

2021 Holiday Peak Readiness

Peak Considerations

Expert Panel Discussion

IBM Order Management SaaS

- *Have a question?*
- *Care to share your experiences or resolutions to similar scenarios?*

Click **Participants** and then click **Raise hand** 🙋 next to your name.

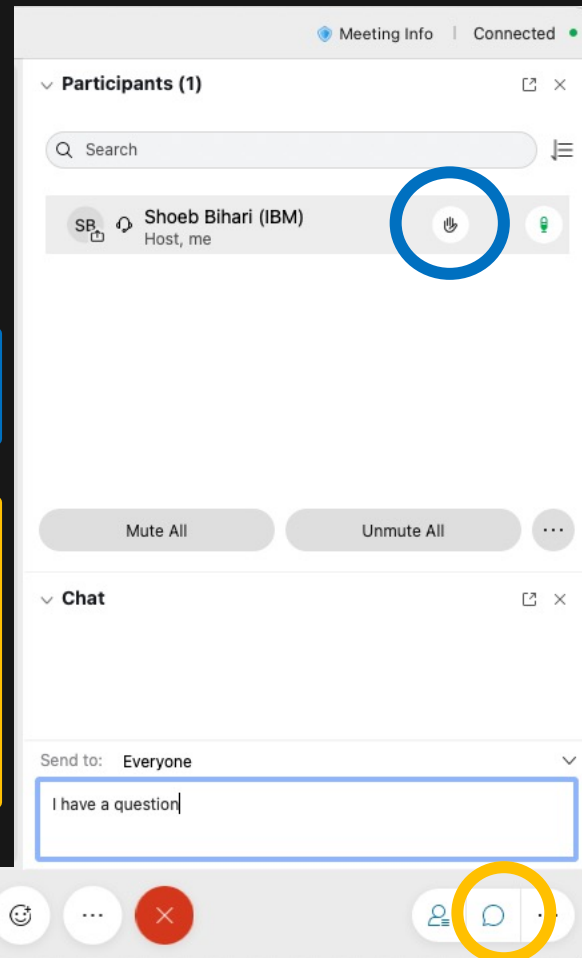
To lower your hand, click **Lower hand** 🙋 next to your name in the Participants panel.

To send a chat message:

- 1 Open the Chat panel from the link in the lower right of the meeting window:



- 2 In the **Send to** or **To** drop-down list, select the recipient of the message.
- 3 Enter your message in the chat text box, then press **Enter** on your keyboard.



Meeting Info | Connected

Participants (1)

Search

SB, Shoeb Bihari (IBM)
Host, me

Mute All Unmute All

Chat

Send to: Everyone

I have a question

Mute Start video Share Record

Speakers



Mike Callaghan
Program Director –
Sterling Support



Chris Burgess
Supervisor, Americas Support Experience
Team (SET)
AI Applications Cloud and Cognitive Software



Shoeb Bihari
Technical Lead –
IBM Sterling Order Management
Solutions



Senthil Ponnusamy
Monitoring Lead
IBM Sterling Order Management
Solutions



Paresh Vinaykya
Executive Technical Account Manager



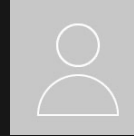
Bobby Thomas
Software Performance Architect – Order
Management on Cloud



Vijaya (Viji) Bashyam
Senior Architect – IBM Sterling Order Management
Solutions



Vishal Arora
Software Developer– IBM Sterling Order
Management Solutions



Sarat Devineni
SRE –
IBM Sterling Order Management
Solutions



Amar Jyothi Annamalai
Performance Architect –
IBM Sterling Inventory Visibility

*Our **IBM Sterling Support** mission is to partner together to proactively go through some of the key challenges, considerations, and best practices to help ensure a successful peak season powered by the IBM Order Management SaaS platform.*

! Be sure to catch our Jul 2021 Holiday Peak Readiness kick-off webcast: [Slides](#) / [Recording](#)

Next generation

Legacy

Discussion Topics



Let's discuss!!

1

Sourcing and Scheduling Considerations

- Distribution Group
- Scheduling Rules
- Smart Sourcing
- Capacity Cache

2

Considerations for Web Store, and Peak Testing

- Common Scenarios
- Best Practices
- Testing
- Customization

3

OMoC Self-Service Tool Demo

- Performance Dashboards
- Graylog

4

SLAs for BOPIS & Curb-Side Pick up Orders

- Optimal flow
- Considerations

Sourcing & Scheduling Considerations

Sourcing is the process of determining from which node or supplier a product should be shipped. A sourcing rule can be created by specifying one or more of the following key parameters:

- Item Classifications or Item ID
- Geographical region of the ship-to location or ship-to node
- Minimum available capacity
- Fulfillment type
- Seller organization
- Sourcing criteria

For each sourcing rule, you can then specify a sequence of node or distribution group to be used for sourcing the product, and performance of optimization logic depends on the complexity this setup.

What are the optimal configuration to maximize the performance?

- Number of nodes qualifying from sourcing should be as minimized by region, proximity, etc.
- If all nodes are qualifying, Capacity availability constraint should be avoided, or capacity cache should be used
- Use of Externally defined sourcing Vs Sourcing Correction
- If reservation node can be considered as final ship node then it should be passed on order line. This avoids schedule order to consider sourcing again.
- Order profile for pre-prod should be very similar to what is expected in production with regards to DC Ship, SFS, Pick up.

Distribution Group

Optimal Size

Sourcing rule region hierarchy

Priority of nodes when using multiple sequences

Using sequence of sourcing and consideration around high availability.

Scheduling Rules

Use Geography?

Priority of nodes

Distance of nodes from the ship-to location

Date when the delivery can be made

Number of resultant shipments

Cost-based scheduling

Legacy

Smart Sourcing

Reading and processing inventory for nodes that do not contain inventory is costly.

To improve performance, smart sourcing can be used to dynamically determine the nodes to consider for sourcing product items. How does this really work and what benefits can we expect for inventory lookup promising APIs?



Smart Sourcing

Capacity Cache

How to address contention around resource pool tables? (Ex. YFS_RES_POOL_CAPCTY_CONSMP TN)

When to use capacity cache?

How does asynchronous capacity calculation feature work?

Can it lead to over allocation?



Capacity Cache

Scenarios worth considering for Capacity ...

How to optimize node capacity feature for optimal performance?



Real Scenarios (Real impact...)

Business Objective

- ✓ Node has no individual capacity limit based on the delivery method [Ship and Pick]
- ✓ Node cannot handle any extra orders above the allocated capacity

Design Consideration

As per the business requirement, system should not send any orders beyond the allocated capacity. To avoid over allocation, set the following property **yfs.nodcapacity.lock** as Y and **yfs.nodcapacity.threshold** to make sure system will lock the capacity only when it is less than the set threshold.

Calculate the capacity asynchronously by using a time-triggered agent, this agent calculates the capacity upfront and stores in the **YFS_CAPACITY_AVAILABILITY** table from where the data can be read during promising calls.

- `yfs.nodcapacity.ignoreCacheForLowAvailability`: If set to true, the application reads capacity from the database ignoring the cached availability if it is below the defined threshold. This property works in conjunction with the Node capacity locking properties. **Low capacity value will be determined based on threshold setup.**
- `yfs.capacityAvailablity.ignoreCacheForUpdateMode`: If set to true, promising APIs requiring capacity availability and intending to update, e.g. `scheduleOrder` API will read capacity from the database ignoring the cached availability.

Related Properties:

- `yfs.nodcapacity.lock`
- `yfs.nodcapacity.threshold`
- `yfs.nodcapacity.ignoreCacheForLowAvailability`
- `yfs.capacityAvailablity.ignoreCacheForUpdateMode`
- `yfs.useNodeLocaleTimeForCapacityCheck`
- `yfs.persitCapacityAdjustments`



Considerations for Web Store, and Peak Testing

Regular store operations can lead to performance issues if the underlying API's input or template is not validated to avoid excessive contention on the DB side or resource issues on the application server-side.

Following best practices and recommendations for API input, template, customizations, etc., can help to eliminate some of these situations.

How do we detect and avoid these performance issues?

- *Common Scenarios*
- *Best Practices*
- *Testing*
- *Customization*

Common Scenarios

- getShipmentList api is invoked without 'SelectMethod="NO_LOCK"' parameter
- Not properly trimmed API input templates
- Using pagination for custom store UI

Best Practices

- Importance of testing
- Regularly purge exception, audit and transactional tables
- Optimize API output templates
- Close shipments & orders

Testing

- Identify all the activities performed by store associates and quantify them
- Record the browser session (script) for each activity using JMeter to identify all requests hitting the backend server
- Update the scripts to parameterized requests
- Execute the load scenarios with concurrent users

Customization

- Heavy queries and performance issues
- Controls on notification polling, where UI keeps refreshing/polling notifications even if user is inactive

OMoC Self-Service Tool

IBM OMoC is focused on providing client/SI necessary tooling to understand key aspects of the service

- Dashboards are available to help proactively monitor application usage and performance ([KC link](#))
- ***NOTE:*** Monitoring dashboards are currently enabled only for the Production & Pre-Production environment.

The following types of dashboards are available:

- Server Resource Utilization ([KC link](#))
- Agent and Integration Server Performance
- Database metrics
- Application Server Performance
- API Performance
- Business Performance
- JMS Metrics
- Agent and Integration Servers JVM Metrics
- Application Server JVM Metrics
- Refer to webcast replay for more details! (*doesn't not cover new dashboards*).

[IBM Self Service tool\(SST\) for Order Management](#)

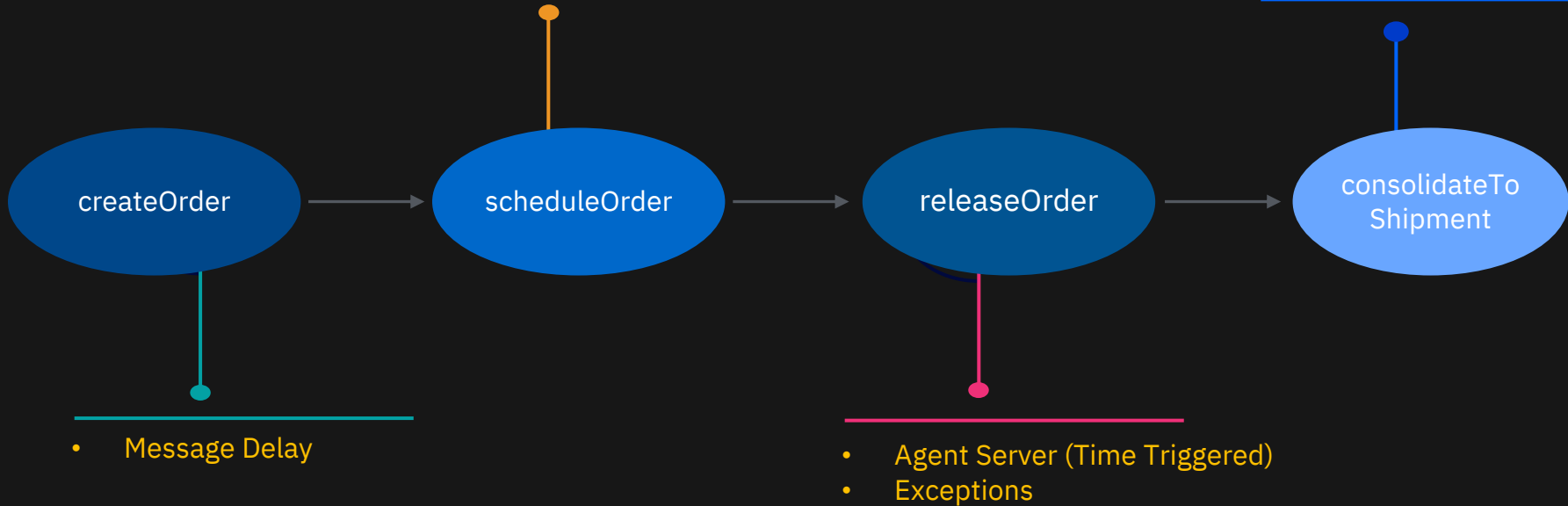
DEMO

SLAs for BOPIS & Curb-Side Pick up Orders



- Agent Server (Time Triggered)
- Backordered delay
- Exception (Pending payment authorization, etc.) delay
- Hold (REMORSE / FRAUD CHECK, etc.)
- EarliestScheduleDate

- Agent Server (Time Triggered)
- Exception



IBM Sterling

Sourcing Rules

Sourcing : Which Rule and When?

Products Being Shipped

An order line with a product that has no ShipNode specified will use sourcing rules for Products Being Shipped

Products Being Delivered

During a Work Order creation if no NodeKey is specified on Work Order Products Being Delivered sourcing rules will be used to stamp a Node on order line and Work Order

Provided Services

If Work Order does not include the product:

Sourcing Rules for Provided Services will be used to find a node to stamp on the Work Order, at the Work Order creation if no NodeKey is specified

If Work Order has a product on it:

Sourcing Rules for Provided Services will be used to find a node to stamp on the Work Order and on the product line, at the Work Order creation if no NodeKey is specified

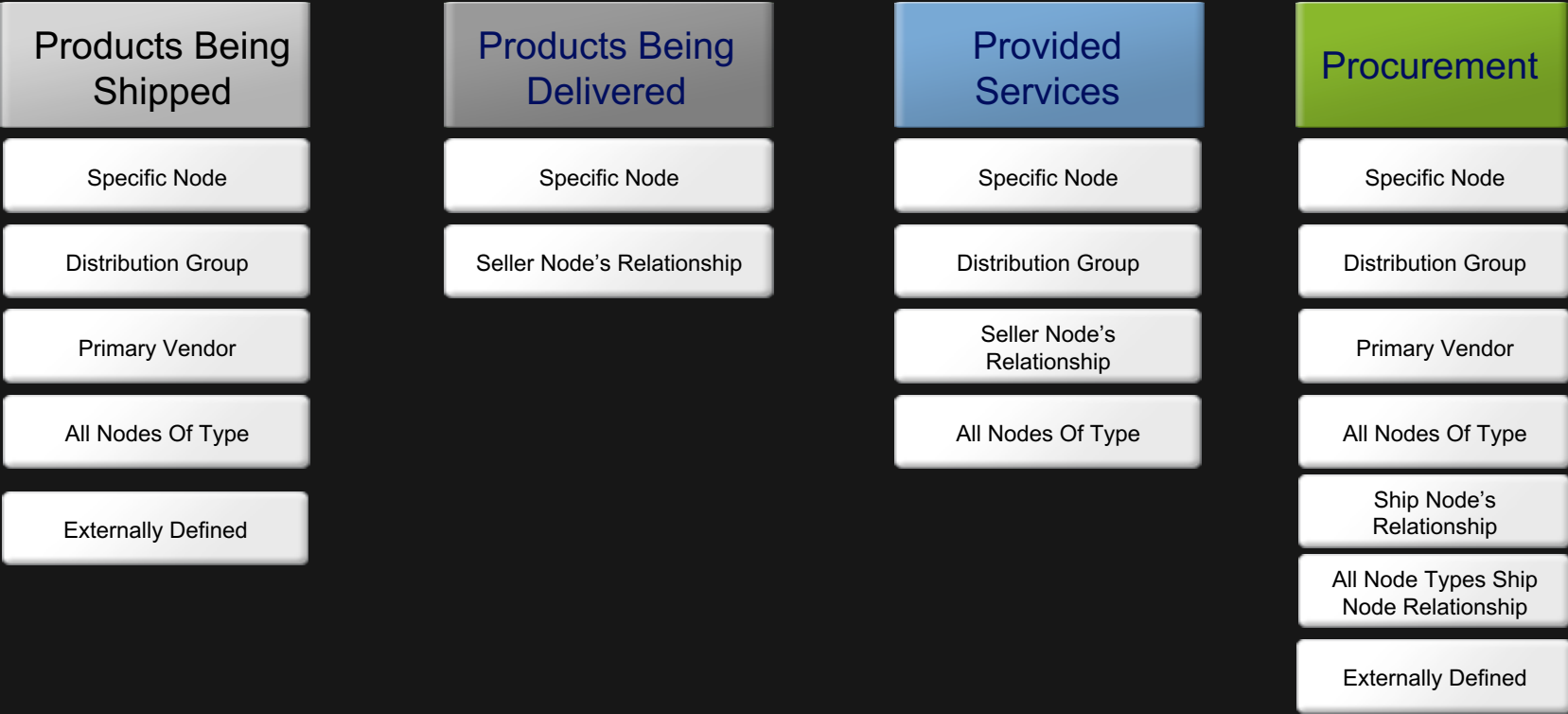
Procurement

If a product is needed to be procured from another node and ProcureFromNode is not specified on line; Procurement Sourcing Rules will be used

Sourcing Stack



Following is the stack of template driven sourcing details:



Sourcing Determination and Filters



Specific sourcing rule will be determined based on information on Customer and Order/Item Attributes

File Applications Action Help

Sourcing Rule for Product Being Shipped (DEFAULT)

Fulfillment Type

Order Sourcing Classification

When Seller organization is

This Organization

All Sellers

And Product characteristics are

Item ID

Item Classification

Primary Classification (Extn_ExtnDescription)

Secondary Classification (Extn_ExtnItemclassification)

Tertiary Classification (Extn_ExtnItemNumeric)

All Items

And Product is being shipped to

This Region

This Node

The Nodes of Type

Any Address

Use following sourcing templates in the sequence specified

Sourced From List	
Sequence No	Sourcing Template

Sourcing Determination and Filters



Filters further helps in reducing the number of nodes to be considered for fulfilment based on Inventory, Capacity, distance, etc

The screenshot shows a software window titled "Applications Manager" with a sub-header "Sourced From Details". The window contains several configuration options for sourcing:

- Sequence No: 1
- Template Type: Distribution Group : <... >
- Substitution Is Allowed:
- Expand to next sourcing sequence to minimize number of shipments:
- Consider only those nodes that are Within: 0.00 (Radius)
- Consider Only Those Nodes That Have A Minimum Available Capacity Of: 0 % Within The Next: (Day(s))
- Consider Only Idle Asset:
- Is Smart Sourcing Allowed:
- Procure/Transfer to this Node when inventory is not available:
- Work Order Creation Is Allowed:
- Consider the following inventory during sourcing:
 - All Inventory:
 - Inventory that will be available in the next: 0 day(s)

Best Practices for Optimal performance



Number of nodes qualifying from sourcing should be as minimized by

- : Evaluate Region based sourcing set up
- : Use of Proximity in sourcing
- : Custom logic
- : Smart sourcing
- : Multi-sequence sourcing rules

If all nodes are qualifying, Capacity availability constraint should be avoided or capacity cache should be used

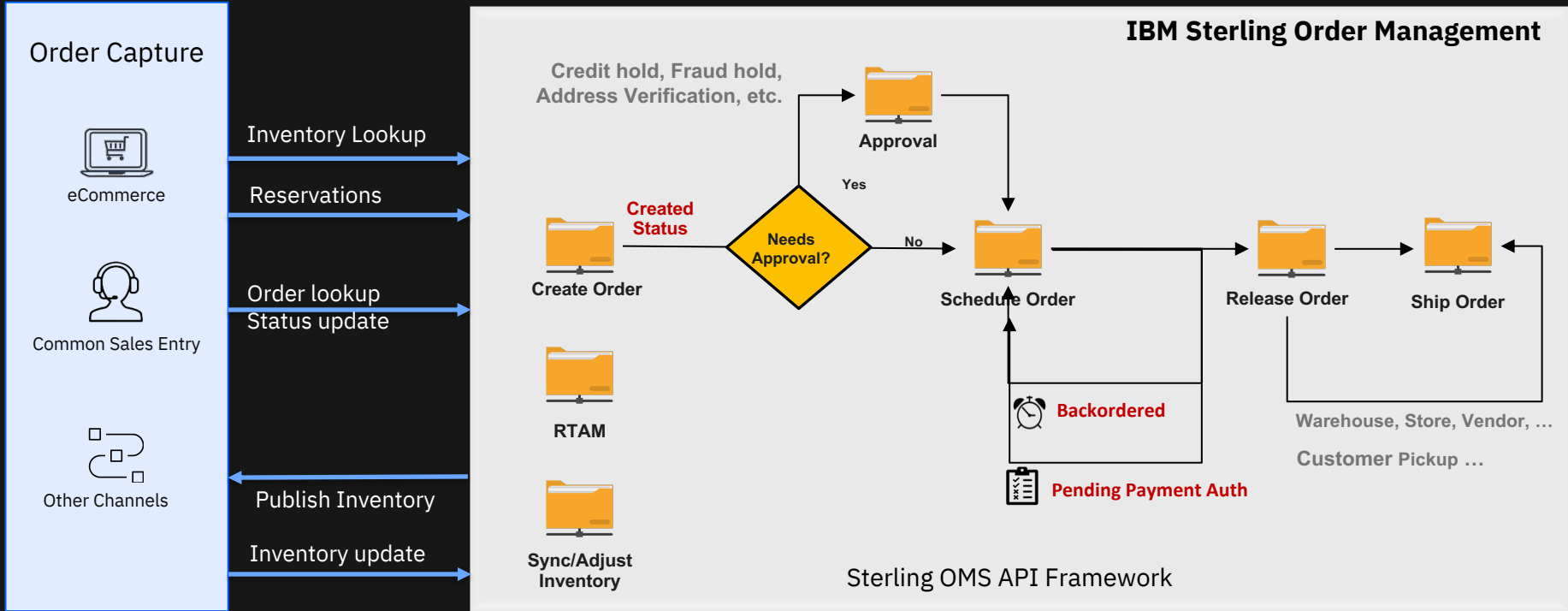
Use of Externally defined sourcing Vs Sourcing Correction

Order profile for pre-prod should be very similar to what is expected in production w.r.t DC Ship, SFS, Pick up.

If reservation node can be considered as final ship node then it should be passed on order line. This avoids schedule order to consider sourcing again.

Happy Path Order Workflow

...things happen...



Availability Lookup Considerations



If your inventory lookup is not right, then it will impact other component areas.

- *Reservations*
- *Order Capture*
- *Schedule Order*
- *Release Order*

ATP vs. Promising API

Use Optimal APIs and output template.

Procurements, optimizations (such as Cost, Date, etc.).

Shipment transfers, constraints (such as Ship Complete, Ship Single Node, etc.).

- `getATP, checkAvailability, getAvailabilityCache`
- `getAvailableInventory, findInventory`

Lead Times

Whenever we review customer's scheduling rules, we typically see that lead times are set to default (30/60 days).

How does "lead time" impact availability lookup APIs? What should be the optimal value?

Can availability lookup be kept simple as scheduling process will handle the optimization?



[Lead Time configuration](#)

Solver Optimization

There is quite a bit of optimization that happens within promising API (Cost), as such there might be certain level of performance implications.

What are mandatory and safe guarding properties that we can set to ensure cost based optimization done by promising logic does not add overhead? How to keep optimization light-weight?

`yfs.solver.MaxChoiceFailures`

Smart Sourcing

Legacy

Reading and processing inventory for nodes that do not contain inventory is costly.

To improve performance, smart sourcing can be used to dynamically determine the nodes to consider for sourcing product items. How does this really work and what benefits can we expect for inventory lookup promising APIs?



[Smart Sourcing](#)

Optimistic Lock Avoidance

Inventory locking only on low availability -

`yfs.hotsku.lockOnlyOnLowAvailability =Y`

Initial availability assumed to be high and lock is avoided until availability becomes low (an entry is present in the `INV_INVENTORY_ITEM_LOCK` table)

Availability calculation does not lock the `YFS_INVENTORY_ITEM` record unless corresponding record exists in `INV_INVENTORY_ITEM_LOCK` for the demand type.

**Not applicable for OMoC NextGen 2.0*

- **Reservations**
- **Order Capture**
- **Schedule Order**
- **Release Order**



[All about HotSKU
Optimistic Lock Avoidance](#)

INV_INVENTORY_ITEM_LOCK

- Rolling average calculation determines if availability low
- Entries for item-node-demand type combination
- If availability becomes high, then `INV_INVENTORY_ITEM_LOCK` record is removed.

Other Impacting Properties

- `yfs.hotsku.useGranularLockingForItem` Whenever availability at a ship node is low, a lock record will be inserted for Item-ship node combination
- `yfs.Hotsku.useAvailabilityAcrossNodes` Considers availability across all the nodes to decide if availability is low and item needs to be locked.

Purpose

- The purpose for the lock record. Valid values:
 - 10 (Lock as Availability is now low),
 - 11 (Use previous Hot SKU functionality).
 - Added 20 – Low availability when granular locking is enabled.

Avoid locks to YFS_INVENTORY_ITEM

- `yfs.hotsku.lockItemOnInventoryChanges=N`
- Inventory adjustments continue to contend on the same `YFS_INVENTORY_ITEM`
- Lock contention moved from item level down to the item-supply/item-demand level

Damage Control

Techniques to bring Stability to Order flow

Throttle the costly transactions, or make it faster.

- ***Inventory Lookup***
- ***Reservations***
- ***Order Capture***
- ***Schedule Order***
- ***Release Order***



Understanding Backlog

- Understanding the pending workload and Business commitments.
- What is the priority?

JVM Profile

- Reduce # of JVMs
- Reduce # of Threads
- Review Heap setting in reference to verbose GC logs.

Segregate Workload

- Orders are not getting scheduled as there are excessive orders getting backordered due to insufficient inventory, how do we improve the scheduling rate for new orders? (OrderFilter=N)
- Releasing SFS and BOPIS orders are priority, how do I ensure those orders get released within desire SLA? (e.g. SFS vs. BOPIS vs. Ship from DC)

Find the bottleneck

- What are the dependent transactions (Payment Auth, Holds, etc.)?
- What are the costly transactions (On success events, etc.) ?
- What were the recent changes?

RTAM

Often, we have seen the issues where Activity based RTAM doesn't perform and not able to publish inventory updates fast enough.

There could be many reasons for this, but main reason that we have seen are:

1. Burst of inventory updates from source system.
2. Slow Supply correction or DG override UE's.
3. Sudden inventory activity insert due to store capacity updates.
4. Configuration (monitoring large number of nodes) .

- **Inventory Lookup**
- **Reservations**
- **Order Capture**
- **Schedule Order**
- **Release Order**



RTAM Options

- Activity Based
- Full Sync (should we be running this instead of activity based?)
- Quick Sync (Often overlooked)
- *How to decide and benchmark the RTAM mode? Any Scenario?*

Follow Best Practices

- Impact of `createInventoryActivity`.
- *What are best practices if RTAM activity based is used to manage Store Capacity (capacity free or filled event)?*
- Avoid monitoring unwanted delivery methods.
- Avoid running full sync with node level monitoring.

Controlling Workload

- RTAM/System should be tuned to process the burst efficiently
- Also needs to be controlled at the source system i.e. OMS should not receive the inventory updates in a big batch

RTAM – Housekeeping

- *Do we have unprocessed/obsolete records in `YFS_INVENTORY_ACTIVITY`/`YFS_INVENTORY_ALERTS`?*
- *What should be the ideal size `YFS_INVENTORY_ACTIVITY` table when running activity based RTAM.*
- *My `YFS_INVENTORY_ALERTS` table had 100 Million records, should we be worried?*

MQ Configuration

Recommendations below are to help mitigate risk against MQ message creation and consumption, which will only magnify under peak load

1. Review SST Performance dashboard and address any JMS-related exceptions before peak ([see KC link](#))
2. IBM will ensure **MaxChannels** is set to new default of 5000
3. Avoid using message Selector instead have dedicated internal/external queues.
4. Requests to IBM OMoC team for MQ config changes should be supported by results/validation from performance testing and tuning exercise
5. Avoid long running MQ transactions (*hitting default **MaxUncommittedMsg** limit likely indicates too many long running transactions*)
6. Review **MessageBufferPutTime** relative to **ExecuteMessageCreated** statistic from **YFS_STATISTICS_DETAIL** table for any slowness
7. Leverage Self-Serve Tool to find current MQ queue depth
8. Check for queues with very high queue depth(> 15000). Typically queues with large queue depth indicates the backlog and requires immediate action to clear the backlog
9. Ensure JMS properties **yfs.jms.sender.multithreaded**, **yfs.agent.bulk.sender.enabled** & **yfs.jms.sender.anonymous.reuse** are enabled.
10. Engage OMoC Support to generate alerts for specific queues at particular depth threshold

READ OUR BLOG! [MQ on OMoC - gearing up for the peak](#)

Key Lessons Learned

- **MQ Persistent Connection** usage and importance of **retry**
- **JMS Connections** and the potential impact on external system
- **MQ Failover** or network event does terminate in-flight and idle connection
- Understand behavior of applications connecting to IBM MQ when persistent connection are terminated
- Consideration around **reprocessable** service.
 - Reprocessable service can result in server staleness if there are erroneous messages with message length > 1 MB.
 - For erroneous message < 1 MB will be persisted in **YFS_REPROCESS_ERROR** table.
- For custom service, ensure JMS Sender properties are set correctly
 - Retry Interval (milliseconds) 100 ms
 - Number of Retries – at least 3 retries.

Are You READY?

- Are you aware of, and leveraging the new SST performance dashboard?
- What is your current MQ connection / Max Channels high watermark in use?
- How does this compare to **MaxChannels**? Do you have sufficient buffer?
- Are you receiving alerts for max queue depth for all relevant queues?
- Are you checking in on queues with very high queue depth (>15000)



Inventory Visibility

IBM Suggests you take on the following activities proactively:

1. Run a round of **supply sync** to ensure supply picture is updated at IV prior to the peak sales
2. Run **DG sync** a day or two prior to the peak sales, Use **syncDgAvailability = Y** as part of Update Distribution Group
3. Plan ahead to **avoid making changes to DGs** at peak time.
4. Set **safety stock** for items , this helps maintain a buffer inventory to ensure the channels do not over-promise
5. Ensure custom processes reaching out to IV are not generating **excessive tokens**
6. Have a **retry mechanism** for any 5XX errors being seen from the frontend systems. OOB adapter currently handles these errors using an auto retry mechanism set for agents and integration servers.
7. Avoid **payload limits** from being hit by ensuring the payload size that reaches IV from all channels is <500 KB
8. If using phase 1 IV adapter, ensure below properties are set-
 1. `yfs.purge.MergeDemandSupplyMultiRec=true`
 2. `yfs.inventorySnapshot.DemandSupplyMultiRec=true`

Maximizing IV usage

1. To enhance the performance:
 - Call IV directly to make reservations, as opposed to using `reserveAvailableInventory` from OMS
 - Reach out to IV directly for real-time availability (from front end)
2. Consider using Availability APIs:
 - For product details ,use the Get Node Availability Product Breakup API or Get Network Availability Product Breakup API (`/availability/{itemId}/network`) for the item in question.
 - For carts or orders, use the Get Node /network Availability API

Snapshots

1. Ensure that the front end has updated snapshot of availability by calling `ProductAvailabilityToSell.ShipNodeSnapshot` or `DgAvailabilityChange/publish_all` few hours prior to peak sales
2. Use the new Jobs API to understand which jobs are in progress (like DG sync, `ProductAvailabilityToSell.ShipNodeSnapshot`) before triggering another job
3. If using COS, request access to the cloud-storage-tools utility which will help with download/deletion of events from COS buckets

Real Time Availability Monitor (RTAM)

1. **Full sync mode is expensive**, should be scheduled only during low-traffic hours (ie. during night). For peak season, careful planning needs to be done around business events and planned traffic
2. **Preferred methods** to push inventory availability out to external system are **Activity-based** and **Quick-sync** mode
3. **For Activity-based RTAM**, validate activities created by node capacity changes
4. **Optimize RTAM performance** with this high level configuration:
 - `yfs.jms.sender.multithreaded`
Enable when publishing alerts to external queue; allows RTAM agent threads to publish message parallelly
 - `yfs.yfs.rtam.readInventoryForOnlyActivityNodes`
Implicitly enabled based on conditions:
 - a. Running activity based RTAM
 - b. Monitoring item availability at node level and network level.
 - c. Property not explicitly disabled (i.e. `yfs.rtam.readInventoryForOnlyActivityNodes=N`)
 - d. Monitored items is not a bundle.
 - e. All DGs at network level are monitored at node level
 - **Number of Records To Buffer** can be increased to 25K in reference to `MessageBufferPutTime`
 - Disable `Compute availability information with Ship Dates for RTAM` flag (`COMPUTE_AVAILABILITY_FOR_RTAM = N`) if availability information is not required.
 - Optimize Event template to remove unnecessary attributes

Hot SKU / OLA


Legacy

1. Hot SKU & OLA feature are enabled by default in OMoC.
2. Review and tune Hot SKU factory values based on expected order profile.
[KC Link](#), [Community article](#)
3. Review `INV_INVENTORY_ITEM_LOCK` table during performance tests and pre/post sales to assess the level of contention and items involved.
4. With OLA enabled, there will be granular level of locking on `YFS_INVENTORY_DEMAND` record (item, ship node, demand level)
 - During peak time, avoid running inventory synchronization which updates `YFS_INVENTORY_DEMAND` along side `YFS_INVENTORY_SUPPLY`
 - Configure RTAM monitoring to generate alert to replenish the inventory or mark the item unavailable

Capacity

- Enhance the promising APIs performance by enabling capacity availability agent. Additional Guidance: [KC Link](#), [Blog coming soon!!](#)
 - This is a time-triggered agent that calculates the capacity upfront and stores in the `YFS_CAPACITY_AVAILABILITY` table from where the data can be read during promising calls.
- Depending on the business objective consider following property `yfs.nodecapacity.lock=N` to avoid locking during capacity check.

Smart Sourcing

- Reading and processing inventory for nodes that do not contain inventory is costly.
- To improve performance, smart sourcing can be used to dynamically determine the nodes to consider for sourcing product items.  [Smart Sourcing](#)

Agent/Server Configuration

The following configurations have repeatedly been recommended to clients to help address known performance, stability, scalability issues:

- 1. Tune memory parameters** based on analysis from GC logs ([KC link](#))
 - For NextGen platform, ensure appropriate profile is selected based on testing: Balanced, Compute, Memory
- 2. Optimize schedule** of batch processes around expected peak loads
 - Ex. Complete Inventory Sync before the peak hours begin; Add more JVMs in the wee hours of the peak day and complete the process well before the rush*
- 3. Disable non-critical agents** to reduce unnecessary contention and usage of resources by disabling any processes that are not business-critical during peak
 1. Disable IBA which is known to be intrusive, and best to avoid peak usage
 2. Explore option of disabling non-critical Order/Shipment Monitor rules
 3. Configure and run Close Order/Shipment agent before peak season ([link](#))
4. Avoid having multiple agent criteria in same agent configuration; makes for easier troubleshooting.
5. Ensure naming convention of agent/integration server accurately represent function which it performs; makes for easier troubleshooting and assessing the impact
6. Reduce message payload by optimizing API, event templates, pull only required data (set `TotalNumberOfRecords` to restrict output)

Performance Profiles

NextGen

There are three server performance profiles for the agent or integration server.

- 1. Balanced:** Provides moderate memory and computing power for typical OMS workload.
- 2. Compute:** Provides additional computing power and moderate memory. This type of profile is more suitable for workload like RTAM, etc.
- 3. Memory:** Provides additional memory and moderate computing power. This profile type is more suitable for purge agent.

Which profile should we use?

There is no definite formula to identify the right profile. However, you can use the following guidelines to select a performance profile for optimal results.

- Start with a single thread for the server, single instance of the server, and the **Balanced** profile.
- Increase the number of threads gradually to arrive at the correct profile and maximum number of threads per server.
- If CPU or memory allocation does not change significantly with each additional thread, continue with the **Balanced** performance profile. Servers that spend most of the time calling external services display this kind of resource use pattern.
- If the JVM heap utilization stays around 80% or increases significantly with each additional thread, change the profile to **Memory**.
- With any of the performance profiles, if the CPU allocation stays around 70% or memory allocation stays around 80%, you might scale the server (Pod) rather than increase the number of threads.

Guidelines [for selecting the performance profile to improve server performance](#)

- **Self Service Monitoring Dashboards – JVM, DB, JMS, API and Service performance.**
- **OMS SMC Console – Monitoring cache statistics and making thread changes.**
- **Query YFS_STATISTICS_DETAIL table for API and Service performance stats.**



OBJECTIVE: Selecting an optimal performance profile

Next generation



PROBLEM: Target to achieve 30k TPH for createOrder w/ 2.5 average lines

- Select optimal server profile and thread configuration for createOrder integration service to ensure service can scale w/ custom logic and configuration.



APPROACH:

Configure create order integration server in OMS

- Initial Threads =1
- Performance Profile = Balanced
- Default # of JVM Instances = 1

Execute and monitor the server performance via **Self Service Tool**

- API Response time (ms)
- Invocations (rpm)
- Container CPU Utilizations
- GC CPU Utilizations
- JVM Heap Utilization
- Order Lines Throughput

Observe and Adjust the configuration until throughput is achieved

- Increase the # of threads
- Switch Performance Profile
- Increase the # of JVM instance

NOTE: Below numbers represent OOB createOrder with some customization

Server Type	Integration	Performance Profile	Balanced				
Server Name	CreateOrderServer						
JVM Instances	No. of Threads Per JVM Instance	API Response (ms)	Invocations (rpm)	Container CPU %	GC CPU %	Heap Utilization	Order/Hour (2.5 average lines)
1	1	198	305	51.5	3	52.7	18760
1	2	285	420	85	7	57.8	25200
1	3	410	432	96	12	62.4	25920
2	4	580	804	99	19	70	47398

Server Type	Integration	Performance Profile	Compute				
Server Name	CreateOrderServer						
JVM Instances	No. of Threads Per JVM Instance	API Response (ms)	Invocations (rpm)	Container CPU %	GC CPU %	Heap Utilization	Order/Hour (2.5 average lines)
1	1	182	324	25.6	1.4	29.1	19440
1	2	196	612	47	3	35.5	36720
1	3	219	810	61	5.87	44.2	48600
1	4	230	1041	75	6.47	51.7	62100

Optimal Solution:

- ✓ Threads = 3
- ✓ Performance Profile = Compute
- ✓ JVM Instances = 1



RECOMMENDATIONS:

- Spawning additional (untuned) instances of agent to try and improve throughput let to **exhaustion of resource** allocation available
- Review KC Guidelines to select performance profile** | **Review community article on Sterling OMS Performance Profiles**



OBJECTIVE: Avoid Message Selectors



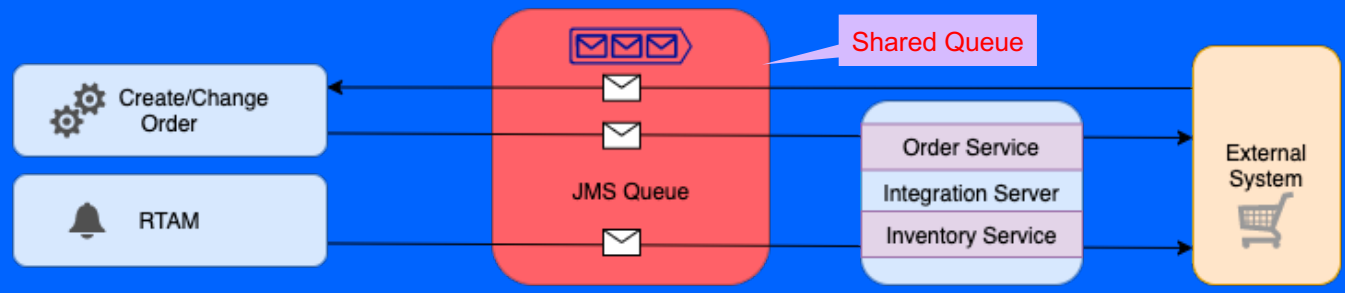
PROBLEM: OMS is experiencing downtime in Production !

- Significantly high backlog for some of the order process integration queues, createOrder not moving / extremely slow for several hours
- RTAM server was stuck for several hours resulting in impacting inventory picture on eComm and other channels



OBSERVATIONS:

- Consistently high CPU observed on MQ server.
- Identified that primary contributor of MQ high CPU / slowness was due to use of MQ 'selector' to pull specific message types from a shared queue
- In one case, shared queue used for WCS-OMS integration reached 800K messages, both WCS/OMS were PUT/GET to same queue as part of process



- Issue was resolved by eliminating the use of message selector by separating the queues within the integration pipeline.



RECOMMENDATIONS:

- Do not use message selector where equal distribution of workload is not guaranteed
- Understand transaction flows, timings and high watermarks



OBJECTIVE: Avoid Server Staleness

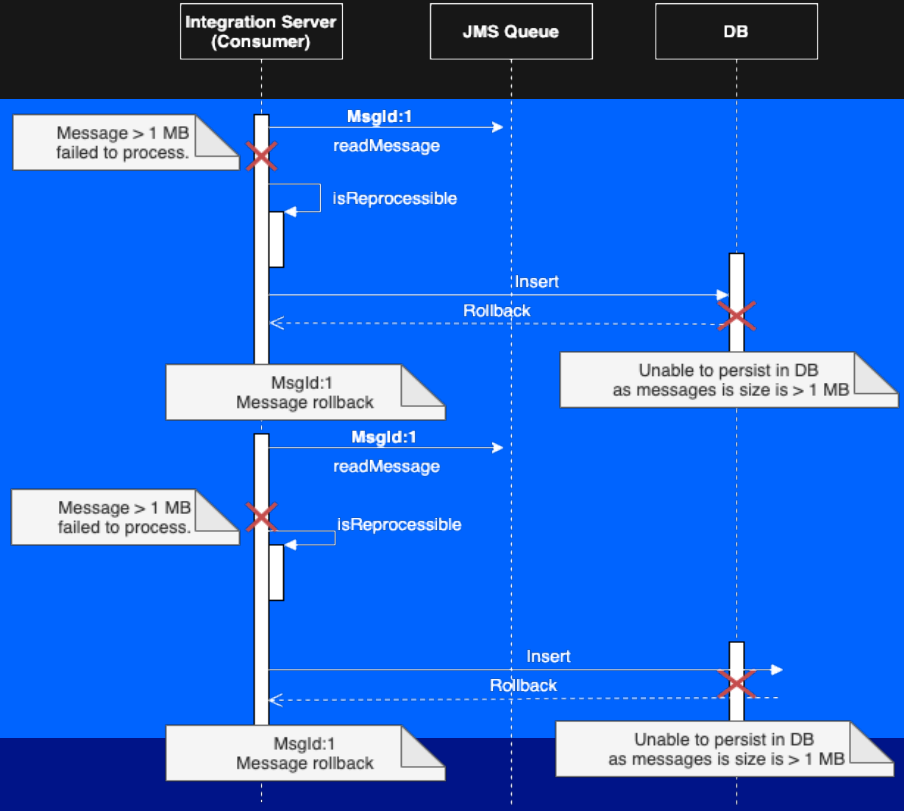


PROBLEM: Orders not being processed in Production !



OBSERVATIONS:

- Excessive Errors noticed as part of proactive monitoring from `createOrder` Integration Server:
 - `ErrorCode="YFC0001" ErrorDescription="Record already exists in the database."`
 - `ErrorCode="YDB92_001" ErrorDescription="The data bound to the column is invalid in this context."`
- Reprocessable errors flag set to Y
- Message size >1 MB in corresponding queue, Was unable to insert on `yfs_reprocessable_error` table as MESSAGE size exceeded 1 MB
- Server running in single threaded mode
- Mitigation – Removed “bad messages” manually from queue



RECOMMENDATIONS:

- Exercise caution while accepting payloads for integration servers with `reprocessable` errors flag set to Y
- Look into increasing threads on integration server as interim relief

Performance Testing Guidance



Performance testing is an art, but a mandatory one! It is imperative to vet out issues in advance on pre-production load testing, rather than wait for it to surface as a business critical production issue!

Refer to Knowledge Center for detailed Tuning and performance guidance.

- 1. Projected peak volumes** – Ensure business and IT are in sync on expected peak loads to ensure planned tests are accurate. Test for forecast of peak orders, orderlines, common cart skus/items, and real time inventory calls
- 2. Representative Combination Tests** – Assemble components to reflect real time DATA, scenarios and run in parallel to ensure adherence with NFR; Stage data for various components and run them under full load (ie. Create + Schedule + Release+ Create Shipment + Confirm Shipment + Inventory Snapshot (IV))
- 3. Agent and integration servers** – ensure asynchronous batch processing components are tested in isolation and in combination with broader workload; ensure to tune agents (processes, threads, heap size) to meet expected peak SLAs/NFRs on throughput
- 4. Test Failure Scenarios** – validate resiliency of overall system and operations, ensuring graceful recovery if front-end channel (web, mobile, Call Center, Store, EDI, JMS), backend OMS, or external integration endpoints fail. Include ‘kill switches’ in any components that can be disabled to avoid magnifying an isolated issue into system wide one, especially for any synchronous calls.
- 5. Confirm Peak days and Hours** - Share any specific key dates or max burst times with IBM Support, including code freezes, flash sales.
- 6. Coordinate with IBM** - Inform IBM (CSM/Support) in advance when load tests are planned if any data or diagnostics (such as against Database) are needing to be captured; IBM can also then review internal metrics and response in parallel.
→ **Inform IBM in advance of major configuration changes (sourcing rule: Increase in SFS due to COVID).**

Ensure you are clear on the actuals vs. projections of to facilitate accurate tests, and for IBM OMoC team to validate sufficient resources

Metric	2020 Peak	2021 YTD Peak	2021 Proj. Peak	2021 Load Test Peak
Orders / hour (max)	?	?	?	?
Orderlines / hour (max)	?	?	?	?
Get Inventory Availability	?	?	?	?
Reserve Inventory	?	?	?	?
Inventory Adjustment trickle	?	?	?	?
Inventory Adjustment burst	?	?	?	?
Concurrent Store/CC users	?	?	?	?

- Are you leaving sufficient buffer between load tests and holiday freeze?
- Are performance tests representative of volumes (order lines, real time sync calls), plus:
 - Asynchronous batch processes (agent and integration servers)
 - Store (Pick, Pack, Ship) and Call Center concurrent user activity. **Validate all navigation flows.**
- Do your performance test runs cover any potential common cart item, to determine if further configuration changes may be needed to avoid contention:
 - Realistic inventory availability picture
 - Orderlines to include expected commonality of free gift / hot items
- Performance validation of initial data load related to peak. (Migrating Orders, Catalog, etc)
- Initial Inventory synchronization to IV.
- Will the endpoints external to OMoC be able to scale (rate limits, performance)?
- Adoption of Self Serve Tools & subscription to alert and notification.

Position for Peak Success



To best position for success on the OMS platform, it is important to understand how your application handles various scenarios known to challenges performance or stability. Testing in pre-production with data/workloads representative of production enables ability identify and address issues without impact to production business and operations.

1. Resource/Hardware **sizing** based on segment profile, but is validated as OUTCOME of performance testing, not a replacement for it
2. **Database** is common bottleneck, not due to capacity, but untuned queries, missing indexes, competing processes, unqualified end-user searches
3. Underlying config **data** has significant impact on performance, including database query execution plans, inventory sourcing rule evaluation
4. Accumulation of **transactional data** over long periods of time (and failure to purge as possible), may degraded query performance
5. **Item distribution** and commonality must reflect realistic peak load; high-demand / hot items (free-gift) may significantly impact concurrent processes
6. Composition of a custom service (**Service definition framework**) can lead to inefficient execution or potential lock contention, reducing throughput
7. Understanding **queueing**/de-queueing rates to align with business SLA / expectation (ie. create order, confirm shipment); SI needs to know when there is an issue to intervene / troubleshoot (ie. particular queue depth)
8. Agent/integration server throughput must be sufficient, but remain below **max resource allocation**; varies on number of instances, server profile
9. Important to understand / validate impact to upstream application (eComm) if specific synchronous calls into OMS slow or become unavailable



Real Scenarios (Real impact...)

- COVID-19 led to more stores enabled for BOPIS which made DG significantly larger, which led to more time for synchronous inventory availability calls; similar scenarios where client had to split nodes in DG to improve throughput
- Rapid ramp up of **in-store associates** led to several unqualified searches in Store and Call Center apps which caused significant overall degradation
- multiAPI made 8 successive API calls led to poor response, needed to be refactored to **use asynchronous** requests (via MQ to drop message on queue)
- **Custom service call** to getOrderList API was missing OHK in input, each invocation caused fetch of 5K records which led to a crash, had to limit records
- Needed to **throttle down** instances/threads of agent to reduce concurrency contention issues (Create/Schedule/Release) and optimize throughput
- Gradual **memory leak** led to out-of-memory condition after a couple days; similarly untuned heap led to excessive GC overhead, high CPU, slowness
- Daily manual processing of orders via java client against single JVM bypassed load balancer and overwhelmed JVM to OOM/crash
- Upstream eComm site was unable to gracefully handle a short period of unavailability from backend OMS and took hours to recover
- Unintentionally carrying capacity for high volume node during the peak. (Example: Popup /Temporary fulfilment warehouse)
- Avoid changes to DG in IV during peak time

Understanding the problem – well defined problem is easy to resolve (mitigate)

Timelines, Impacted components, Diagnostics captured, Business impact play an important role in the road to system recovery.

Application / Agents/ Integration Server Health

- Issue and alerting condition
- Measure of degradation and business impact seen – Performance or Functional?
 - **Performance:** Are all APIs slow or particular (e.g. Inventory Lookup / Reservation vs. Order status updates / order lookup)? Is it intermittent or always slow?
 - **Functional:** Is this issue reproduceable? Can issue be observed outside the monitoring app? Is there an error? few errors or excessive? How frequent?

Timing/Timeline

- When did the issue start?
- Any patterns observed with other processes or under specific load?
- Does issue worsen over time?

Impacted Components

- Are only certain servers affected?
- What are all dependent services/component for the servers? (External system, MQ, SMTP)

Be prepared for peak days!

- Confirm focal, maintain internal contact sheet.
- Define a support schedule (24x7 peak-hour coverage)
- Setup a communication plan.
- Document escalation criteria and procedures
- Ensure activities by business, performance, operations are in sync

Contact IBM Support in a crisis

- Contact IBM Support to open a severity 1 Case
- Escalate case by new Escalate button in the Support Portal



ICEBERG EFFECT

Supporting IBM Sterling Order Management on Cloud

