If queue manager still viable, Client may experience increased response times at peak traffic times and back pressure.

Servicing Provider may be under duress at peak times, but MQ buffer coupled with task throttling can protect it. Client may experience increased response times.

***MQ can be configured to protect itself**

Transaction and Software Pipelines Principles

Figure: Increasing the Process Rate by Adding a Pipeline Distributor and Pipelines



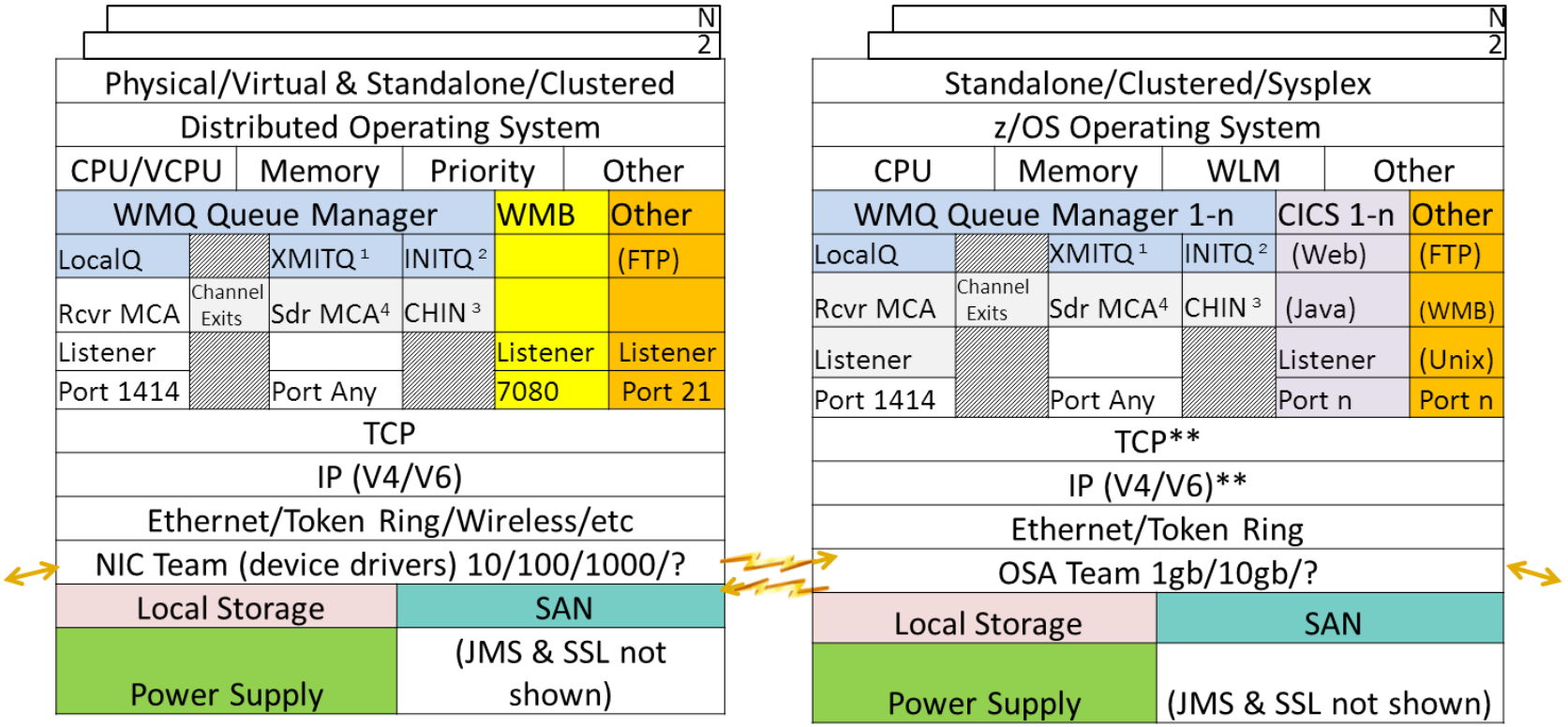
MQ Queue Manager and a Service Provider are still restrictions on traffic flow, but adding more load balanced instances or pipelines mitigates this issue. This design could be leveraged for parallel processing as well.

Note-Right size the number of pipelines you add to avoid having underutilized pipes.

Source: Lee Wheaton@MQTC 2018

IBM MQ as a Reactive System

Pipelines within Pipelines or Systems within Systems



* - Sender channel startup sequence: 1) message is written to the transmit queue, 2) which may result in a trigger message being put on the system.channel.initq, 3) the channel initiator supervisor picks up the trigger message, and 4) starts the Sender channel that was specified in the transmit queue trigger definition. The sender MCA services the messages on the transmit queue.

**- On z/OS, the TCP/IP protocol stack is a component of the z/OS Communications Server which is not shown in the diagram. Also of note is the Communications Storage Manager component which provides the I/O buffer area for TCP/IP traffic.

Source: Lee Wheaton@SHARE Pittsburgh 2014: What's Wrong with MQ?

The Reactive Manifesto

The Reactive Manifesto: Background

- The Reactive Manifesto is a blueprint for building Responsive, Resilient, Elastic and Message Driven systems and applications to meet current and future business requirements using time tested principles.
- The Reactive Manifesto was formulated in 2013 and refined in 2014 by Jonas Bonér, Dave Farley, Roland Kuhn, Martin Thompson and others.
- The Reactive Manifesto encapsulates and expands on pipeline principles to provide a more coherent architectural blueprint
- **Proactive**-anticipating an event or situation will occur and resolving (or mitigating) it in advance before it does occur
- **Reactive**-acting in response to an event or situation **real time** (when it occurs) rather than creating or controlling it.
- **Manifesto**-a written statement declaring publicly the intentions, motives, or views of its issuer
- The Reactive Manifesto can be found at https://www.reactivemanifesto.org/
- Additional Info: Kuhn with Hanafee & Allen: Reactive Design Patterns

The Reactive Manifesto: Premise (1)

Content from the Reactive Manifesto (used for the rest of this section):

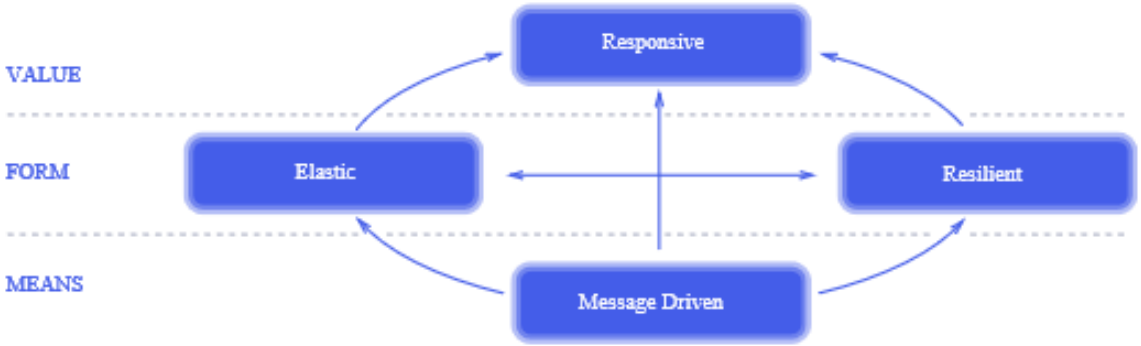
- "Organisations … are … discovering patterns for building software that look the same."
- "application requirements have changed dramatically in recent years...Today applications are deployed on everything from mobile devices to cloud-based clusters running thousands of multi-core processors."
- **"Users expect millisecond response times and 100% uptime."**
- "a coherent approach to systems architecture is needed" to meet demands
- **"we want systems that are Responsive, Resilient, Elastic and Message Driven. We call these Reactive Systems"**
- Systems built as Reactive Systems are:
  - more flexible, resilient, loosely-coupled and scalable
  - easier to develop and amenable to change
  - more tolerant of failure (and when failure does occur they meet it with elegance rather than disaster)
  - highly responsive (giving users effective interactive feedback & productivity)
  - better positioned to meet modern *(and future)* demands

The Reactive Manifesto: Premise (2)

- “Large systems are composed of smaller ones and therefore depend on the Reactive properties of their constituents. This means that Reactive Systems apply design principles so these properties apply at all levels of scale, making them composable.”

The Reactive Manifesto: Construct

The Four Principles of the Reactive Manifesto



Source: The Reactive Manifesto

The Reactive Manifesto: Responsive Principle

Responsive Attributes

- The system responds in a timely manner if at all possible
- Problems may be detected quickly and dealt with effectively
- Provides rapid and consistent response times
- Provides a consistent quality of service.

This consistent behaviour in turn simplifies error handling, builds end user confidence, and encourages further interaction.

The Reactive Manifesto: Resilient Principle

Resilient Attributes

- The system (service) stays responsive in the face of failure.
  Any system that is not resilient will be unresponsive after a failure.

- Resilience is achieved by:
  - ✓ Replication - Executing a component simultaneously in different places
    i.e. workload is distributed using multiple threads, processes, network nodes or computing centers
  - ✓ Containment - The client of a component is not burdened with handling the components's failures. Failures are contained within each component ensuring that parts of the system can fail and recover without compromising the system as a whole.
  - ✓ Isolation – Decoupling the sender and receiver by using asynchronous boundaries and communicating through message-passing
  - ✓ Delegation - Delegating a task asynchronously to another component which allows the delegating component to perform other processing (i.e. non-blocking)

- Recovery of each component is delegated to another (external) component and high-availability is ensured by replication where necessary.

The Reactive Manifesto: Elastic Principle

Elastic Attributes

- Scalability - the system needs to be scalable to allow it to benefit from the dynamic addition, or removal, of resources at runtime.
- Elasticity - the throughput of a system scales up or down automatically to meet varying demand as a resource is proportionally added or removed.
- Elasticity builds upon scalability and expands on it by adding the notion of automatic resource management. Reactive Systems can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs.
- The system stays responsive under varying workload. The system design has no contention points or central bottlenecks which results in the ability to shard or replicate components and distribute inputs among them.
- Reactive Systems support predictive, as well as Reactive, scaling algorithms by providing relevant live performance measures. They achieve elasticity in a cost-effective way on commodity hardware and software platforms.

The Reactive Manifesto: Message Driven Principle

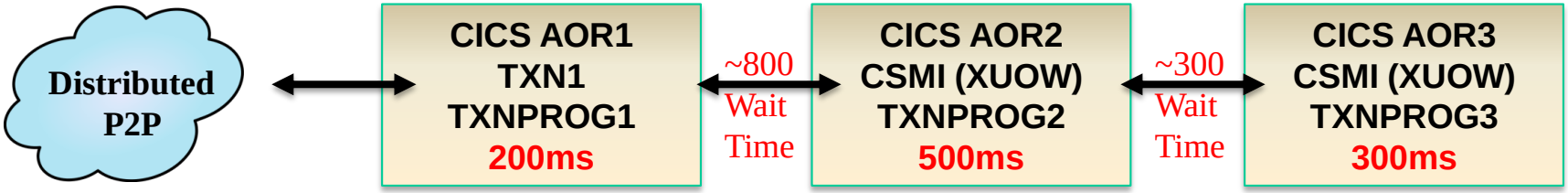
Message Driven Attributes

- Reactive Systems establish boundaries between components to ensure loose coupling, isolation and location transparency.
- Reactive Systems depend on asynchronous message-passing to establish these boundaries for them.
  - A message is an item of data that is sent to a specific destination.
  - A message can contain an encoded event as its payload.
  - Employing explicit message-passing enables load management, elasticity, and flow control by shaping and monitoring the message queues in the system and applying back-pressure when necessary [to protect downstream components from failing due to an unsustainable load].
  - Asynchronous message-passing is a Non-blocking protocol that allows recipients to only consume resources while they're active resulting in less system overhead [across the board].
- A boundary provides the means to delegate failures as messages.
  - Location transparent messaging makes it possible for the management of failure to work with the same constructs and semantics across a cluster or within a single host.

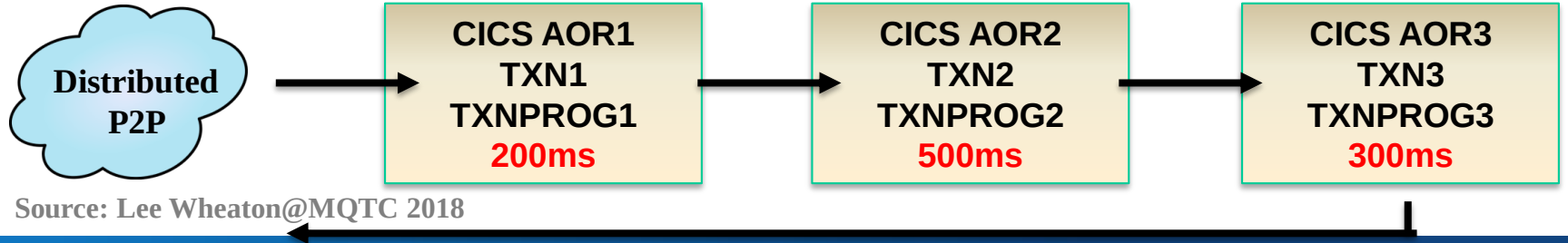
The Reactive Manifesto: Message Driven Principle

Message Driven Attributes: Blocking versus Non-Blocking

Example 1: Shows blocking of further processing (i.e. wait state) across the board using a synchronous-like protocol until control is returned back to each component. e.g. Distributed, AOR1 and AOR2 are suspended until AOR3 completes it's work. Several mirror tasks and record locks are in play and TXN1 and CSMI tie up processing slots that count against the CICS regions Max Tasks setting. More pressure is brought to bear on the front-end system if responses are not timely enough to release connections needed for new connections. Abend affects 3 AOR



Example 2: Shows non-blocking or async protocol resulting in significantly less overhead. Assumption that Prog1/2/3 can stand alone. Guaranteed delivery comes into play.



Source: Lee Wheaton@MQTC 2018

Note that TXN1, TXN2 and TXN3 could also be run in parallel since they are all standalone.

The Reactive Manifesto: Proposed Principle?

Guaranteed Delivery Principle (1)

- Guaranteed Delivery of messages is the ideal goal, especially for messages critical to the business, it's customers and regulators.
- What's the sense in creating the best proactive/reactive systems if we can't deliver the message with integrity to its correct destination?
  - Can you afford to lose 1% of your transactions and/or not know where they went?
  - Can you lose one message worth over a million dollars?
  - A time sensitive message is delayed and goes stale. Any consequences/penalties?
  - Can you afford to lose the confidence of your customers?
- Critical messages should be trackable and never lost. However, if a message is undeliverable, then one Reactive approach is to have a re-playable Repository to accept such messages like IBM MQ's DLQ.
- **'Assured Delivery'** comes into play due to lack of control over the people factor, technology limitations/design and product upgrades/compatibility.
  e.g. a developer codes an MQGET without syncpoint control and the production program version gets a message and then abends - losing the message

The Reactive Manifesto: Proposed Principle?

Guaranteed Delivery Principle (2)

Attributes:

- A Message is delivered once and only once
- Messages are delivered within a timely manner (e.g. SLA, shop standard)
- Developers persist critical messages, use syncpoint control and react correctly to abnormal return codes
- Authentication of the sender, the receiver and other nodes as needed
- Transaction and Data Integrity is maintained across the pipeline (e.g. XA)
- Secured Pipelines to prevent Intrusions/Hacks
- Undeliverable messages should not be lost and are traceable (i.e. audit logs, Event messages). A replayable Repository (e.g. Transmit Queue, Dead Letter Queue) should be the storage medium for such messages.
- Messages arriving at their destination queues are picked up for processing by their service provider in an elastic manner. (e.g. triggering)
- Messages can be verified as reaching their destination via Responses, Event Messages, follow-up Inquiry transaction, Audit Logs or other means
- Message Store & Forward capabilities

IBM MQ

as a

Reactive System

Intro-Overview

This section illustrates the features/functions of IBM MQ that align with the principles for Pipelines and the Reactive Manifesto.

IBM MQ: The Universal Messaging Backbone

The Power of MQ:

Assured delivery

MQ replication

Async capability (non-blocking)

Scalability, Resiliency and Elasticity

Parallel processing

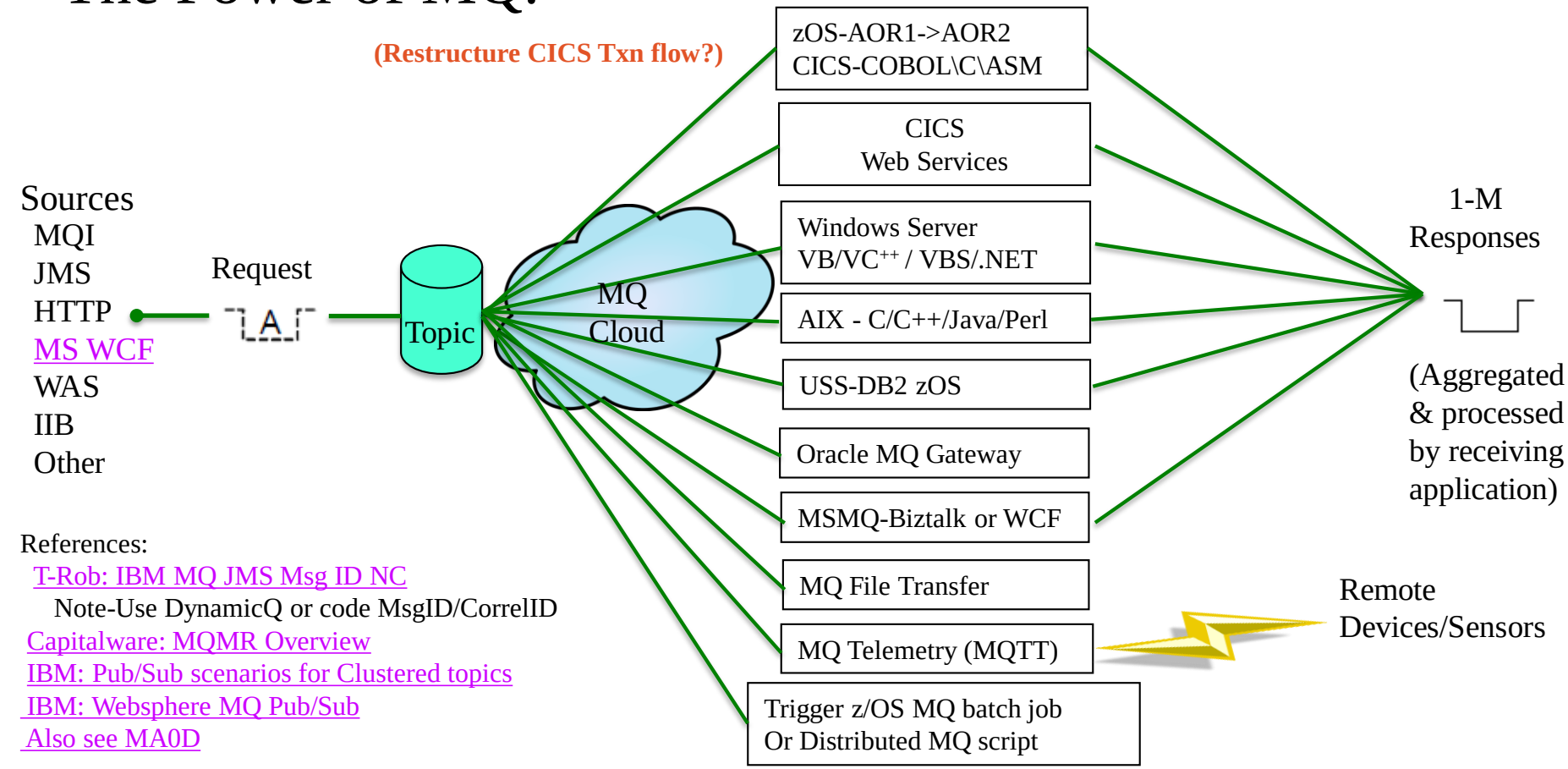
Runs on 80+ operating platforms

Next Slide: One MQ message broadcast to multiple processing locations

IBM: How to generate duplicate MQ messages from one message

IBM MQ: The Universal Messaging Backbone

The Power of MQ:

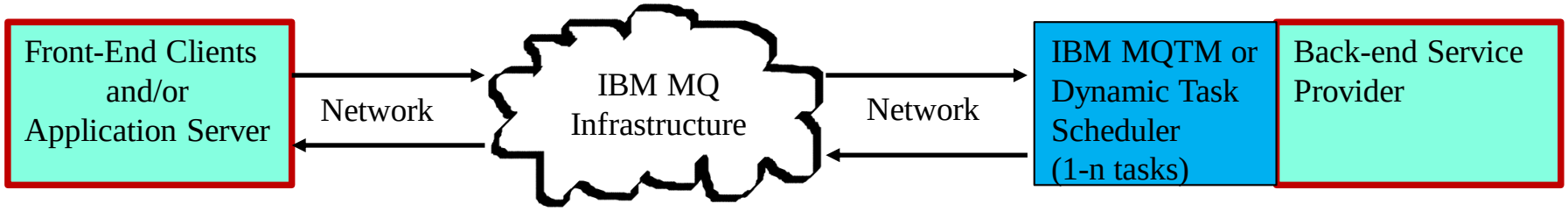


References:

- T-Rob: IBM MQ JMS Msg ID NC
  - Note-Use DynamicQ or code MsgID/CorrelID
- Capitalware: MQMR Overview
- IBM: Pub/Sub scenarios for Clustered topics
- IBM: Websphere MQ Pub/Sub
- Also see MA0D

Source: Lee Wheaton@SHARE San Antonio 2016: Thoughts on MQ Architecture & Design

IBM MQ: Responsive Features (1)



- Response times within the IBM MQ infrastructure are normally static if given sufficient MQ resources and stable network bandwidth for peak traffic times.
- However, MQ response time will differ to a degree depending on the message size. More network packets are needed as the message size increases.
  - Using IBM MQ's message compression feature may improve network response time
- Dynamically increase/decrease broker/triggered tasks to optimally service the MQ queues and reduce unnecessary resource consumption.
- Normally the bulk of an end users response time will be due to the amount of work performed in the front-end and back-end applications (outside of MQ). An account history lookup transaction will take longer to process than a balance inquiry and require more network packets.
- See IBM MQ Family - Performance Reports

IBM MQ: Responsive Features (2)

- Overall end user response time (for the whole pipeline) can be monitored on the client side.
- End to end MQ response times (within the MQ infrastructure) can be monitored with vendor monitors or writing your own MQ client program to periodically send roundtrip MQ messages and recording the elapsed time.
- MQ monitors can detect queue buildups, channel issues/latency, message aging and do other MQ health checks as well as notify administrators of issues and/or perform corrective actions.
- MQI calls receive back a condition code and a reason code that the developer can use to determine the appropriate action to take next.
- MQ Client connection pooling via JMS or netscaler devices (sticky setting) reduces new connection overhead which improves response time
- Network QOS rules can give priority to MQ traffic over the network
- MQMD priority can be set on MQ messages to give a specific message or a class of messages processing priority (on MQGET's) over other messages within the same queue assuming Msgdlvsq is set to Priority.
- Performance gains can be achieved by using non-persistent messages instead of persistent messages but developers need to mitigate the risk of lost messages.

IBM MQ: Responsive Features (3)

Responsive Principle and Planned Latency (1)

Planned latency - dynamically controlled throttling of electronic traffic while meeting SLA or other response time objectives.

Goals:

1) support end user think time to reduce errors
2) protect downstream components (like a dam on a lake)
3) smooth out peak and valley traffic patterns (i.e. steady state)
   - minimizes maxed out components under normal conditions
   - better component utilization due to a steady stream of traffic
   - reduces resource contention/collisions
4) reduce costs by using commodity hardware

Caveat: Certain transactions involving stock trades, wire transfers, time-sensitive or other content will need the lowest latency possible. **(ie no user interaction)**

MQ & CICS throttling for planned latency will be reviewed in later sections.

IBM MQ: Responsive Features (4)

Responsive Principle and Planned Latency (2)

Paraphrased excerpts from “Response Time and Display Rate in Human Performance with Computers “ by Ben Shneiderman:

- End user productivity increases as response time decreases
- Error rates increase with too short or too long a response time
- User think time is needed between commands to read/comprehend the response content, make sound decisions based on the content, determine the next steps and perform accurate data entry (typing) before submitting the next command.
- Working too quickly or user fatigue can result in stress/mistakes or jeopardize human life (e.g. air traffic controllers or medical systems) *Carpal Tunnel? Eyes?*
- The amount and complexity of response content presented to the end user may require additional think time.
  - *My question: will the end user read all of the response content or was system resources/ bandwidth wasted to generate content that won’t be read (and increasing response time)*
- End users become accustomed to a response time and a +/- 50% variance in the mean response time will not impact their performance. e.g. 1.5-2.5 sec range for 2”

Source: Ben Shneiderman@acm.org: Response Time & Display Rate in Human Performance with Computers

IBM MQ: Responsive Features (5)

Responsive Principle and Planned Latency (3)

- The following articles discuss using Back Pressure as a form of planned latency when the input rate is greater than the processing rate:

  http://blog.clear-measure.com/backpressure-in-message-based-systems
  https://dzone.com/articles/applying-back-pressure-when

IBM MQ: Resilient Features (1)

The following major features or redundancies in IBM MQ allow it to stay responsive or protect itself in the face of failure:

- IBM MQ runs 24x7 under virtualized hosts and z/OS Sysplex and in the Cloud
- Multi-Instance Queue Manager or Power HA
- IBM MQ Queue Manager Clustering
- MQ z/OS shared queues and shared channels
- Primary & Secondary log files and log triplewrite integrity
- IBM MQ support for XA external syncpoint coordination and 2 phased commit
- MQ Messages can be load balanced via network devices and/or software
- Sender Channel *CONNAME* and MQ Client Channel Table allow multiple connection entries for auto-reconnects on network drops
- MQ and TCP *KEEPALIVE* setting (i.e. cleans up orphaned connections)
- Queue Manager *Maxhands* parameter controls the number of objects that a single connection can have open at the same time
- Queue Manager *Maxumsgs* parameter limits the number of uncommitted messages under any one sync point
- Qmgr *Maxchl* & *Actchl* parms control max number of current/active channels allowed
- MQ SVRCONN Channel *Maxinst* and *Maxinstc* sets the max client connections allowed to start for this channel and the max connections allowed for a single client
- Reference: IBM Knowledge Center: Automatic client reconnection

IBM MQ: Resilient Features (2)

- MQ Queue *Maxdepth* and *Maxmsgl* help protect the queue manager or meet certain application requirements (e.g. mitigate application buffer overflow)
- MQ OAM (and/or IPSEC) policies allow in only channel traffic from locations defined to the OAM ruleset.
- IBM MQ has a transmit queue to temporarily house messages in transit that are undeliverable due to a network/other outage. Once the outage is resolved, the housed messages continue their trip on to their destination. i.e. Store & Forward
- IBM MQ has a Dead Letter Queue to temporarily or permanently house undeliverable messages. A DLQ handler can be run to forward DLQ messages.
- IBM Tivoli or some 3rd party MQ Monitoring tools can monitor MQ's health and take action as needed

  e.g. Turn on a trigger flag for a local queue at a specified time or event
  Replay messages in a dead letter queue to complete their processing
  Start/stop channels at a specified time or event
  Trigger predefined remedial or administrative scripts at a specified time/event
  Alert administrators to MQ health anomalies
- IBM MQ can pass back to it's clients a reason code which the client can react to
- IBM MQ has a number of exit points that can be used for customized solutions
- *Other features not listed here*

IBM MQ: Elastic Features (1)

- Some companies are processing billions of MQ messages each day.
- IBM MQ as a system can scale horizontally. MQ programming has multi-threading capabilities and multicore possibilities to scale vertically to a degree
- MQ in the Cloud is capable of dynamically adding/removing MQ resources. Outside the Cloud, new MQ resources normally need to be preplanned and budgeted. Then MQ queue managers can be built as spares or held in reserve until varied online to add capacity. When traffic goes down, the queue manager is varied offline. (Check your MQ licensing). Note-test queue managers could be refitted to service production using automation scripts.

  Caveat: If you have purchased your own hardware, software and licensing to add MQ capacity, it could be argued to run at your full hardware/software capacity all the time to be cost effective and forgo auto resource management overhead (eg yo-yo effect) and risk (ie resource may not come up)
- Capacity planning is critical to supporting MQ elasticity and minimizing infrastructure costs.
- MQ clustering or MQ parallel sysplex allows the service to continue while you take queue managers offline for maintenance/triage or bring online to add capacity.

IBM MQ: Elastic Features (2)

- The IBM MQ pub/sub feature can be used to broadcast Events to multiple components within a pipeline for service orchestration or to have them act in concert
- MQ throughput can be scaled dynamically for elasticity by controlling the number of servicing programs triggered for each queue. The number of queue servicing programs are increased (up to a set limit) as the queue depth increases and go down as the queue is drained. The CICS AOR is given as much MQ traffic as it can handle. Ideally, multiple AOR's are used with CICSPlex WLM and TCLASS setting. More details on dynamic throttling are provided later in this document.
- Some ways for MQ to create backpressure on the front-end (FE)

  A) Using MQ dynamic triggering as noted above will result in the backend not being overrun and still allow the front-end to keep sending messages. However, in peak times, it is possible that queue depths will grow resulting in increasing response times on the front-end until MQGet with wait times out.

  B) A (homegrown) program detects the downstream is being overrun (based on qdepths) and sends an Event msg to front-end program to throttle their traffic

  C) A (homegrown) monitor detects the downstream is being overrun (based on qdepths) & issues commands to take offline 1 or more MQ servers in the pool. While this option will work, it's not recommended as it creates havoc on FE.

IBM MQ: Elastic Features (3)

- With it's queues, MQ can buffer the backend application from sudden surges in frontend traffic, help protect it's backend applications from being overrun with transactions and turn backend traffic patterns from peaks and valleys into hills and dales.
- IBM MQ messages can be load balanced. Some SLB switches can detect servers under stress and reroute new traffic to other servers in the team.
- The IBM MQ system stays responsive under varying workload
- IBM MQ can run on commodity hardware
- IBM MQ provides some relevant live performance measures like msgage, curdepth, lgetdate, lgettime, lputdate, lputtime, qtime, nettime, lstmsgda, lstmsgti, chstada, chstati, curseqno, status, msgs, etc.
- However, MQ monitors can also provide relevant live MQ performance measures as noted above and capture this information into histograms which can be leveraged for problem resolution and as input for capacity planning forecasting.

IBM MQ: Message Driven Features (1)

- IBM MQ is Message-Oriented Middleware.
  Per Gartner, in 2015, IBM had a 75% market share in Message Oriented Middleware
- Placing a MQ queue manager between two applications allows their communications to be asynchronous and loosely coupled.
- The location of MQ resources is normally transparent to the user or app program.
- MQI calls to a queue manager are synchronous to facilitate transaction/data integrity and provide status codes to the client, but MQ developers have ways to structure their MQI calls to be non-blocking as noted later in this document.
- An MQ message can contain an Event
- MQ Messages can be load balanced
- IBM MQ can reroute message traffic or create multiple pipeline paths to service a message. e.g. the messages going to a backend server having problems can be redirected to another backend server in good standing.
- IBM MQ has publish/subscribe messaging that decouples the provider from the consumers of that information. Pub/Sub can be used to replicate messages.

IBM MQ: Message Driven Features (2)

- IBM & 3rd party MQ Monitors can do ongoing MQ healthchecks of your whole MQ infrastructure and admins can be notified of any failures/anomalies. Some MQ monitors have the ability to take corrective action for certain problems.

- IBM MQ returns back a condition and reason code after each call, and report messages if requested, to notify the requestor/developer on the status of their request.

- Failures that IBM MQ detects can show up in MQ event queues and/or syslog/event logs where automation monitoring programs (e.g. Big Brother, OPS/MVS, SNMP) can parse for MQ error eyecatchers and take action or notify MQ Administrators and/or Operations personnel.

IBM MQ: Guaranteed Delivery Features

Guaranteed Delivery should be an 'ideal' goal to always strive for.

- However, Change is ongoing for software, hardware and educational development as well as maintaining the compatibilities that go with it.
- Each vendor in a pipeline can only warrant their products and not end-to-end guaranteed delivery which they don't have total control over. Vendors also can't control how their products are being used (i.e. people factor).
- Companies around the world have Help Desks and triage teams for a reason.
- **'Assured Delivery'** then is the highest level of reliable messaging that can be achieved within the constraints noted above.
- IBM MQ meets the attributes they control that are listed on the Guaranteed Delivery Principle (2) slide.

Per Leif Davidsen, **"Half of the value of IBM MQ is not just the assured once and once only delivery but that your business has the visibility that the message was delivery successfully or not."**

Source: Leif Davidsen@wordpress.com: Power is nothing without control – IBM MQ V9.0.1
Reference: **Leif Davidsen: What is IBM MQ and why do you need it?**

IBM MQ

Reactive Programming

IBM MQ Reactive Programming (1)

Reactive Programming is an asynchronous non-blocking development model using data streams, a task-based concurrency model and employs non-blocking I/O. Data stream content can contain events, server requests, messaging, and even values. Advanced reactive programming may leverage multicore programming and parallelism.

- Reactive programming is *event-driven.* Reactive systems are *message-driven.*
- Systems and application RP bridge the gaps/deficiencies in reactive systems and allow coding of custom reactive solutions especially events.
- RP can be used to dynamically apply back pressure when a pipeline is maxed out
- RP "supports decomposing the problem into multiple discrete steps where each can be executed in an asynchronous and non-blocking fashion, and then be composed to produce a workflow—possibly unbounded in its inputs or outputs."
- "The primary benefits of reactive programming are: increased utilization of computing resources on multicore and multi-CPU hardware; and increased performance by reducing serialization points."
- "A secondary benefit is one of developer productivity as…it typically removes the need for explicit coordination between active components."

Source: Bonér & Klang@oreilly.com: Reactive programming vs. Reactive systems

IBM MQ Reactive Programming (2)

IBM MQ is asynchronous by nature. However, MQ developers need to code their MQ calls so they are non-blocking where feasible.

- Programs needing to serialize their code logic for integrity or having certain dependencies may still need to be synchronous and of a blocking nature.
- Ideally, MQ developers can decompose their program logic into multiple discrete units of work (UOW) so each UOW can be isolated and sent as a fire & forget message, in a non-blocking manner, and processed downstream concurrently/in parallel with the other UOWs.

  OR

  Using Pub/Sub, one message is replicated to multiple processing destinations so that each one process an isolated & discrete unit of work (UOW) in parallel
- If a response message is needed, each UOW can use MQMD ReplyToQ and *correlid/msgid* to (asynchronously) put the message back to the sender response queue where the sender application can reassemble the responses as a coherent whole. Note-Other solutions may be possible.
- MQPUT1 is closest to async non-blocking mode and the Reactive Manifesto.
- Do MQPUT, do other non-MQ work then do MQGET at end (Not quite compliant)
- Do MQPUT1 in one thread/component and MQGET response in different thread
- Use Event and Report Messages in lieu of Response messages

IBM MQ Reactive Programming (3)

Throttling MQ Traffic and still meet SLAs or Other Time Objectives

Presented below are several examples on how to throttle MQ traffic. The goal is to **1)** achieve elasticity, **2)** cap the number of concurrent queue servicing programs to protect the servicing provider (e.g. CICS AOR) while **3)** keeping the queue depth at a minimum and still meet SLAs. CICS TCLASS can work in concert with MQ cap to smooth out MQ traffic patterns. Note-MQ Process Def can contain the capping info.

- Local Queue is defined with Trigger First. Avoid Trigger Every as generates much more trigger messages on the InitQ and CKTI starts a CICS task for each trigger.
- The first triggered/master program does a MQINQ on it's qdepth and, based on a homegrown algorithm, kicks off as many servicing programs as needed or until it hits the Process cap. The master program periodically monitors the qdepth and ends when the queue is drained. Each servicing program continually processes messages off their assigned queue and end their program when they get a 2033

AND/OR

- A standalone MQ monitoring program, like CKTI, can periodically MQINQ the qdepths of the local queues in its purview and kick off as many servicing programs as needed or until the Process cap is reached. Each queue's Process object definitions can be pre-loaded at program start-up. Avoid 1 CICS task for each msg
- Create a hybrid solution from the previous two solutions noted above

IBM MQ Reactive Programming (4)

IBM MQ supports Events, Event Queues and Report Messages

- IBM MQ has queue manager, performance, channel, command and configuration event queues containing informational, warnings or error events related to these entities. e.g. channel starting up, queue full.
  - Events are standard IBM MQ messages containing a message descriptor and message data. **Developers can create their own events and send them off as MQ messages.**

  Reference: IBM Knowledge Center: Sample program to monitor instrumentation events

- MQ developers can request or generate MQ report messages that inform the calling program on the status of their MQ request. The calling program can then determine if any next steps are needed or provide this feedback to their end user. These report messages can be especially useful for fire and forget mode.

  1) The queue manager generates a Confirm on Arrival (COA) and Confirm on Delivery(COD)
  2) The developer generates a positive action notification (PAN) or negative action notification (NAN) report message to denote if the request was successfully processed or not.

  Reference: IBM Knowledge Center: Instrumentation Events

Developers can issue a PCF call (or the much less powerful MQINQ call) to query the attributes and runtime status of MQ objects and act on the results.
e.g. Current queue depth, Queue open for Input Count, Queue open for Output count, process attributes, Queue Manager Name, DLQ name, etc. PCF calls are best for Reactive Programming.

IBM MQ Reactive Programming (5)

- **After each MQ call, the developer's program checks a MQ condition code and reason code to determine if there were any problems & reacts to them promptly.**
  - MQ systems and utility programs also provide error codes.
- **MQ developer's must avoid coding affinities to use specific servers or queue managers and their program code should be able to run concurrently on multiple queue managers/servers within a MQ cluster or Sysplex.**
  - Avoid hard coded MQ parms that take control away from MQ Administrators
- **Specify fail_if_quiescing on your MQ calls. Don't hold QManagers hostage.**
- **Persist Messages and do MQGET under syncpoint control for assured delivery**
- **The developer is responsible for isolating/compartmentalizing program code**
- **All programs in a pipeline need to avoid creating program loops, resource contention or other bottlenecks that impact the whole pipeline**
- **Developers should leverage previously successful messaging patterns for new business solutions and for code consistency**

IBM MQ Reactive Programming (6)

Reference: Kevin Webber@redelastic.com: What is Reactive Programming?

Clement Escoffier@dzone.com: 5 Things to Know About Reactive Programming

Andre Staltz@github.com: The introduction to Reactive Programming you've been missing

Konrad Malawski@oreilly.com: Why Reactive? (free ebook)

Summary

In this session, we first reviewed:

- Transaction & Software Pipeline Principles
- The Reactive Manifesto

Then reviewed:

- IBM MQ as a Reactive System
- IBM MQ Reactive Programming

Hopefully, you agree that IBM MQ is a fully Reactive System and then some...

Questions & Answers

IBM MQ as a Reactive System

Thank you for attending this session!