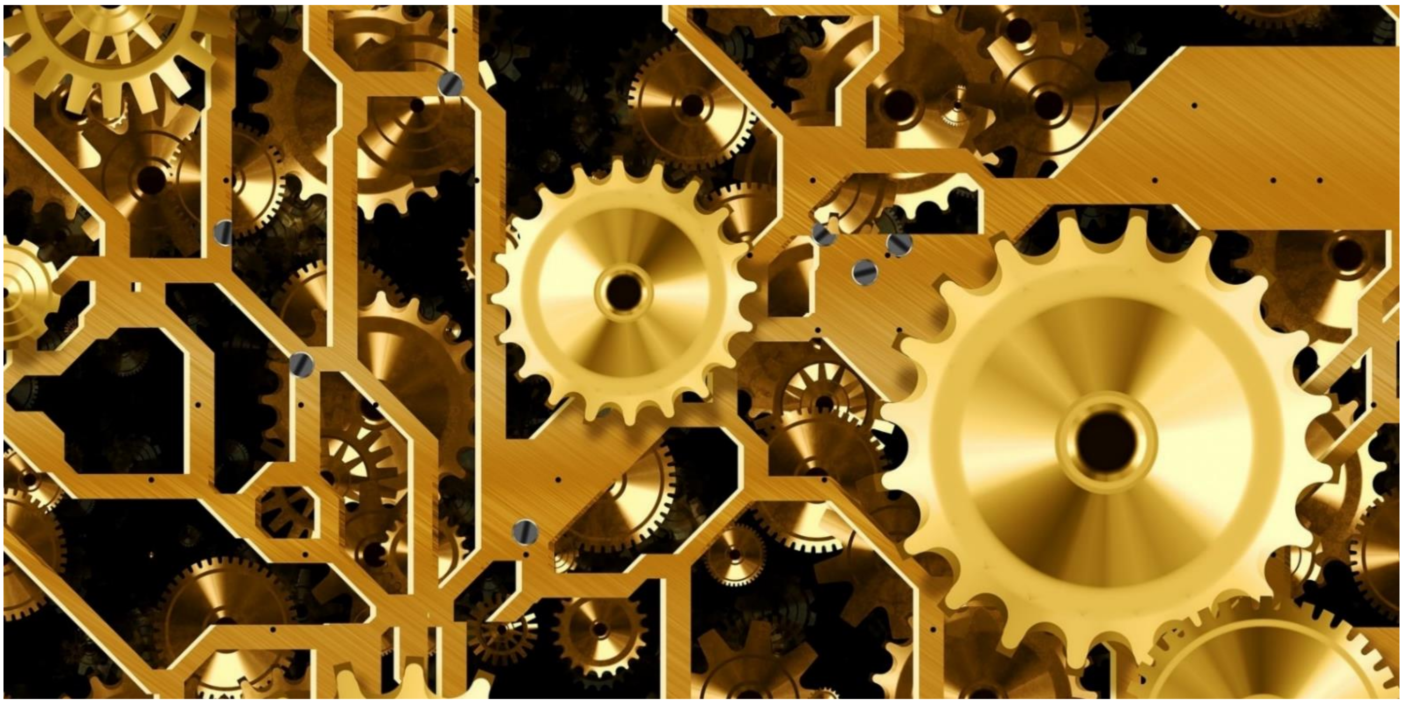


IBM Workload Scheduler
HCL Workload Automation

V10.1.0.1 Performance Report

Document version 1.0

Paolo Cavazza
Giorgio Corsetti
Lorenzo Nichele
Alessandro Tomasi

Workload Automation Performance Team - HCL Software Rome Lab

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Workload Scheduler is a state-of-the-art production workload manager, designed to help customers meet their present and future data processing challenges. It enables systematic enterprise-wide workload processing for both calendar and event-based (real-time) workloads across applications and platforms. Workload Scheduler simplifies systems management across distributed environments by integrating systems management functions. Workload Scheduler plans, automates, and controls the processing of your enterprise's entire production workload.

Pressures in today's data processing environment make it increasingly difficult to guarantee a high level of service to customers. Many installations find that their batch window is shrinking. More critical jobs must be finished before the workload for the following morning begins. Conversely, requirements for the integrated availability of online services during the traditional batch window put pressure on the resources available for processing the production workload.

Workload Scheduler simplifies systems management across heterogeneous environments by integrating systems management functions.

1.1 What's new since version 10.1

In the last year, a major release 10.1 and one fix pack have been released.

For more details about the new features introduced with these fix packs, see the Summary of enhancements in the online product documentation in IBM® Knowledge Center:

- 10.1 GA
- 10.1 FP1

This new major release 10.1 and the related fix pack introduced several changes in the Workload Scheduler infrastructure and functional capabilities. Some of these changes have been considered with special focus during Performance Test phase because there was the need to understand how those changes might affect or not the overall performance and scalability of the product, if compared with the previous releases.

In particular, the focus areas from Performance Testing perspective have been:

- Automatic encryption at rest for key product files: the key IBM® Workload Scheduler files (such as the Symphony file, messages queues, and the `useropts` file) are automatically encrypted for all fresh installations using AES-256 or AES-128 cryptography and/or the encryption can be enabled after having installed 10.1 release on pre-existing environments.
- New Workload Designer application in the Dynamic Workload Console (DWC in this publication): an innovative infrastructure and design have been thought to simplify the user experience and create a more responsive, fast, and fluid user interface.
- Automatic SSL configuration for fault-tolerant agents: configure IBM WebSphere Application Server Liberty Base and fault-tolerant agents in SSL mode and provide custom certificates at installation time using the command-line installation and setting the `sslkeysfolder` and `sslpassword` parameters. This ensures encryption in motion for all data moving within customer environments.
- Support for file transfers via the Workstation-to-Workstation internal protocol: it is possible to transfer files to and from agents connected to the same Master Domain Manager (MDM in this publication). It is no longer required to specify the address of the workstations involved in the file transfer, but it is enough to define the workstations from the File Transfer integration. The only constraint using this new feature is to have both the Master Domain Manager and agents at version 10.1 or higher.

2 Scope

2.1 Executive summary

The objective of the tests described in this document is to report the performance results for the new version of the product, V10.1.0.1, executed in a test environment based on the VMWare ESX® - Linux x86 in comparison with previous versions (see Workload Scheduler 9.5.0.2 performance report).

The most outstanding performance results could be summarized as follows:

- The throughput analysis reconfirms the performance and scalability levels assured in previous releases. From the workload scheduler user perspective, that means, for instance, no substantial delay in the scheduling of a job when the same job is ready to start and no substantial latency in the job and job stream status update from the Dynamic Workload Console, even during huge job submission peaks.
- After having enabled the new automatic encryption at rest and encryption in motion features, there are very minimal performance impacts on product throughput capabilities (e.g. average job schedule delay per minute for those jobs scheduled versus dynamic agents increased for a few seconds) and on Master Domain Manager CPU utilization during the jobs submission peak, but these minimal performance impacts are strongly balanced out by the benefits of having enabled a more secure configuration for the product.
- The scalability and the performance of the Dynamic Workload Console reconfirm the same level of quality assured in previous releases. Moreover, the new Workload Designer application confirms to simplify the user experience and creates a more responsive, fast, and fluid user experience.
- The new Workstation-to-Workstation file transfer internal protocol guarantees a faster file transfer scenario if compared with the previous implementation (file transfer from an agent to an SSH Server and then from the SSH Server to another agent). The improvements measured using the new protocol, in relation to the file transfer execution time, are in the range between 27% and 43%.

3 Performance Test

3.1 Test Approach

As specified in section 2.1, the main focus for Workload Scheduler 10.1 performance tests was to validate that the new version of the product allows the same results in terms of throughputs and capacity of previous releases; to achieve this, the scheduling throughput, resource consumption, and reliability are continuously monitored and certified.

In this context, continuous monitoring was applied with special focus on key performance indicators (scheduling and mirroring throughput, average delays, internal queues sizes) and on the main hardware resources (CPU, memory, disk busy) to prevent memory leaks, unexpected hardware consumption, and product performance degradation during long run workload scenarios.

3.2 Environment

The test environment was based on virtual machines hosted on VMware ESXi servers running on Dell™ PowerEdge R630 Intel™ Xeon(R) CPU E5-2650 v4 @ 2.20GHz. All tests were performed in a 10 GB local area network. The following table summarizes the software used and the version:

OS	Linux Red Hat® server 8.3 Kernel 4.18.0-240.15.1.el8_3.x86_64	
RDBMS	IBM DB2 v11.5.8	Oracle® 19c Enterprise Edition
J2EE	IBM WebSphere Application Server Liberty Base 22.0.0.13 IBM Semeru Runtime Open Edition 11.0.15.0 (build 11.0.15+10)	
LDAP	IBM Directory Server 7.2	
WA	10.1.0.1	

Table 1. Software level of code

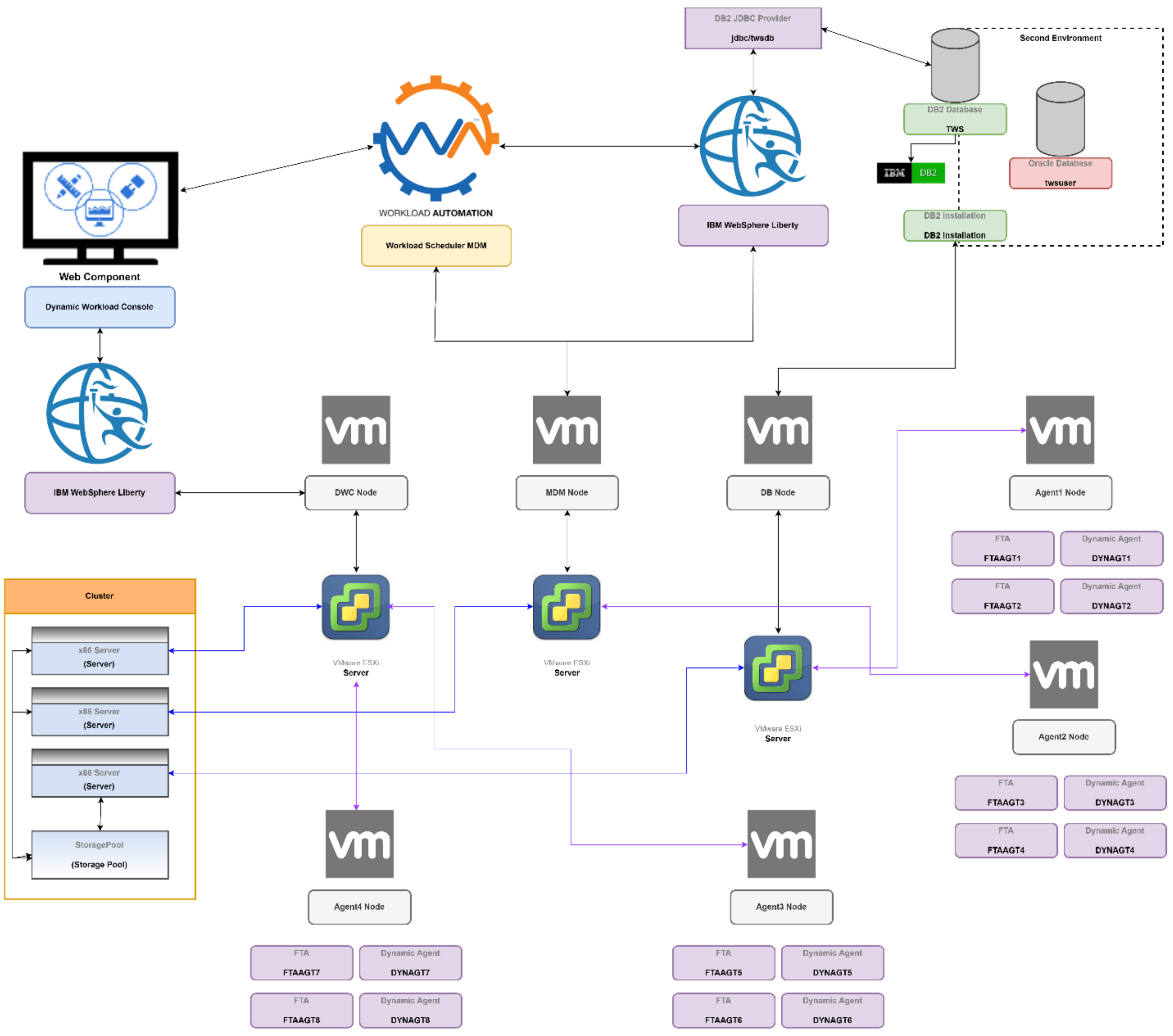
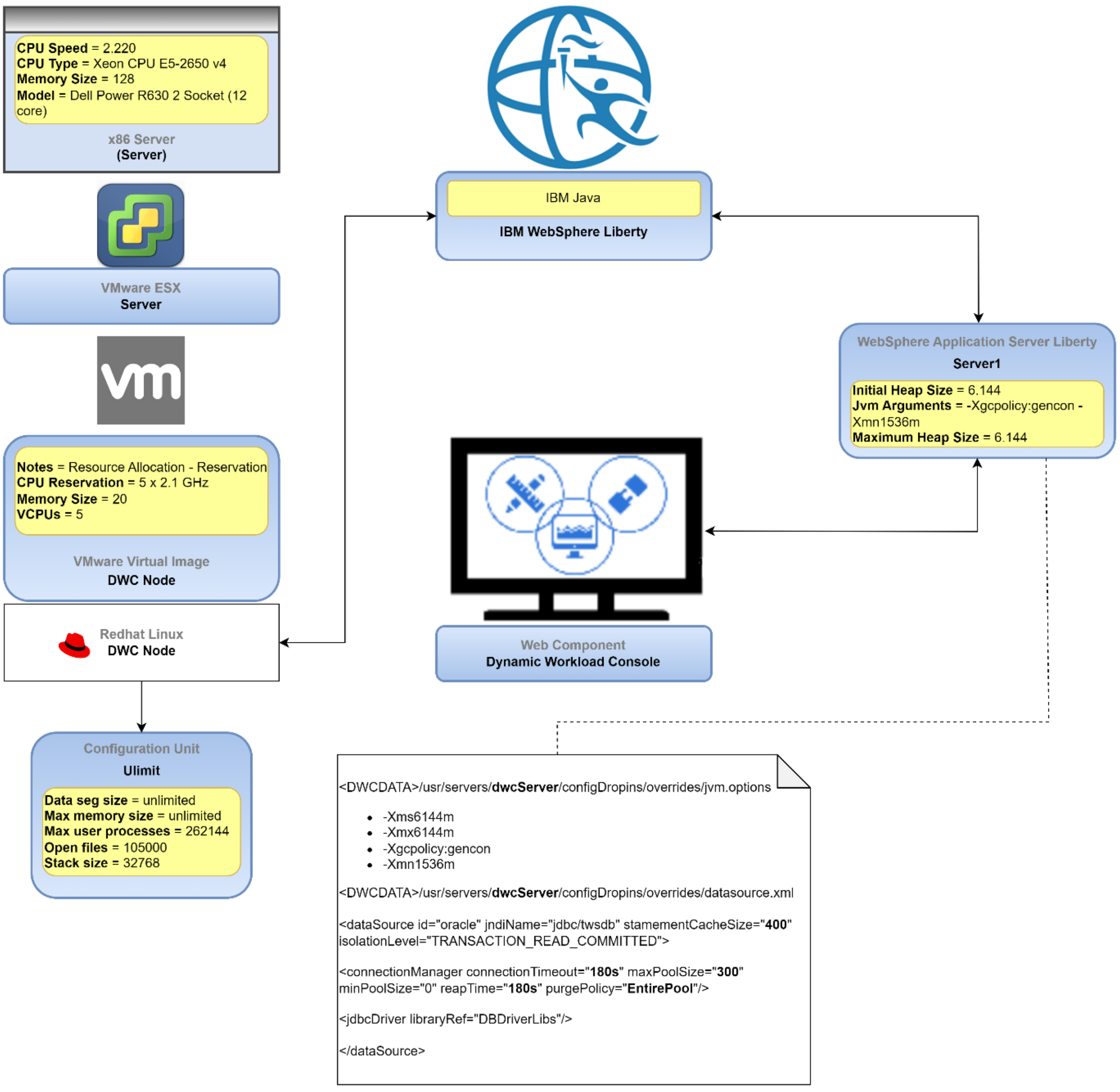


Figure 1. Overall deploy view of test environment


CPU Speed = 2.220
CPU Type = Xeon CPU E5-2650 v4
Memory Size = 128
Model = Dell Power R630 2 Socket (12 core)
x86 Server
(Server)
VMware ESX
Server
Notes = Resource Allocation - Reservation
CPU Reservation = 5 x 2.1 GHz
Memory Size = 20
VCPUs = 5
VMware Virtual Image
DWC Node
Redhat Linux
DWC Node
Configuration Unit
Ulimit
Data seg size = unlimited
Max memory size = unlimited
Max user processes = 262144
Open files = 105000
Stack size = 32768
IBM Java
IBM WebSphere Liberty
WebSphere Application Server Liberty
Server1
Initial Heap Size = 6.144
Jvm Arguments = -Xgcpolicy:gencon -Xmn1536m
Maximum Heap Size = 6.144
Web Component
Dynamic Workload Console

<DWCDATA>/usr/servers/**dwcServer**/configDropins/overrides/jvm.options

- -Xms6144m
- -Xmx6144m
- -Xgcpolicy:gencon
- -Xmn1536m

<DWCDATA>/usr/servers/**dwcServer**/configDropins/overrides/datasource.xml

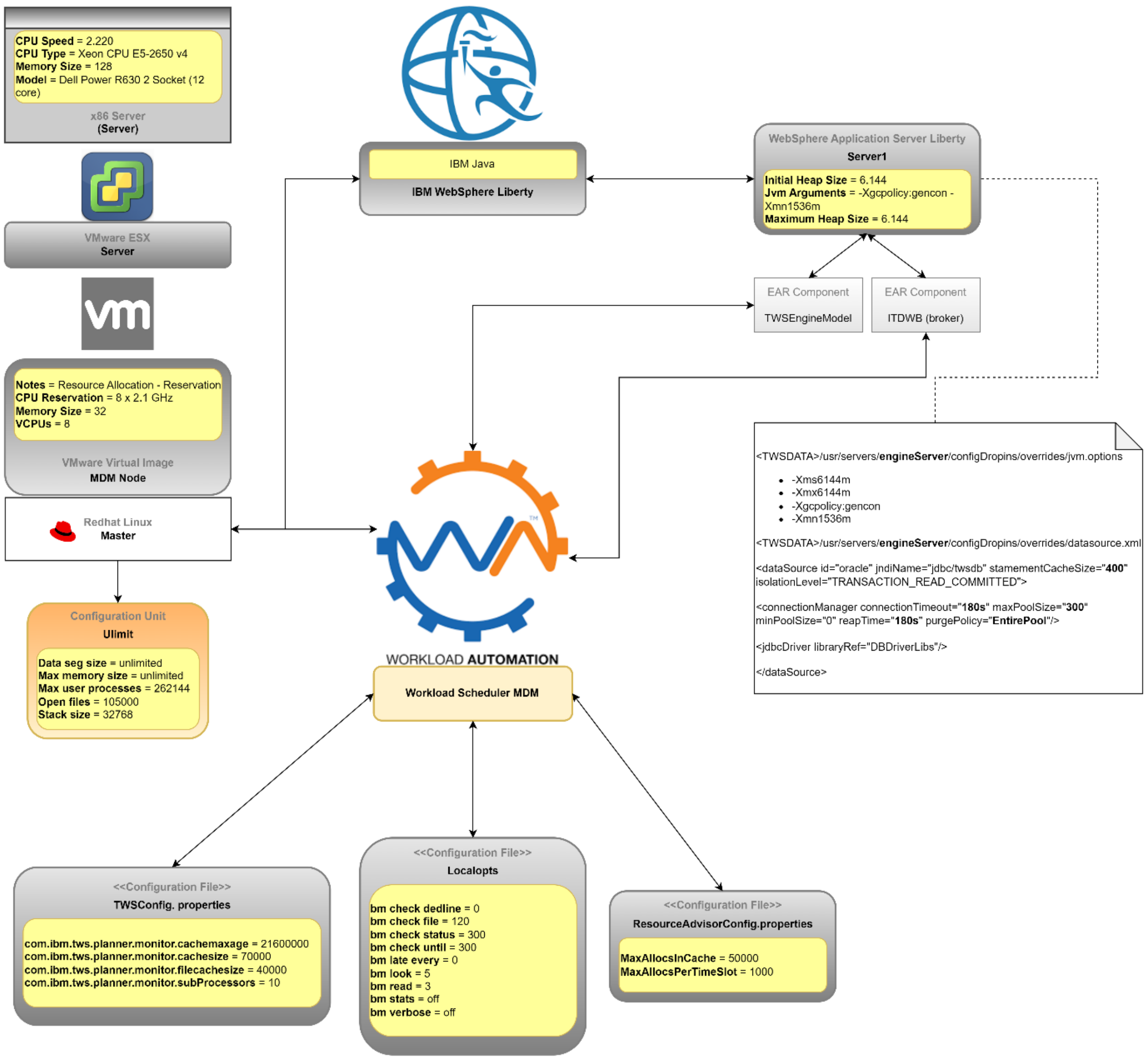
```
<dataSource id="oracle" jndiName="jdbc/twsdb" stamementCacheSize="400"
isolationLevel="TRANSACTION_READ_COMMITTED">

<connectionManager connectionTimeout="180s" maxPoolSize="300"
minPoolSize="0" reapTime="180s" purgePolicy="EntirePool"/>

<jdbcDriver libraryRef="DBDriverLibs"/>

</dataSource>
```

Figure 2. Dynamic Workload Console node configuration



CPU Speed = 2.220
CPU Type = Xeon CPU E5-2650 v4
Memory Size = 128
Model = Dell Power R630 2 Socket (12 core)

Notes = Resource Allocation - Reservation
CPU Reservation = 8 x 2.1 GHz
Memory Size = 32
VCPUs = 8

Data seg size = unlimited
Max memory size = unlimited
Max user processes = 262144
Open files = 105000
Stack size = 32768

Initial Heap Size = 6.144
Jvm Arguments = -Xgcpolicy:gencon -Xmn1536m
Maximum Heap Size = 6.144

<TWSDATA>/usr/servers/**engineServer**/configDropins/overrides/jvm.options

- -Xms6144m
- -Xmx6144m
- -Xgcpolicy:gencon
- -Xmn1536m

<TWSDATA>/usr/servers/**engineServer**/configDropins/overrides/datasource.xml

<dataSource id="oracle" jndiName="jdbc/twsdb" statementCacheSize="**400**" isolationLevel="TRANSACTION_READ_COMMITTED">

<connectionManager connectionTimeout="**180s**" maxPoolSize="**300**" minPoolSize="0" reapTime="**180s**" purgePolicy="**EntirePool**"/>

<jdbcDriver libraryRef="DBDriverLibs"/>

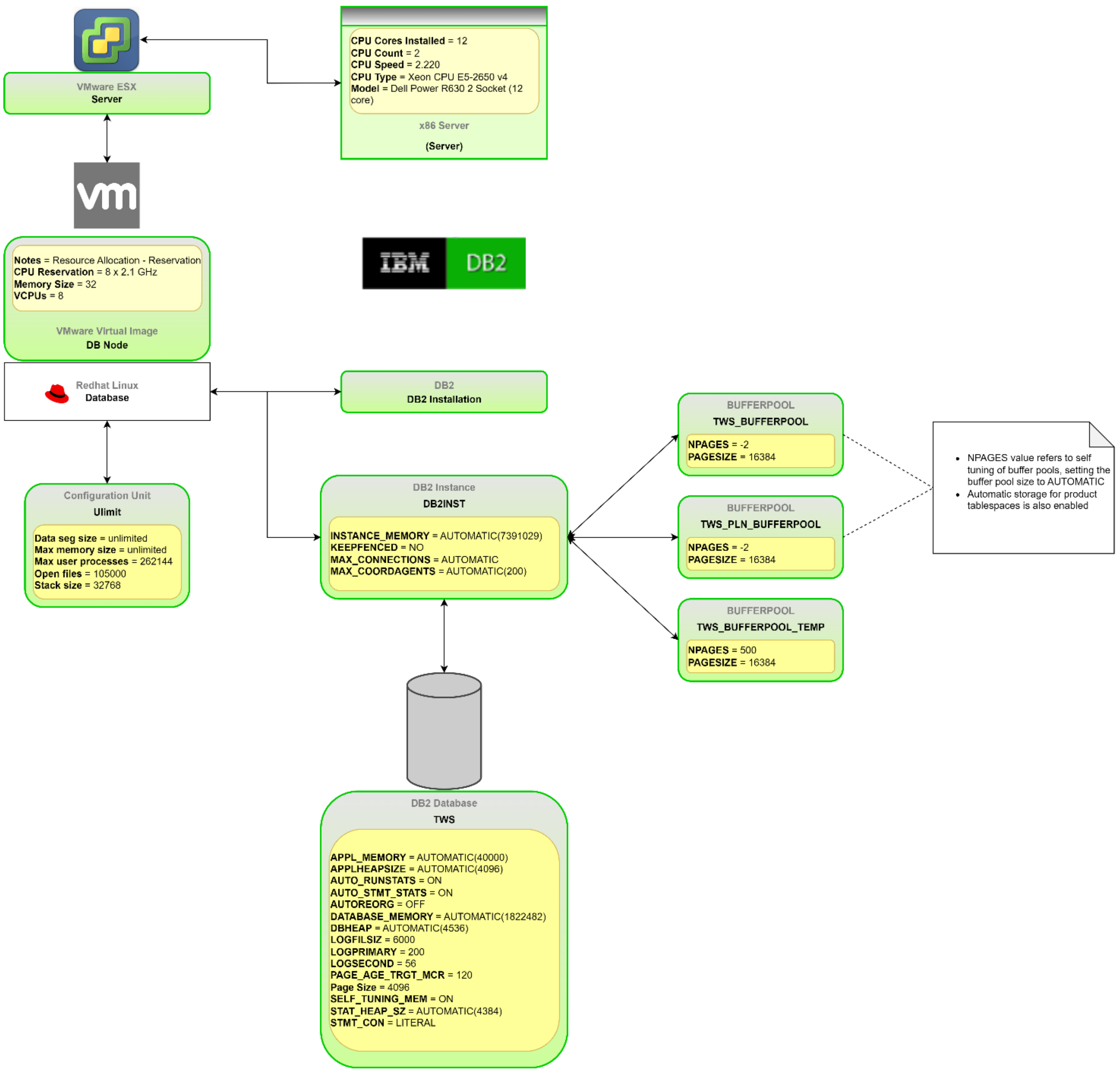
</dataSource>

com.ibm.tws.planner.monitor.cachemaxage = 21600000
com.ibm.tws.planner.monitor.cachesize = 70000
com.ibm.tws.planner.monitor.filecachesize = 40000
com.ibm.tws.planner.monitor.subProcessors = 10

bm check dedline = 0
bm check file = 120
bm check status = 300
bm check until = 300
bm late every = 0
bm look = 5
bm read = 3
bm stats = off
bm verbose = off

MaxAllocsInCache = 50000
MaxAllocsPerTimeSlot = 1000

Figure 3. Master Domain Manager node configurations



CPU Cores Installed = 12
CPU Count = 2
CPU Speed = 2.220
CPU Type = Xeon CPU E5-2650 v4
Model = Dell Power R630 2 Socket (12 core)

Notes = Resource Allocation - Reservation
CPU Reservation = 8 x 2.1 GHz
Memory Size = 32
VCPUs = 8

Data seg size = unlimited
Max memory size = unlimited
Max user processes = 262144
Open files = 105000
Stack size = 32768

INSTANCE_MEMORY = AUTOMATIC(7391029)
KEEPFENCED = NO
MAX_CONNECTIONS = AUTOMATIC
MAX_COORDAGENTS = AUTOMATIC(200)

TWS_BUFFERPOOL: **NPAGES** = -2
PAGESIZE = 16384

TWS_PLN_BUFFERPOOL: **NPAGES** = -2
PAGESIZE = 16384

TWS_BUFFERPOOL_TEMP: **NPAGES** = 500
PAGESIZE = 16384

- NPAGES value refers to self tuning of buffer pools, setting the buffer pool size to AUTOMATIC
- Automatic storage for product tablespaces is also enabled

APPL_MEMORY = AUTOMATIC(40000)
APPLHEAPSIZE = AUTOMATIC(4096)
AUTO_RUNSTATS = ON
AUTO_STMT_STATS = ON
AUTOREORG = OFF
DATABASE_MEMORY = AUTOMATIC(1822482)
DBHEAP = AUTOMATIC(4536)
LOGFILSIZ = 6000
LOGPRIMARY = 200
LOGSECOND = 56
PAGE_AGE_TRGT_MCR = 120
Page Size = 4096
SELF_TUNING_MEM = ON
STAT_HEAP_SZ = AUTOMATIC(4384)
STMT_CON = LITERAL

Figure 4. Database node configurations

Note: the same Virtual Machine configuration (CPU reservation = 8 x 2.1 GHz, RAM reservation: 32GB) has been applied to the Virtual Machine hosting Oracle 19c Server (second Performance Test Environment) without any further customization related to Oracle 19c configuration.

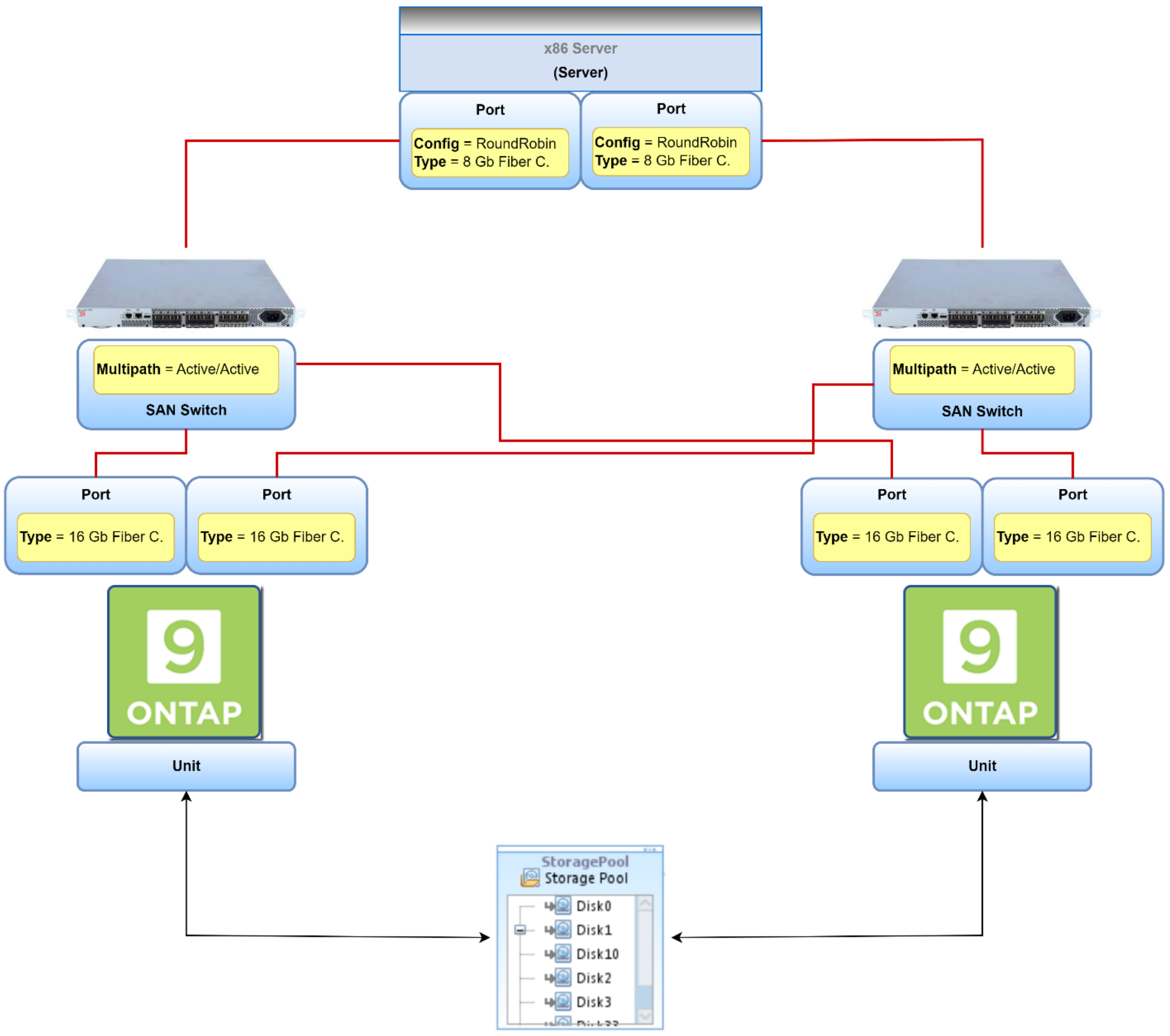


Figure 5. Storage Solution

3.3 Test Benchmarks and Results

3.3.1 Scheduling Workload

This section reports the details of the workload included in the daily production plan deployed in the Performance Test environments. The workload is distributed among fault tolerant and dynamic agents. The total number of jobs that are executed daily is around 550000 jobs per day.

The scheduling objects (such as jobs, job steams, workstations and others used in the daily workload) have been organized in a hierarchy of folders to validate that this feature does not determine any unexpected performance impacts on product scheduling capabilities.

Standard FTA and Dynamic Agent schedule
• Plan includes **124800 jobs scheduled in around 3 hours**. In particular, there are **52000 jobs** scheduled to be executed during a 10-minute peak. In addition, only for dynamic agents, around **361000** jobs are scheduled in 5 hours (1200 jobs/min).
WSA - Critical path
• 48 complex patterns, composed of multiple linked job streams (4) with 10 jobs each. 4 jobs for each complex pattern are defined as critical jobs
EDWA
• **200 TWS-Objects rules** - each rule matches a workstation and job name belonging to the daily production plan mentioned above and the success state of job execution. The action, in case of event matching, is to create a new message log. Normally, at the end of each test run, 4140 events (Message loggers) are generated. • **File-created rules** -These event monitor rules generate a specific message logger each time a new file with a predefined naming convention is created on each agent. In total, 240 events (message loggers) were generated each hour, which means 1 event every 4 minutes on each of the 16 agents.
Conditional Dependencies
• 5% of additional workload, an additional 3200 jobs/800 job streams over 4 dynamic agents and 4 FTA. This means that there are 100 job streams for each agent, half of which have internal dependencies and the other half has external dependencies. In the case of conditional dependencies, there are also 800 join conditions overall.
Ad Hoc Submission (conman sbs)
• Dynamic submission of jobs using the command "conman sbs" to submit a job stream with 20 different jobs (5 per agent) with dependencies between them in a chain. In total, 1000 dynamic jobs were submitted in 10 minutes, from 16:40 to 16:50.
File Dependencies
• 35000 job streams with a single job definition and a single file dependency defined at job stream level, distributed across the 8 FTAs present in the test environment (4375 jobs per agent). These 35000 job streams have a time dependency that is different for each FTA: the single block of 4375 job streams per agent is scheduled to start one hour after the preceding one for a duration of 8 hours long.
File Transfer
• **SSH protocol** - 1 File Transfer job scheduled to run every hour from 16:15 until 20:15 (5 instances per day). This File Transfer job is configured to copy 10 files x 1GB size from a SSH Server to a dynamic agent. • **Workstation-to-Workstation protocol -** 3 File Transfer jobs configured to copy from a dynamic agent to another dynamic agent respectively 5 files x 1GB size (6 instances per day starting from 04:00 until 04:55, one instance each 11 minutes); 50 files x 100KB size (6 instances per day starting from 06:00 until 06:55, one instance each 11 minutes); 100 files x 1MB size (6 instances per day starting from 08:00 until 08:55, one instance each 11 minutes).

Table 2. Daily plan workload composition

This workload is used as a standard benchmark for establishing key performance indicators whose baseline is continuously verified to track performance enhancements over time and product releases.

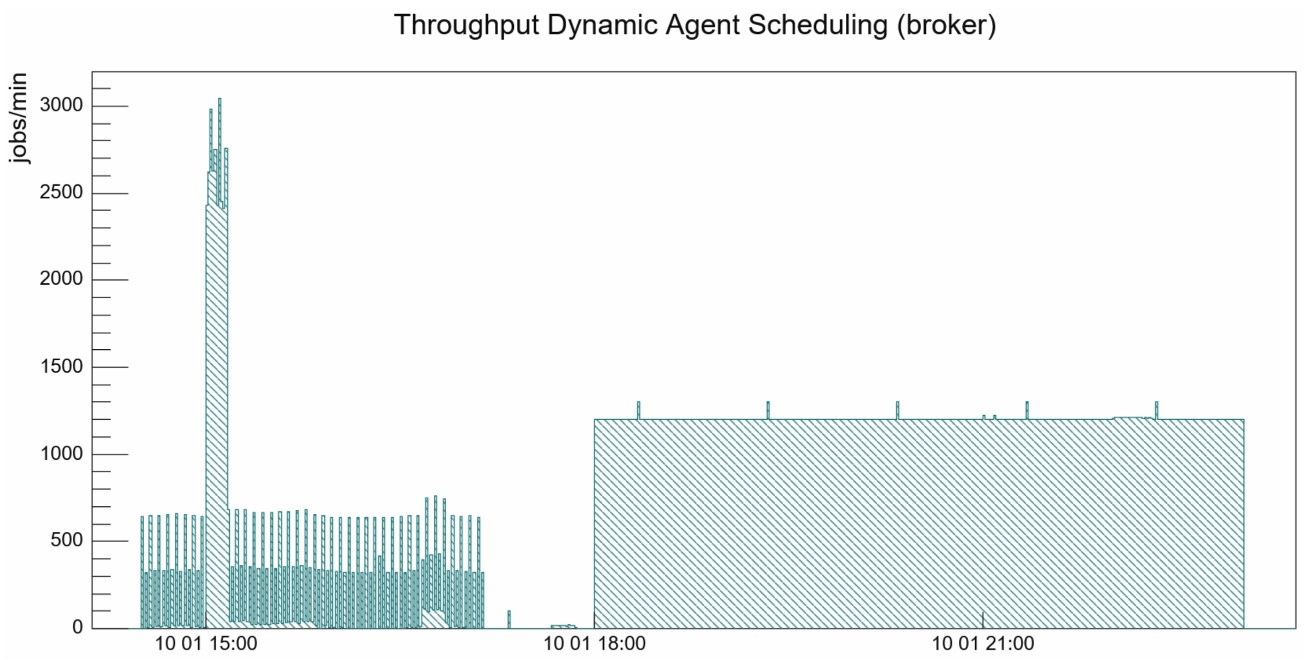


Figure 6. Dynamic agent throughput in daily workload

The workload represented in Figure 6 is the daily load of dynamic agent scheduling. In particular, the jobs from 18:00 to 23:00 are standard schedules with "every" option. The workload from 14:30 to 17:10 has different components, as can be observed from the zoomed view of Figure 7.

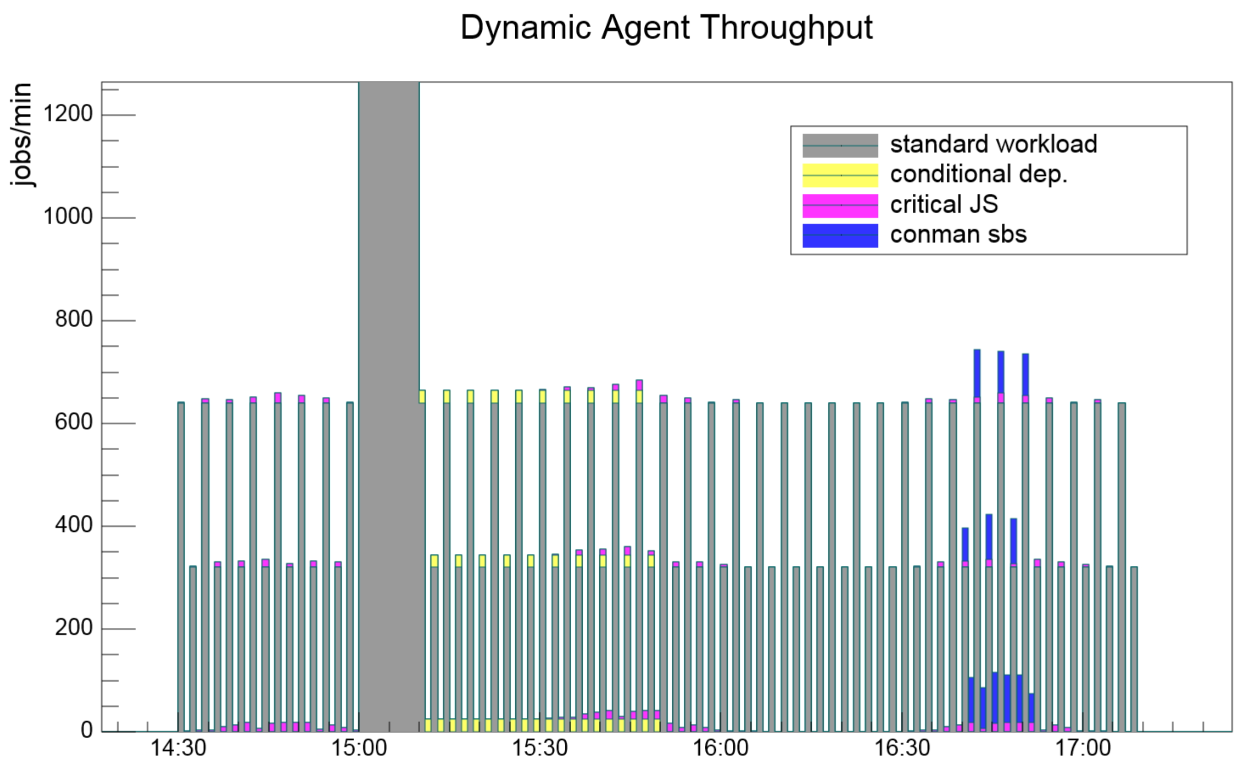


Figure 7. Zoomed view of dynamic agent scheduling

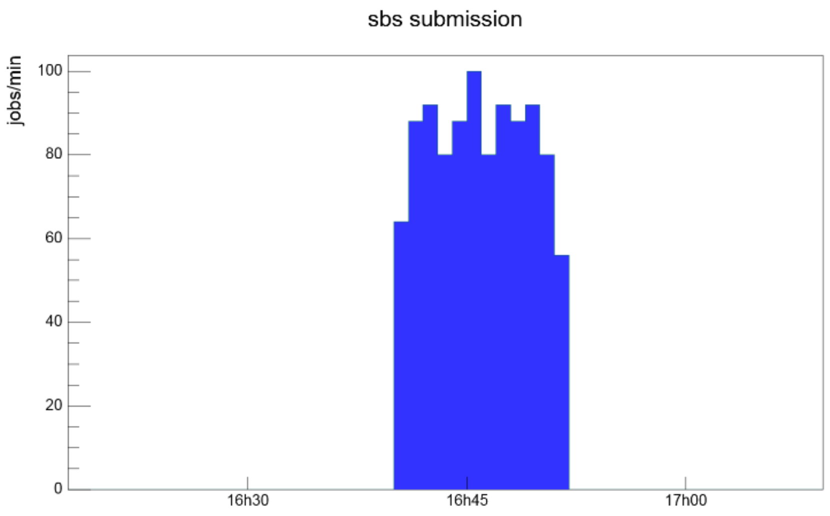


Figure 8. Dynamic agent schedule: ad hoc submission with 1000 scheduled jobs

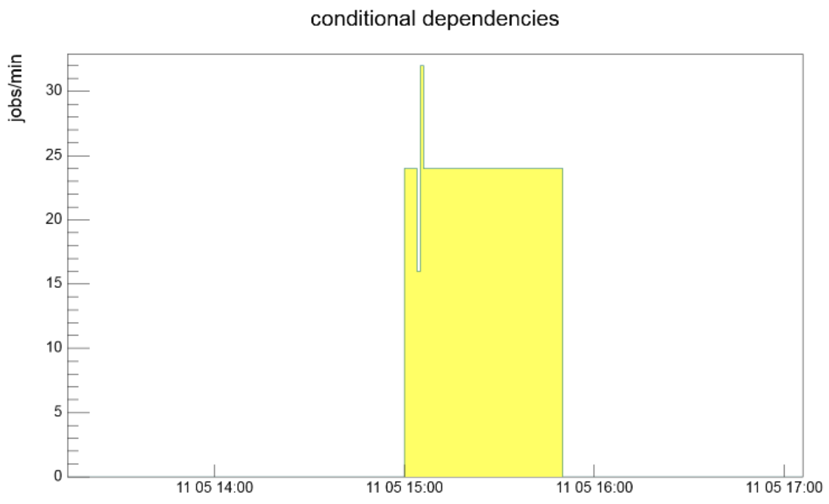


Figure 9. Dynamic agent schedule: jobs with internal and external conditional dependencies (3200 jobs)

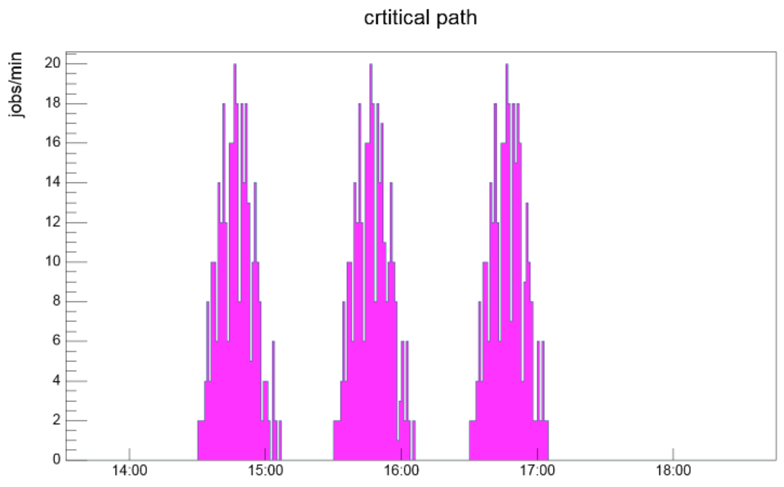


Figure 10. Dynamic agent schedule: jobs belonging to a critical path (Workload Service Assurance)

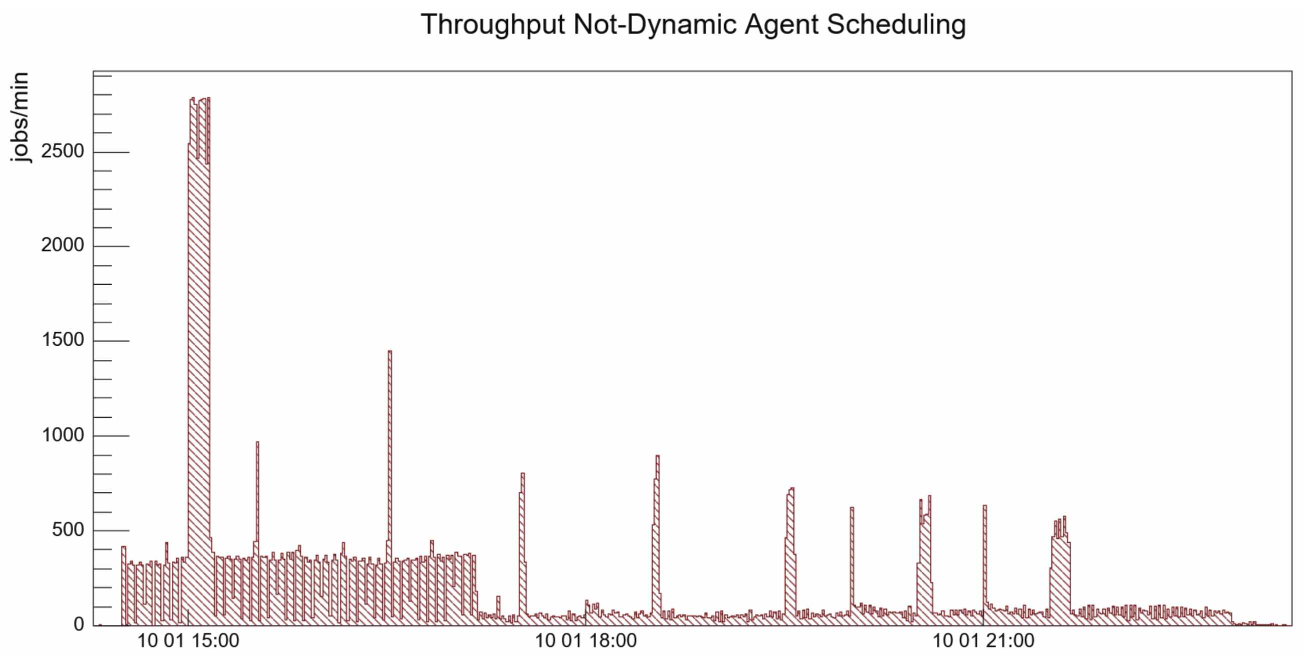


Figure 11. Fault tolerant agent schedule throughput in daily workload

The scheduling throughput represented in this section did not suffer any queuing phenomenon (incoming and outgoing throughput are equivalent).

In fact, the throughput analysis reconfirms the performance and scalability levels assured in previous releases. From the workload scheduler user perspective, that means, for instance, no substantial delay in the scheduling of a job when the same job is ready to start (see Figure 12) and no substantial latency in the job and job stream status update on the Dynamic Workload Console (see Figure 13).

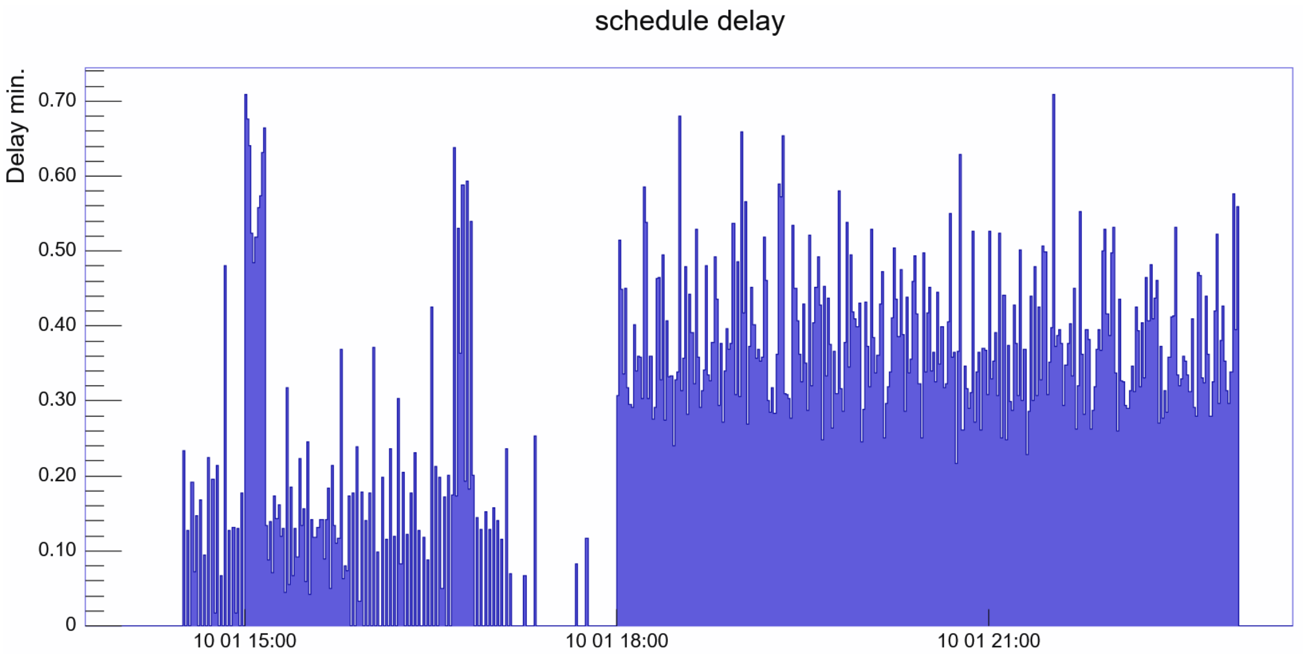


Figure 12. Average job schedule delay per minute over time (jobs scheduled on dynamic agents)

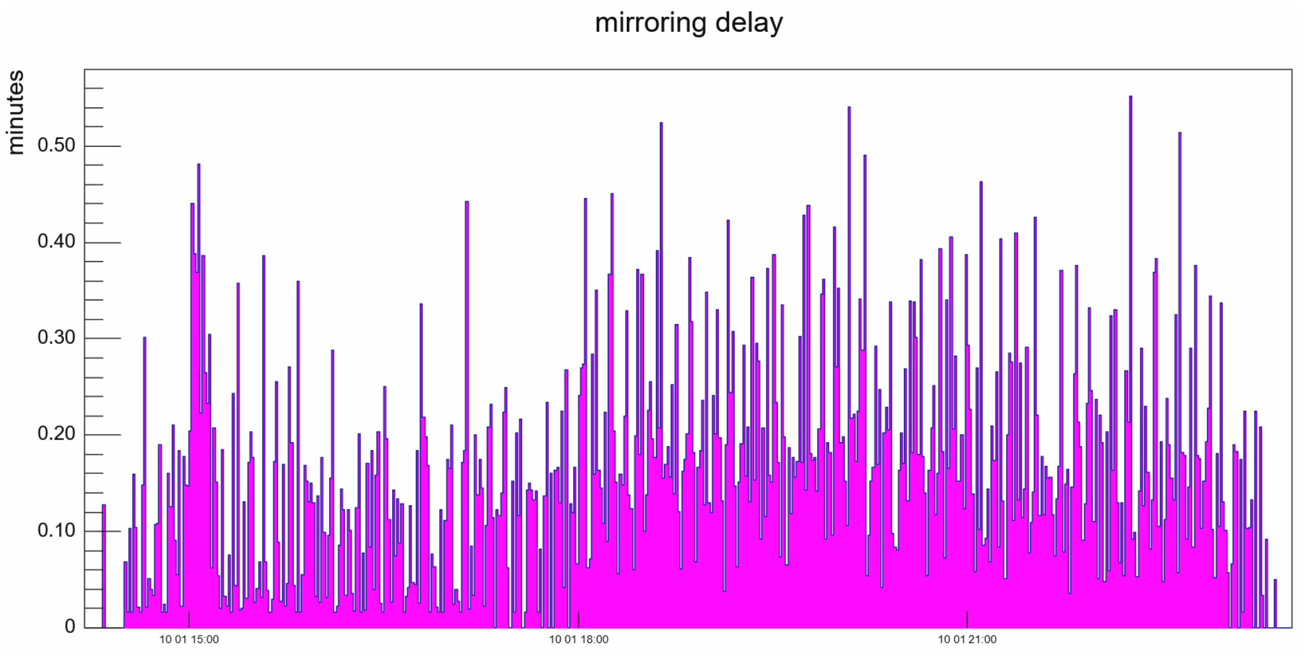


Figure 13. Job plan status update delay over time

The above results evidence the promptness of the Workload Scheduler, even in case of an intensive workload (2600 jobs/min both for dynamic and fault tolerant agents), with a dynamic scheduling delay of less than 1 minute. It must be remarked, once again, that these results must be correlated with the test environment and the workload discussed in this context; nevertheless, they could be considered as references while planning a Workload Scheduler deployment.

Figure 14 shows the CPU resource utilization trend on the MDM machine with respect to the outcoming throughput for the dynamic agent and fault tolerant agent scheduling as described in section 3.3.1. It is interesting to underline how the MDM CPU footprint for 10.1 and 9.5 product versions are similar, even if 10.1 version has added new functionalities, which guarantee greater security enforcements:

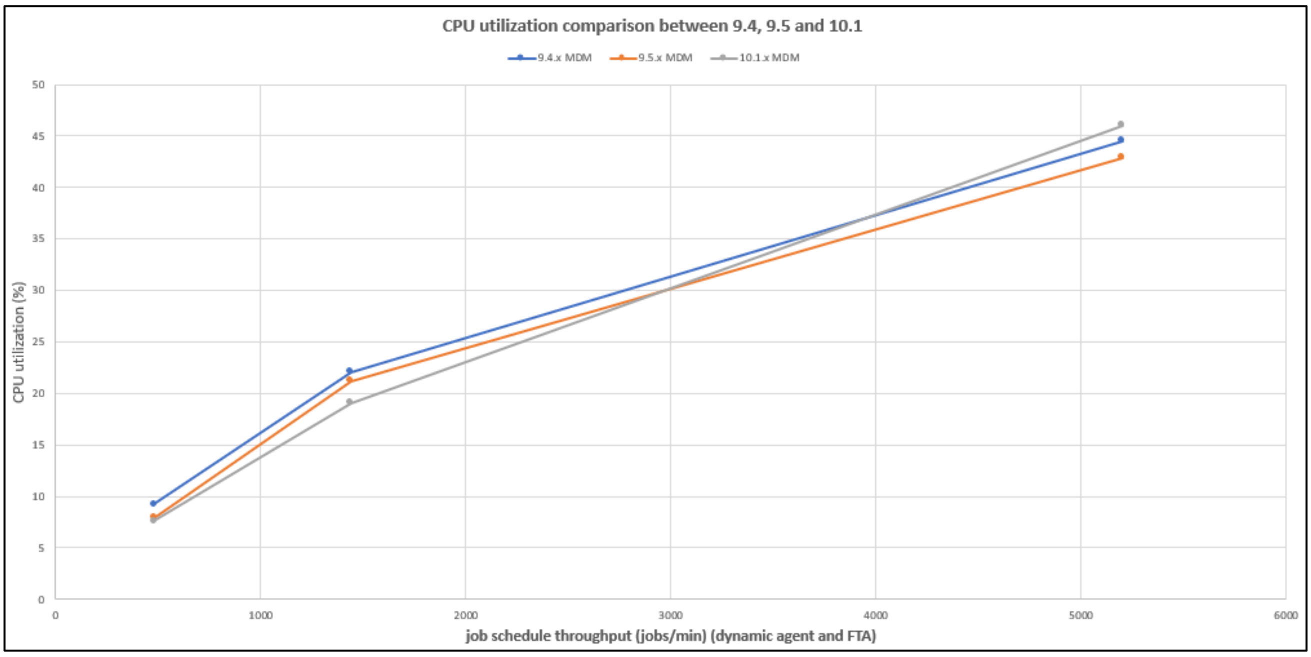


Figure 14. Total average CPU utilization at Master Domain Manager vs different workload

3.3.2 Automatic encryption and full SSL enablement for Fault Tolerant Agents

As already mentioned in section 1.1 of this document, starting from version 10.1, the security of the Workload Scheduler product has been significantly enforced thanks to a couple of new features:

- Automatic encryption at rest for key product files (such as the Symphony file, messages queues, and the `useropts` file)
- Automatic SSL configuration for fault-tolerant agents to ensure encryption in motion for all data moving within Workload Scheduler environments.

These two features have been enabled in one of the two Performance Test Environments (more specifically, the environment with Oracle 19c as RDBMS) to understand how those configuration changes might affect or not the overall performance and scalability of the product, if compared with the previous releases.

After having analyzed the test results collected during 10.1 and 10.1.0.1 project lifecycles, the main outcomes are that there are very minimal performance impacts on product throughput capabilities (e.g. average job schedule delay per minute for those jobs scheduled versus dynamic agents increased for few seconds) and on Master Domain Manager CPU utilization during the jobs submission peak, but these minimal performance impacts are strongly balanced out by the benefits of having enabled a more secure configuration for the product.

In any case, from the Workload Scheduler user perspective, no substantial delay in the scheduling of a job when the same job is ready to start and no substantial latency in the job and job stream status update on the Dynamic Workload Console have been observed during this specific test assessment.

3.3.3 Dynamic Workload Console Workload

The Dynamic Workload Console (DWC) has been tested to verify that response times under load are within expected values.

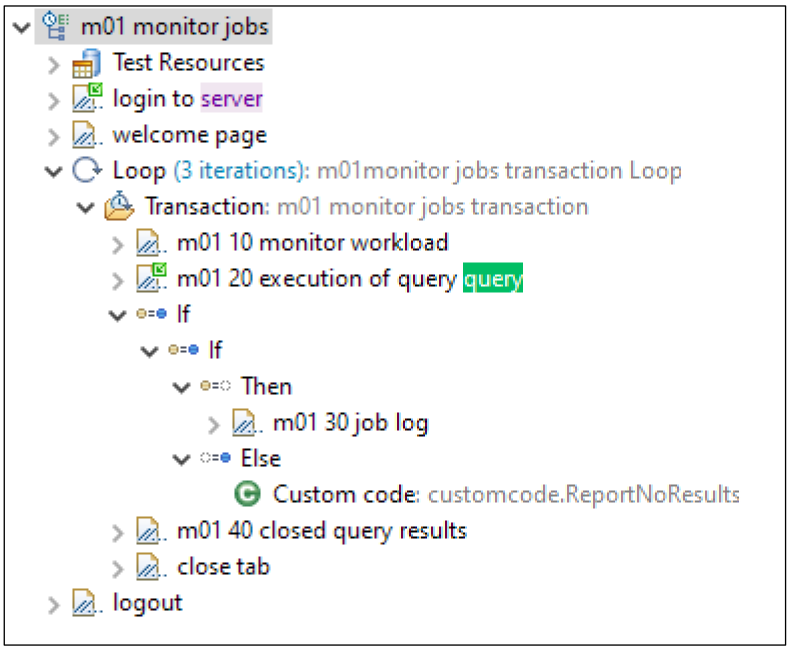
Load tests have been executed against two DWC instances (see Figure 2 for details on single DWC instance configuration) configured to point to the same database, served by DB2 version 11.5.8 running on RHEL 8. Requests have been distributed evenly between the two instances, directly by the load test generator (OneTest Performance), without using any load balancer.

The following table summarizes the user scenarios used to generate the workload, with their fraction of virtual users and iteration rates applied during execution. Different test runs have been executed by varying the iteration rate between the values reported in the table, while keeping constant the total number of virtual users (300).

Test case name	Description	virtual users	Iterations per hour (for each user)
m01 monitor jobs	Monitor workload: run a job direct query and get job log for a completed job. Figure 15	40% (120)	12, 14, 16, 18, 20
m02 plan view	Monitor workload: display job stream from plan view. Figure 16	10% (30)	
m03 monitor ERs	Monitor workload: display event rule and triggered action. Figure 17	20% (60)	
d01 list and view JS	New workload designer: list all job streams, view one of them. Figure 18	9% (27)	
d02 list and view job	New workload designer: list all jobs, view one of them. Figure 19	9% (27)	
d03 list and view ER	New workload designer: list all event rules, view one of them. Figure 20	9% (27)	
d04 create edit delete job	New workload designer: create, modify, and then delete a job. Figure 21	3% (9)	6

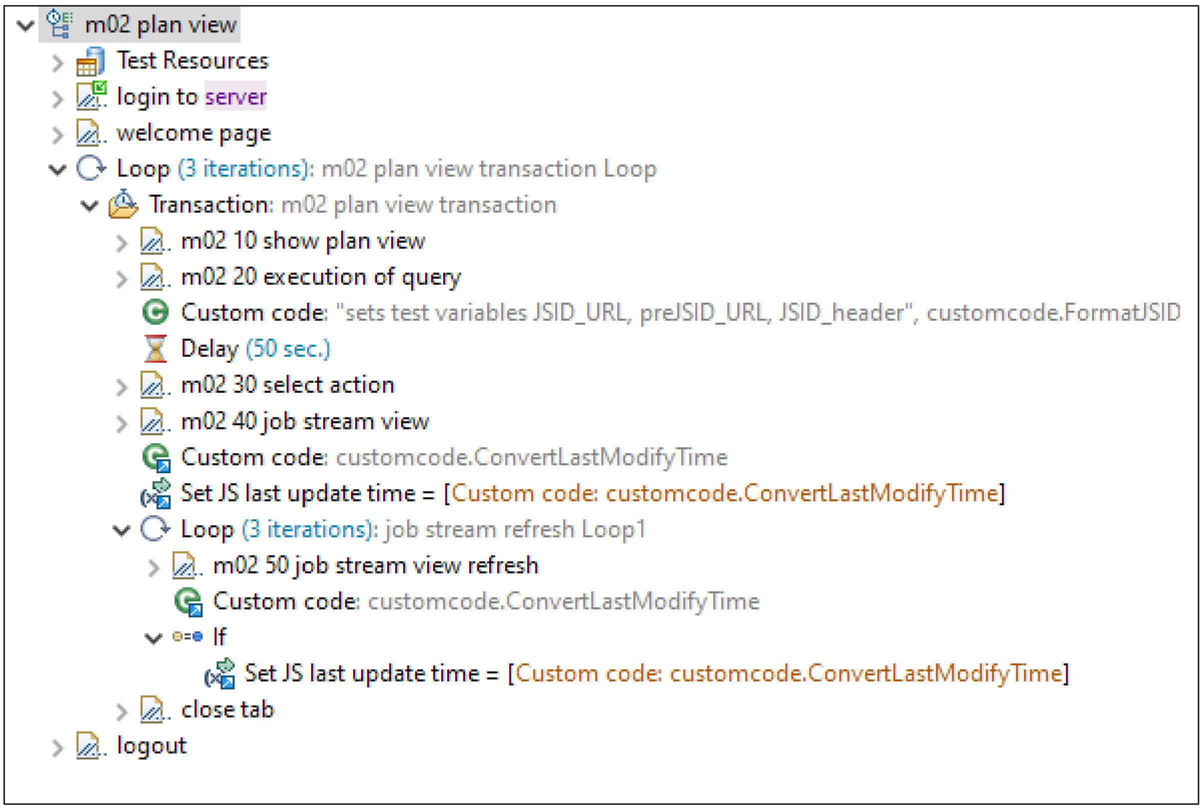
Table 3. DWC test cases

The following figures (15 to 21) show how the seven test cases have been implemented, using OneTest Performance, in terms of "HTTP pages" and flow control.



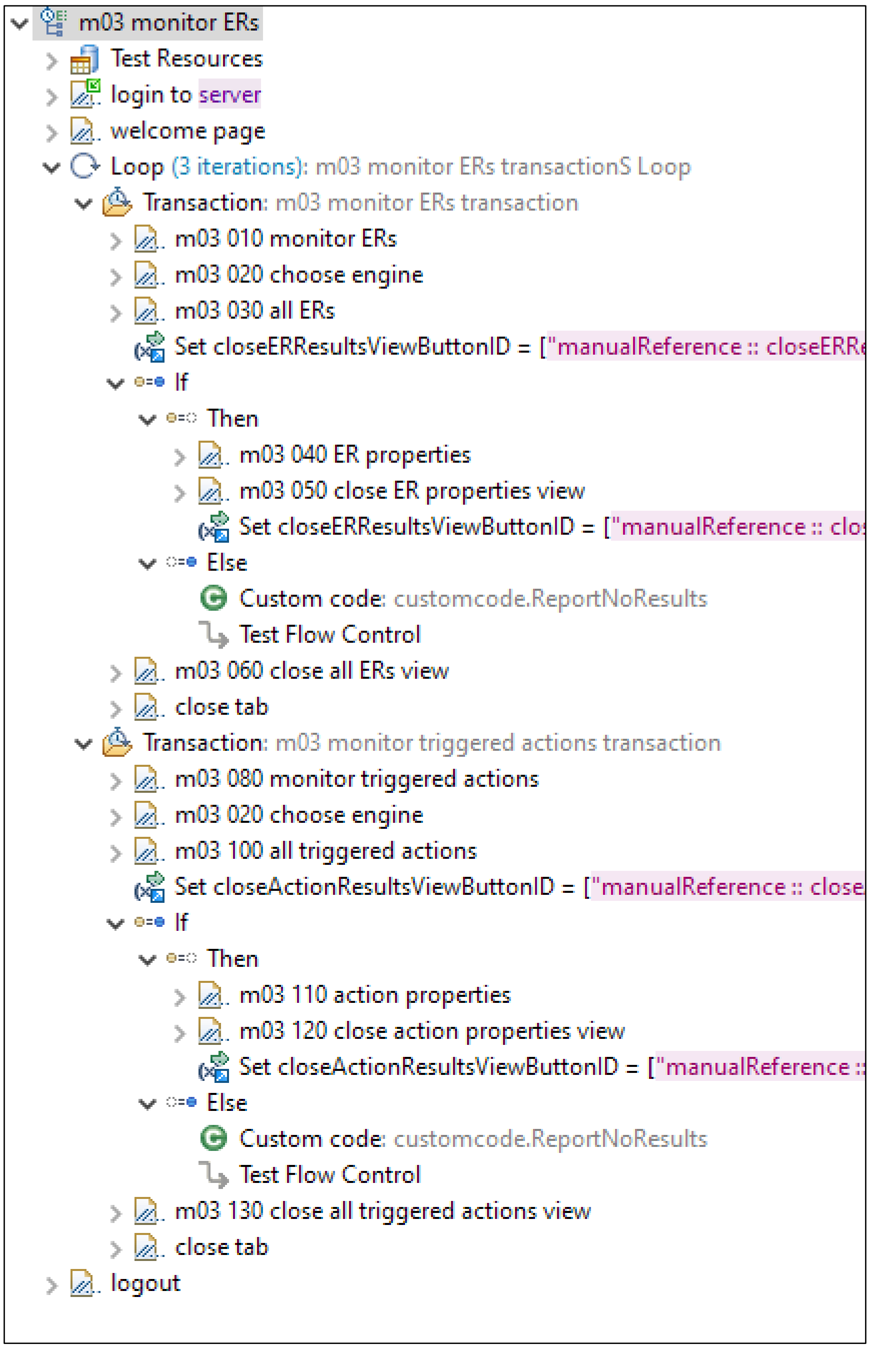
```
m01 monitor jobs
  Test Resources
  login to server
  welcome page
  Loop (3 iterations): m01monitor jobs transaction Loop
    Transaction: m01 monitor jobs transaction
      m01 10 monitor workload
      m01 20 execution of query query
      If
        If
          Then
            m01 30 job log
          Else
            Custom code: customcode.ReportNoResults
      m01 40 closed query results
      close tab
  logout
```

Figure 15. Monitor workload: run a job direct query and get job log for a completed job.



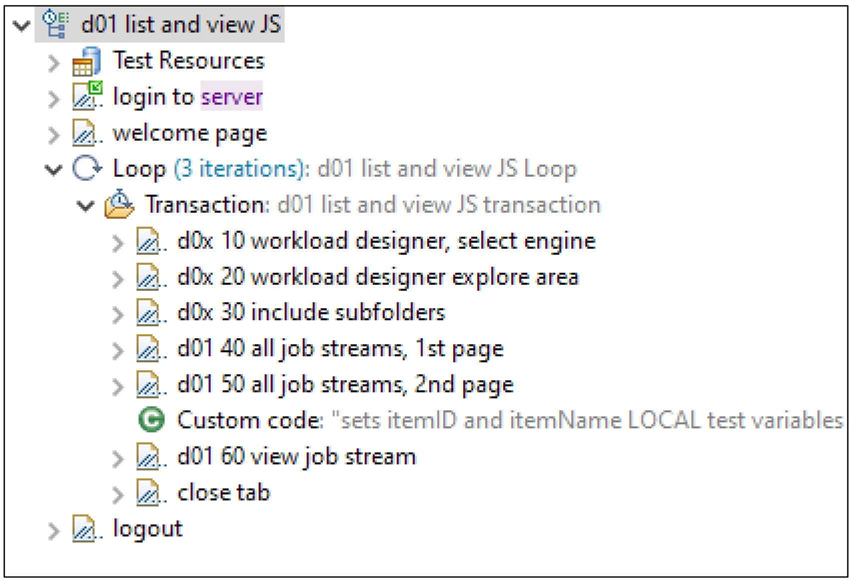
```
m02 plan view
  Test Resources
  login to server
  welcome page
  Loop (3 iterations): m02 plan view transaction Loop
    Transaction: m02 plan view transaction
      m02 10 show plan view
      m02 20 execution of query
      Custom code: "sets test variables JSID_URL, preJSID_URL, JSID_header", customcode.FormatJSID
      Delay (50 sec.)
      m02 30 select action
      m02 40 job stream view
      Custom code: customcode.ConvertLastModifyTime
      Set JS last update time = [Custom code: customcode.ConvertLastModifyTime]
      Loop (3 iterations): job stream refresh Loop1
        m02 50 job stream view refresh
        Custom code: customcode.ConvertLastModifyTime
        If
          Set JS last update time = [Custom code: customcode.ConvertLastModifyTime]
      close tab
  logout
```

Figure 16. Monitor workload: display job stream from plan view.



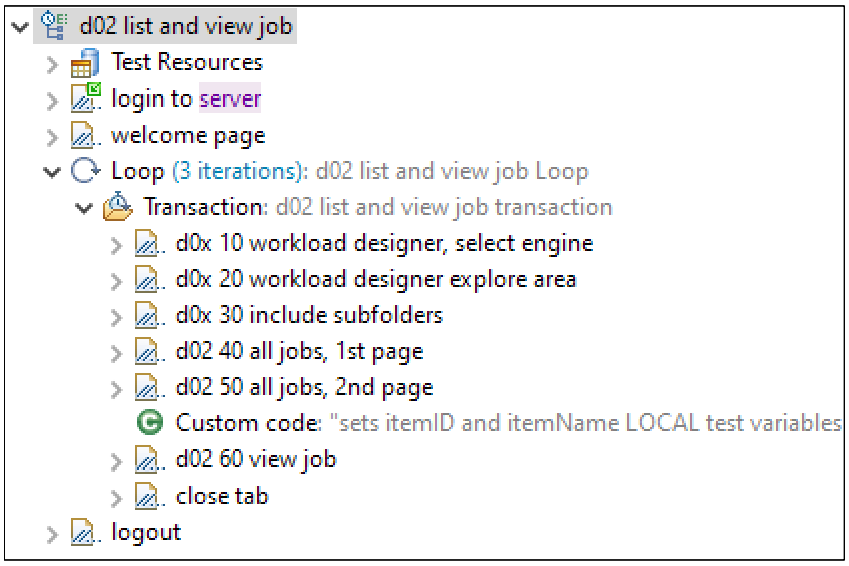
```
m03 monitor ERs
  Test Resources
  login to server
  welcome page
  Loop (3 iterations): m03 monitor ERs transactionS Loop
    Transaction: m03 monitor ERs transaction
      m03 010 monitor ERs
      m03 020 choose engine
      m03 030 all ERs
      Set closeERResultsViewButtonID = ["manualReference :: closeERRe
      If
        Then
          m03 040 ER properties
          m03 050 close ER properties view
          Set closeERResultsViewButtonID = ["manualReference :: clos
        Else
          Custom code: customcode.ReportNoResults
          Test Flow Control
      m03 060 close all ERs view
      close tab
    Transaction: m03 monitor triggered actions transaction
      m03 080 monitor triggered actions
      m03 020 choose engine
      m03 100 all triggered actions
      Set closeActionResultsViewButtonID = ["manualReference :: close
      If
        Then
          m03 110 action properties
          m03 120 close action properties view
          Set closeActionResultsViewButtonID = ["manualReference ::
        Else
          Custom code: customcode.ReportNoResults
          Test Flow Control
      m03 130 close all triggered actions view
      close tab
  logout
```

Figure 17. Monitor workload: display event rule and triggered action.



```
d01 list and view JS
  Test Resources
  login to server
  welcome page
  Loop (3 iterations): d01 list and view JS Loop
    Transaction: d01 list and view JS transaction
      d0x 10 workload designer, select engine
      d0x 20 workload designer explore area
      d0x 30 include subfolders
      d01 40 all job streams, 1st page
      d01 50 all job streams, 2nd page
      Custom code: "sets itemID and itemName LOCAL test variables
      d01 60 view job stream
      close tab
  logout
```

Figure 18. New workload designer: list all job streams, view one of them.



```
d02 list and view job
  Test Resources
  login to server
  welcome page
  Loop (3 iterations): d02 list and view job Loop
    Transaction: d02 list and view job transaction
      d0x 10 workload designer, select engine
      d0x 20 workload designer explore area
      d0x 30 include subfolders
      d02 40 all jobs, 1st page
      d02 50 all jobs, 2nd page
      Custom code: "sets itemID and itemName LOCAL test variables
      d02 60 view job
      close tab
  logout
```

Figure 19. New workload designer: list all jobs, view one of them.

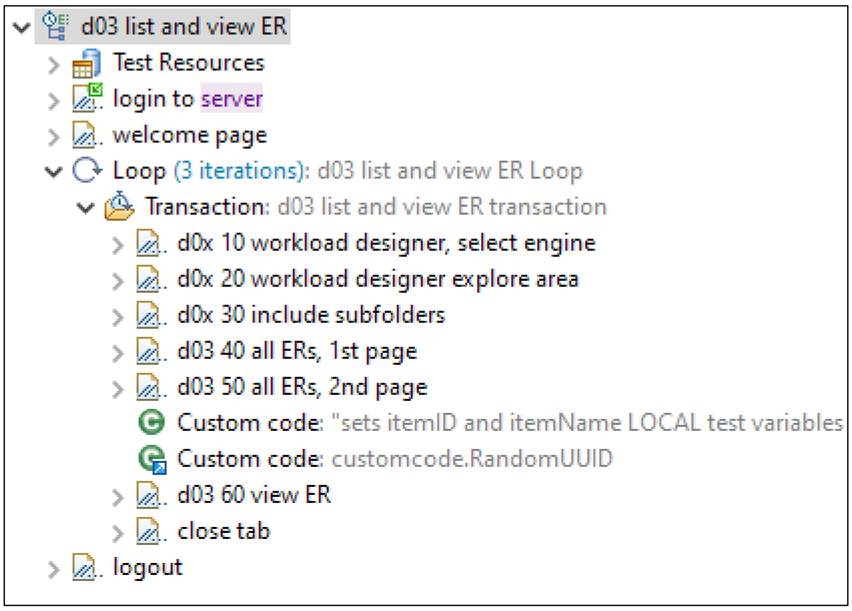

```
d03 list and view ER
  Test Resources
  login to server
  welcome page
  Loop (3 iterations): d03 list and view ER Loop
    Transaction: d03 list and view ER transaction
      d0x 10 workload designer, select engine
      d0x 20 workload designer explore area
      d0x 30 include subfolders
      d03 40 all ERs, 1st page
      d03 50 all ERs, 2nd page
      Custom code: "sets itemID and itemName LOCAL test variables
      Custom code: customcode.RandomUUID
      d03 60 view ER
      close tab
  logout
```

Figure 20. New workload designer: list all event rules, view one of them.

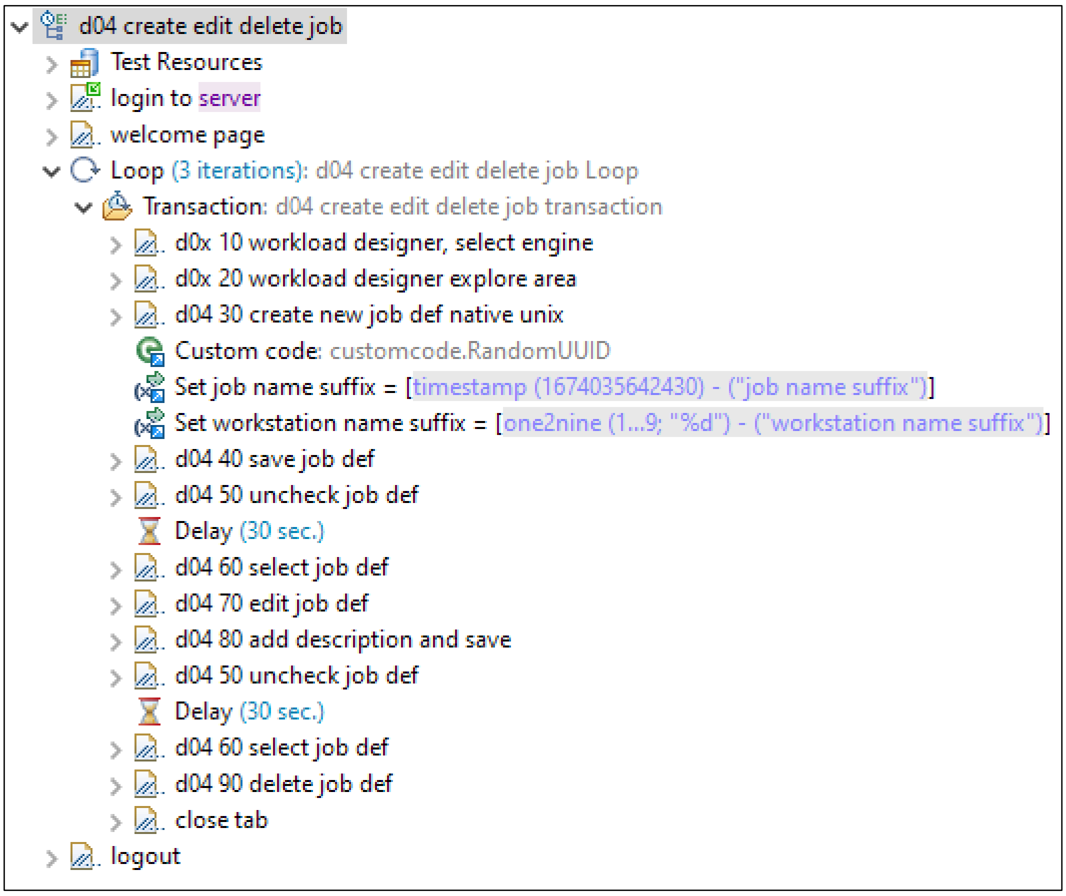

```
d04 create edit delete job
  Test Resources
  login to server
  welcome page
  Loop (3 iterations): d04 create edit delete job Loop
    Transaction: d04 create edit delete job transaction
      d0x 10 workload designer, select engine
      d0x 20 workload designer explore area
      d04 30 create new job def native unix
      Custom code: customcode.RandomUUID
      Set job name suffix = [timestamp (1674035642430) - ("job name suffix")]
      Set workstation name suffix = [one2nine (1...9; "%d") - ("workstation name suffix")]
      d04 40 save job def
      d04 50 uncheck job def
      Delay (30 sec.)
      d04 60 select job def
      d04 70 edit job def
      d04 80 add description and save
      d04 50 uncheck job def
      Delay (30 sec.)
      d04 60 select job def
      d04 90 delete job def
      close tab
  logout
```

Figure 21. New workload designer: create, modify, and then delete a job.

Virtual Users have been ramped up at a rate of 1 user every 6 seconds. Once the target number of virtual users (300) was reached, the load was held fixed for a minimum of 1 hour.

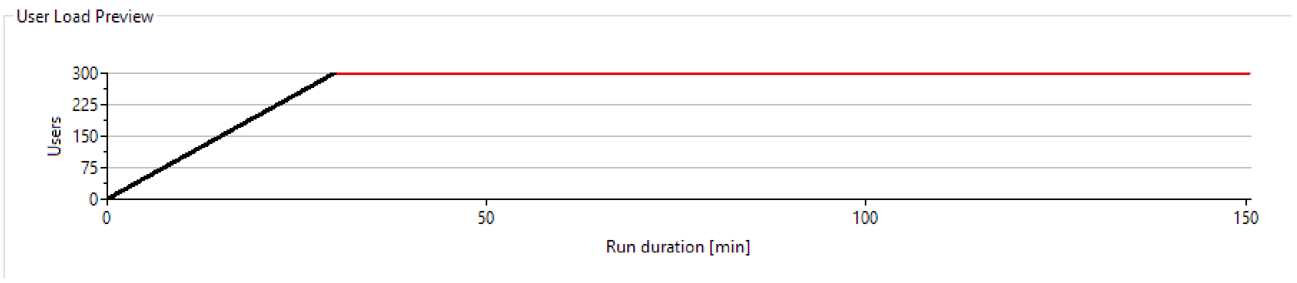


Figure 22. Virtual User load profile.

Response times and page hit rates have been measured during the fixed load time range.
Response times are measured as the difference between when a response arrives at the load generation tool and when the corresponding request has been issued. They do not include any processing time that might occur in the browser because OneTest Performance does not process any browser-specific code, like JavaScript or CSS.

The test runs, performed with different load levels (iteration rate), resulted in a total page hit rate in the range 10-16 HTTP page requests per second, with corresponding overall average response times for all pages in the range 350-700 milliseconds.

The following figure shows the page hit rate over time, as reported by OneTest Performance, for a test run with 300 virtual users and 20 iterations per hour.

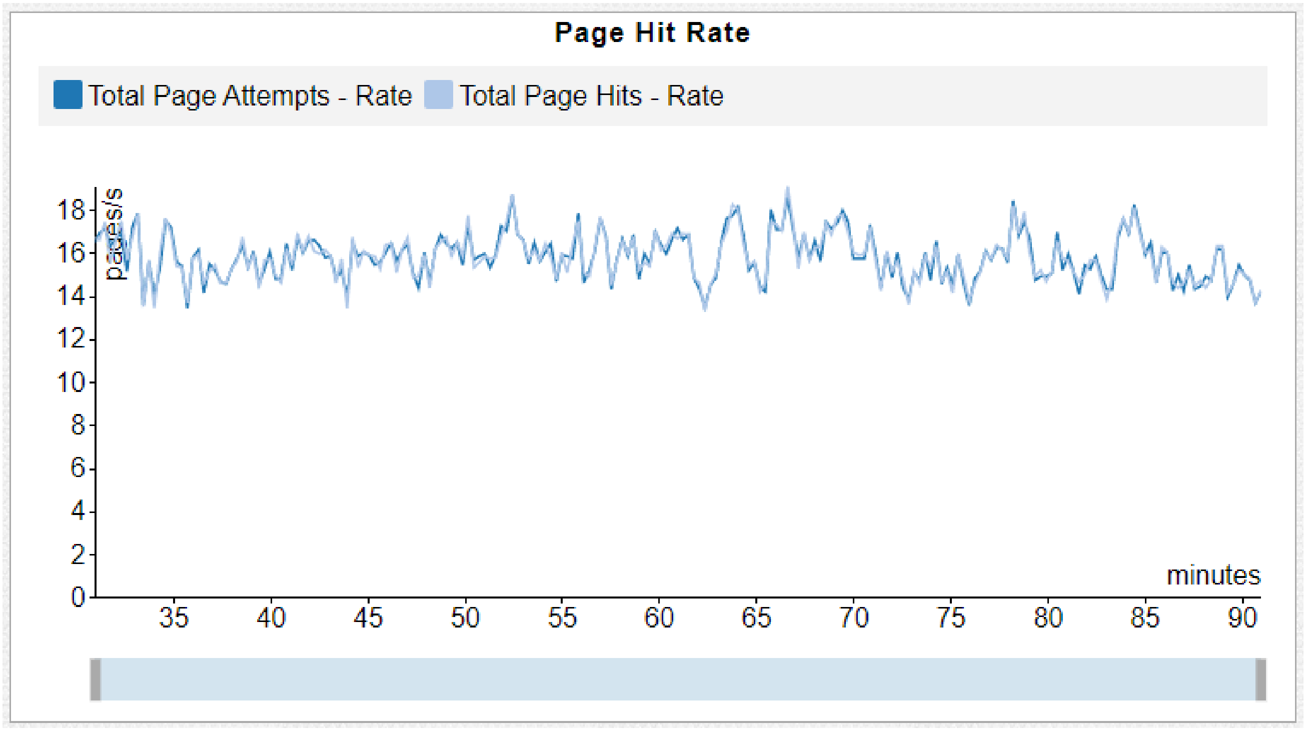
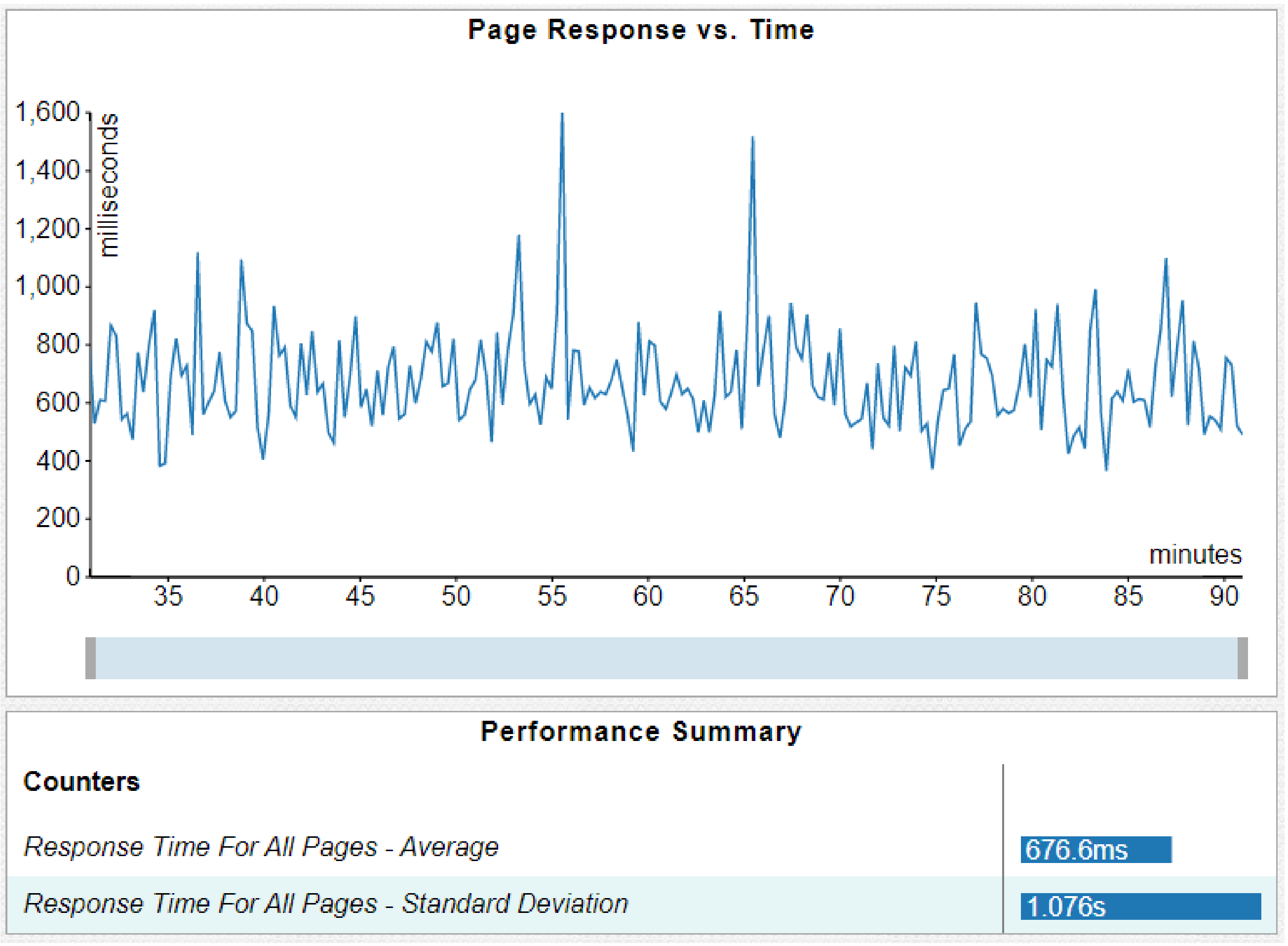


Figure 23. Page request rate for test run with 300 users and 20 iterations/hour.

The next figure shows the response time over time for all pages for the same test run.



Performance Summary

Counters	
Response Time For All Pages - Average	676.6ms
Response Time For All Pages - Standard Deviation	1.076s

Figure 24. Average response time over time for all pages.

For a test environment with around 43,000 job streams and 500,000 jobs in plan each day, the maximum response times have been observed for the monitoring jobs direct queries (step named “m01 20 execution of query” in Figure 15), with average response times within 5 seconds.

All measurements have been collected while the scheduling workload (job streams execution) on the back end (Master Domain Manager) was negligible.
It is expected that, if the same DWC workload is applied while the MDM is busy executing job streams, the DWC response times could be significantly higher than those reported above, and that main backend KPIs (scheduling and mirroring delays) also would be affected.

3.3.4 Workstation-to-Workstation internal File Transfer protocol

As already mentioned in section 1.1 of this document, starting from 10.1 version, the support of the Workstation-to-Workstation internal protocol for file transfers job application plugin has been added. Whitin this new feature, it is possible to transfer files to and from agents connected to the same Master Domain Manager through a file proxy, which is installed by default on the Master Domain Manager machine. This new feature provides a higher flexibility to Workload Scheduler customers to move files from an agent to another, without using a SSH Server as a proxy to implement the same user scenario, as described in the next figure:

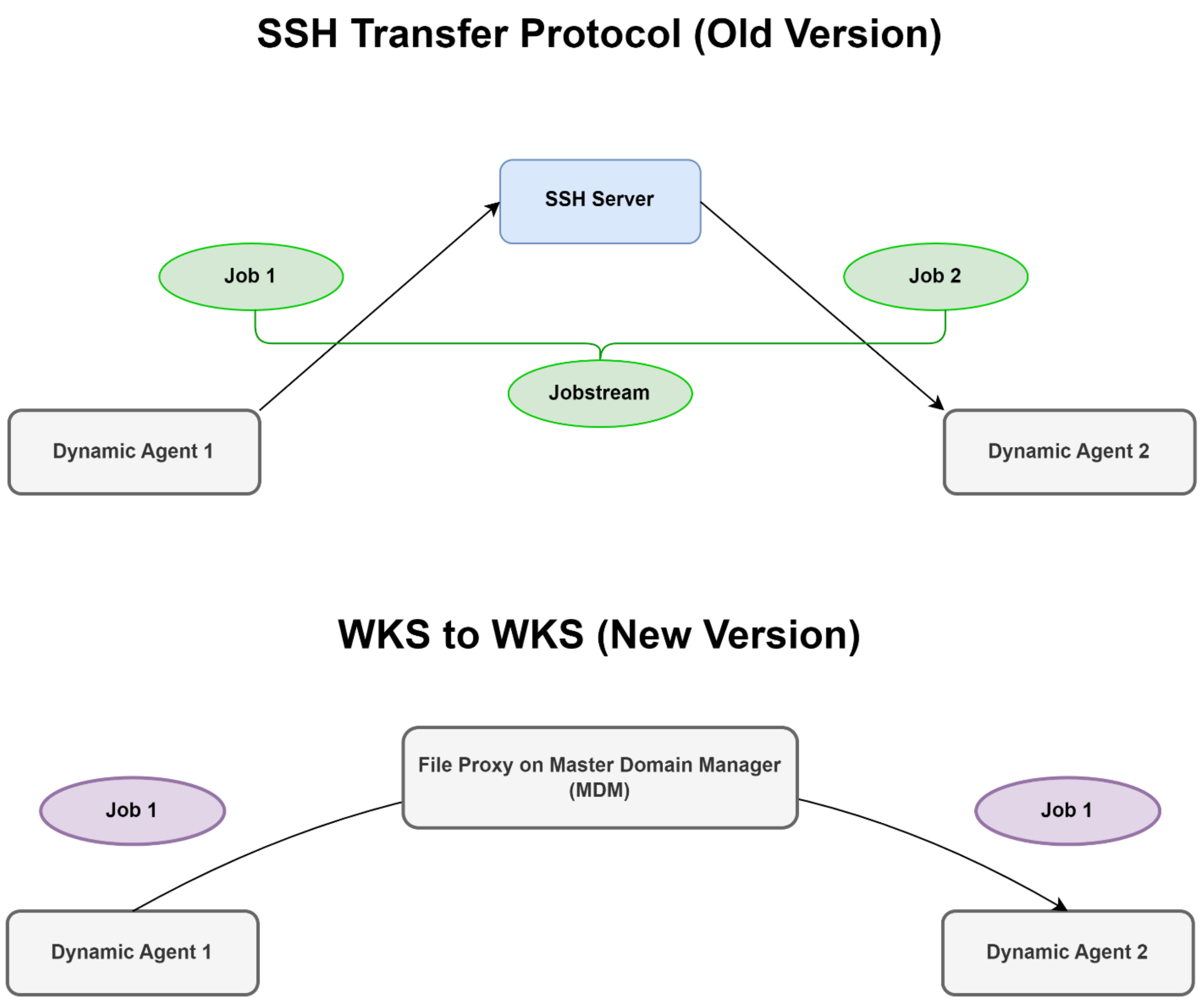


Figure 25. File Transfer scenario implementation using Workstation-to-Workstation protocol

During the 10.1 and 10.1.0.1 project lifecycles, one of the main objectives was to demonstrate that the new feature of «workstation-to-workstation internal file transfer protocol» was able to guarantee a faster file transfer scenario if compared with the previous implementation (file transfer from an agent to an SSH Server and then from the SSH Server to another agent). This performance assessment has been performed considering three different scenarios:

1. 5 files with a size of 1GB each.
2. 50 files with a size of 100KB each.
3. 100 files with a size of 1MB each.

In all these cases, a significant reduction in the File Transfer execution time has been measured when the new Workstation-to-Workstation protocol was used, as documented by the table below:

File Transfer Scenario	File Transfer execution time reduction when Workstation-to-Workstation protocol is used
Scenario #1: 5 files with size 1GB each	-27%
Scenario #2: 50 files with size 100KB each	-43%
Scenario #3: 100 files with 1MB each	-34%

Table 4. File Transfer execution time comparison between SSH and Workstation-to-Workstation protocols

As mentioned before, a new component, named File Proxy, has been added starting from version 10.1, and by default, this component is installed on each Master Domain Manager. This File Proxy is used to securely manage files to be transferred with the File Transfer integration. All transfer operations are performed by Workload Scheduler, with no need of third-party products.

The File Proxy can be optionally installed as a stand-alone component on a workstation different from the Master Domain Manager, for example, to reduce network traffic and resource usage on the Master Domain Manager. This configuration is suggested if the File Transfer integration is used for hundreds of jobs defined in the daily production plan.

4 Best Practices

4.1 Scheduling

Workload Scheduler software offers many a lot of features to perform at best its own objective: orchestrate scheduling flow activities. The principal way to schedule is to have Job Streams included in the plan, by associating it to a run cycle, during the plan generation activity. In addition, the schedule of Jobs and Job Streams could occur dynamically while the plan is running, using, for example, event rules, conman, start conditions, and file dependencies. Even if the latter gives a higher level of versatility to accomplish different business scenarios, some recommendations that must be considered before planning to adopt them to orchestrate the scheduler completely the schedule in case of a heavy high workload.

4.1.1 Scheduling using event rules: Event Processor Throughput

It is possible to have rules that trigger actions like job and job stream submission. These rules could detect, for example, a job status change or file creation events. In all these cases, the events are sent to the event processor queue (`cache.dat`). In the case of a status change, the consumer is the `batchman` process, while in the case of remote file monitoring, the agent itself communicates with the event processor. In all these cases, the final submission throughput strictly depends on event processor throughput capability.

The event processor is a single thread process, and its processing capacity is proportional to the core speed and the I/O. For this reason, it cannot scale horizontally but vertically only by increasing the CPU and/or I/O capabilities.

Detailed considerations about Event Processor Throughput capabilities are available in the section 4.1.1 of Workload Scheduler 9.5.0.2 performance report.

4.1.2 Scheduling using file dependencies

Workload Scheduler allows the release of dependencies to perform scheduling. These releases could depend on several objects (jobs, job streams, resources). File dependency is often a useful feature to implement many business workflows that must be triggered by a file creation. The workload described in 3.3.1 already includes a component of this type acting against fault tolerant agents. This feature has a different impact on the performance if used with a dynamic agent. In the case of a dynamic agent, the entire mechanism is driven by the dynamic domain manager that is in charge of the continuous check on the file existence status. The polling period is driven by the `localopts` property present on the Dynamic Domain Manager:

`bm check file = 300` (120 seconds is the default).

It defines the frequency with which the dynamic agent is contacted by the server about file status. The server workload throughput is ruled by four parameters:

1. Polling period.
2. Number of file dependencies.
3. Network connection between agents and server.
4. Background Scheduling activities.

In the test environment, with a network latency around 0.3 ms, the file check throughput was strongly dependent on the scheduling workload. In this context, the period between two files checks (in the total file dependencies list) passes from few hundredths of second (during no scheduling activity) to tenth of second (during peak time), increasing the time to slide all the file dependencies every T (bm check file).

In a context like the actual one, it is suggested to tune the T (`bm check file)` with the following restriction:

$$T > \frac{N}{10}$$

being N the total number of file dependencies.

A practical example of the considerations above can be found in the section 4.1.2 of Workload Scheduler 9.5.0.2 performance report

4.1.3 Scheduling using conman sbs

The ”`conman sbs`” (or equivalent RESTful calls) command adds a job stream to the plan on the fly. If the network of the added job stream is significantly complex, both in terms of dependencies and cardinality, it could cause a general delay in the plan update mechanism. In this scenario, due to scheduling coherence, all the initial updates pass through the main thread queue (`mirrobox.msg`), bypassing the benefits of multithreading. It is extremely difficult to identify the complexity of the network that would cause this kind of queueing. In any case, the order of magnitude is of several hundreds of objects, considering both jobs in the job streams and internal and/or external dependencies.

4.1.4 Scheduling using “every” option

“Every” feature allows to create a new instance of a job stream or of a job in the plan, but while for the former case the impact is not relevant at run time because the instances are already included in the plan, the latter causes multiple internal “sbj” to the same job stream instance that could affect performances especially in the plan updates.

It is very important to verify if the “EVERY” option is used with minimal values (a few minutes) to avoid increasing the number of jobs in a job stream (a few hundred jobs could impact performances). Several methodologies could help to drive the business needs:

- Move EVERY option at job stream level if possible
- Split the Job stream in multiple job stream with “AT xx till xx” (time partitioning)

4.2 Dynamic domain manager table cleanup policy

While the workload increases in terms of the number of jobs executed per day on dynamic agents, the dynamic domain manager historical tables increase accordingly. Data persistency in this table allows performing job log retrieval for archived plans. The following parameters, in the `<TWA_DATA_DIR>/broker/config/JobDispatcherConfig.properties` file, define the cleanup policy:

- SuccessfulJobsMaxAge
- UnsuccessfulJobsMaxAge
- MoveHistoryDataFrequencyInMins

By default, the cleanup thread starts after the time specified by “`MoveHistoryDataFrequencyInMins`” lapses since the last occurrence completion or application server boots and removes jobs from the table accordingly to their status and age. If the job table is large (magnitude 10^6 rows), and the number of records to delete high (magnitude 10^5), this activity impacts Workload Schedule performance and throughput capabilities. For more details about negative performance impacts, refer to section 4.2 of Workload Scheduler 9.5.0.2 performance report.

To avoid the behavior described above, it could be suggested to handle the policy in a more controlled way. For instance, a specific job could be used to run the cleanup invoking the built-in Workload Scheduler script:

```
<INST_DIR>/TDWB/bin/movehistorydata.sh -successfulJobsMaxAge 240
```

The script could be executed on a job scheduled during the appropriate time window, when the daily jobs execution is low, as shown in the example below, a job stream named “CLEANUP_DWB_DB” which invokes the script `movehistorydata.sh` is scheduled to run every day at 23:45:

```
SCHEDULE MDMWS#CLEANUP_DWB_DB
DESCRIPTION "Added by composer."
ON RUNCYCLE RULE1 "FREQ=DAILY;INTERVAL=1"
AT 2345
:
MDMWS#CLEANUP_DWB_DB
 SCRIPTNAME "<MDM_INST_DIR>/TDWB/bin/movehistorydata.sh -dbUsr db2user -dbPwd
xxxxxxx -successfulJobsMaxAge 240 -notSuccessfulJobsMaxAge 720"
 STREAMLOGON root
 DESCRIPTION "This job is used to invoke the script which performs the cleanup of
old dynamic jobs in the database"
 TASKTYPE UNIX
 RECOVERY STOP
```

If this suggested implementation is adopted, it is needed to properly configure the file `<TWA_DATA_DIR>/broker/config/JobDispatcherConfig.properties.`
In particular, the values specified for `SuccessfulJobsMaxAge` and `MoveHistoryDataFrequencyInMins` in the `JobDispatcherConfig.properties` file need to be higher than the values specified in the MDMWS#CLEANUP_DWB_DB job (see example above):

- `SuccessfulJobsMaxAge = 360 (15 days)`
- `MoveHistoryDataFrequencyInMins = 720 (12 hours)`

5 Recommendations

5.1 CPU capacity

All tests described in this document were executed on virtual CPUs assigned exclusively to VMs (reserved resources). The Workload Scheduler product can run successfully with different configurations. However, in the event additional resources are available, or deploying into a virtual environment where over commitment is possible and resources will be dynamically utilized, the recommended values will permit greater concurrency and reduce processor latency. While planning the correct CPU sizing, the information provided in Table 11 could be a reference point to start. The validity of the superposition property that allows us to assume that the resource usage could be considered as the sum of the UI (DWC) usage plus the core scheduling usage was demonstrated.

5.2 Storage

It is not in the scope of this document to suggest a specific storage solution, but the relevance of I/O capacity was underlined in previous performance reports in relation to the overall product performance.

The numbers presented in Figure 26 could be used as a reference while planning a solution because they are the output of I/O Industry standard benchmark, and they can be considered key performance indicators to be used for comparison. In the reference below, we have used IOzone version 3.465 tool with options “-R -l 5 -u 5 -r 4k -s 100m –F file1 ...file5”:

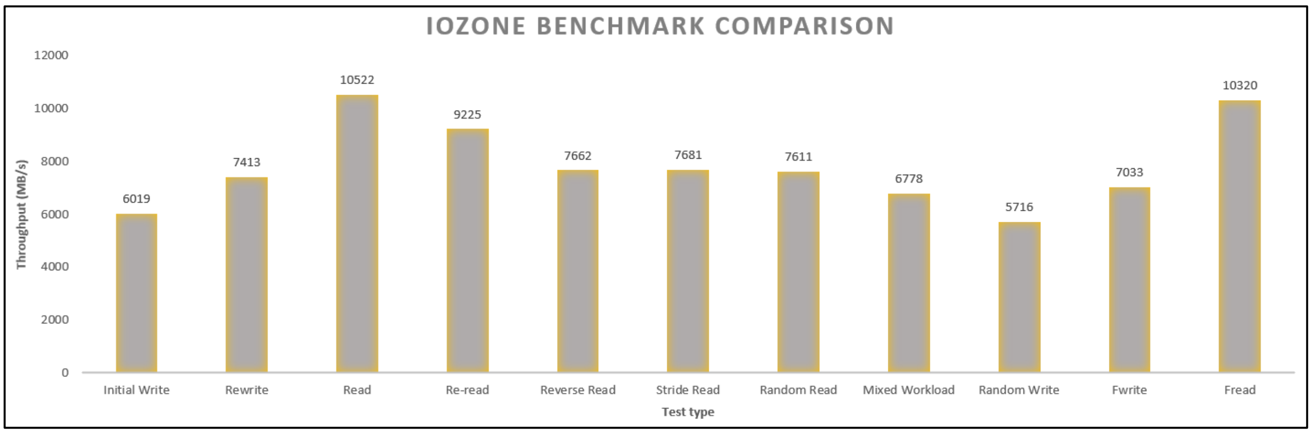


Figure 26. IOzone benchmark results

Moreover, we have monitored VMWare ESXi server disk latency over time while Workload Scheduler daily production plan was running. On average, the disk latency was negligible, with some sporadic peaks lower than 5 ms. Disk latency needs to be constantly monitored because if it increases over the desired threshold of 5 ms, the impacts on product performance would be tangible, causing delays on real time scheduling.

5.3 Memory

RAM size is strongly impacted by the JVM heap size settings whose suggested configuration could be found in the following tables:

DWC users per DWC instance	1 – 10	10 – 50	50 – 150
DWC heap size	2 GB	4 GB	6 GB

Table 5. Dynamic Workload Console IBM WebSphere Application Server Liberty Base heap configuration

Schedule (jobs/min)	1 – 50	50 – 100	100 – 200	>200
MDM Heap size	2 GB	2.5 GB	4 GB	6 GB

Table 6. MDM IBM WebSphere Application Server Liberty Base heap configuration

In addition to the above memory requirements, the native memory for the Java™ process and Workload Scheduler process should be taken into consideration. This means that the RAM allocated for the machines where Dynamic Workload Console and/or Master Domain Manager applications are running needs to be higher than the memory assigned to the IBM WebSphere Application Server Liberty Java Virtual Machines: between 50% and 100% more than the memory assigned to the IBM WebSphere Application Server Liberty Java Virtual Machines.

5.4 Tunings and settings

The following parameters were tuned during performance tests. These appliances are based on common performance best practices, also used in previous releases, and tuning activities during the test execution.

5.4.1 Data Source

Starting from version 9.5, Workload Scheduler has moved from IBM WebSphere Application Server to IBM WebSphere Application Server Liberty Base. As a result, the utilities known as `wastools` have been replaced with some xml templates addressing widely used configurations. You can now configure IBM WebSphere Application Server Liberty Base to work with Workload Scheduler using the templates provided or define your custom .xml files containing your own configuration settings.

IBM WebSphere Application Server Liberty Base retrieves the .xml files from the `overrides` folder:

- MDM/BKM/DDM: `<TWA_DATA_DIR>/usr/servers/engineServer/configDropins/overrides`
- DWC: `<DWC_DATA_DIR>/usr/servers/dwcServer/configDropins/overrides`

and applies the configuration settings defined in each file. The file name is irrelevant, because IBM WebSphere Application Server Liberty Base analyzes each .xml file for its contents.

By default, the datasource settings are specified into the `datasource.xml` file located in the `overrides` folder, and these is the list of suggested values for both MDM and DWC nodes:

- statementCacheSize="400"
- isolationLevel="TRANSACTION_READ_COMMITTED"
- connectionTimeout="180s"
- maxPoolSize="300"
- minPoolSize="0"
- reapTime="180s"
- purgePolicy="EntirePool"

These are the values used to run all the workload scenarios described in this document.

5.4.2 Plan replication in the database (mirroring)

The plan replication feature, also known as mirroring, has been improved release after release through parallelism (multithreading) and caching. The former has defaulted to 6 threads process with 6 related mirrorbox_.msg queues. In case of high rates (thousands of jobs status updates per minute) or other environment configurations (network latency between Master Domain Manager and database), it could be advisable to enlarge the number of mirroring threads and queues:

- `com.ibm.tws.planner.monitor.subProcessors = 10`

Furthermore, the usage of a cache improves performances in the way the plan update processing avoids querying database for information already handled:

- `com.ibm.tws.planner.monitor.cachesize = 70000`
- `com.ibm.tws.planner.monitor.cachemaxage = 21600000`

Since Workload Scheduler version 9.4.0.1, a new caching mechanism was added to accomplish the stress of scenarios with thousands of file dependency status updates:

- `com.ibm.tws.planner.monitor.filecachesize = 40000`

These settings can be specified in the file on the Master Domain Manager:
`<TWA_DATA_DIR>/usr/servers/engineServer/resources/properties/TWSConfig.properties`

5.4.3 Oracle database configuration

The Oracle database configuration that has been used in this context was the default applied by 19c Enterprise Edition installation. It is advisable to enable the Datafile AUTOEXTEND property (ON), considering that the settings and workload described in this section caused a table space occupancy of about 50 GB or greater.

5.4.4 Comprehensive configuration and tuning

Dynamic Workload Console		
Component	**Settings**	**Comment**
WebSphere Liberty JVM options	-Xms<heap size> -Xmx<heap size> -Xmn<nursery size> -Xgcpolicy:gencon -Xdisableexplicitgc	Heap size: • 4096m for up to 50 users/instance • 6144m for up to 150 users/instance Nursery size: ¼ of heap size Configuration file: jvm.options in <DWC_DATA_DIR>/usr/servers/dwcServer/configDropins/overrides/
WebSphere Liberty Datasource	JDBC max Connections = 300	Configuration file: datasource.xml in <DWC_DATA_DIR>/usr/servers/dwcServer/configDropins/overrides/

Table 7. DWC recommended settings.

Master Domain Manager		
Component	**Settings**	**Comment**
batchman	bm check deadline = 0 bm check file = 120 bm check status = 300 bm check untils = 300 bm late every = 0 bm look = 5 bm read = 3 bm stats = off bm verbose = off	Configuration file: <TWA_DATA_DIR>/localopts
WebSphere Liberty JVM options	-Xms<heap size> -Xmx<heap size> -Xmn<nursery size> -Xgcpolicy:gencon -Xdisableexplicitgc	Heap size: • 4096m for up to 200 jobs/min • 6144m for more than 200 jobs/min Nursery size: ¼ of heap size Configuration file: jvm.options in <TWA_DATA_DIR>/usr/servers/engineServer/configDropins/overrides/
WebSphere Liberty Datasource	JDBC Type = 4 Connection Timeout = 180 Max Connections = 300 Min Connections = 0 Purge Policy = EntirePool Reap Time = 180 Statement Cache Size= 400	Configuration file: datasource.xml in <TWA_DATA_DIR>/usr/servers/engineServer/configDropins/overrides/

Table 8. MDM recommended settings.

DB2			
Parameter		**Value**	**Comment**
LOGPRIMARY		200	**LOGPRIMARY**: total number of transaction logs **LOGFILSIZ**: number of 4KB pages for each transaction log **Total transaction logs size**: 200 * 6000 * 4KB ≈ 4.6 GiB
LOGFILSIZ		6000	
KEEPFENCED		NO	
MAX_CONNECTIONS		AUTOMATIC	
MAX_COORDAGENTS		AUTOMATIC	
STMT_CONC		LITERALS	This setting optimizes query execution and reduces CPU usage
SELF_TUNING_MEM		ON	
APPL_MEMORY **APPLHEAPSZ** **DATABASE_MEMORY** **DBHEAP** **STAT_HEAP_SZ**		AUTOMATIC	
AUTO_RUNSTATS		ON	
AUTO_STMT_STATS		ON	
AUTO_REORG		OFF	
PAGE_AGE_TRGT_MCR		120	
TWS_PLN_BUFFPOOL	NPAGES	-2	Automatic (self-tuning)
	PAGESIZE	16384	
TWS_BUFFPOOL_TEMP	NPAGES	500	
	PAGESIZE	16384	
TWS_BUFFPOOL	NPAGES	-2	Automatic (self-tuning)
	PAGESIZE	16384	

Table 9. DB2 recommended settings.

Dynamic Workload Broker		
Feature	**Settings**	**Comment**

Historical data management	MoveHistoryDataFrequencyInMins = 720
	SuccessfulJobsMaxAge = 360
		Configuration file: JobDispatcherConfig.properties in <TWA_DATA_DIR>/broker/config/

Dynamic scheduling	MaxAllocsPerTimeSlot = 1000
	TimeSlotLength = 10
	MaxAllocsInCache = 50000
		Configuration file: ResourceAdvisorConfig.properties in <TWA_DATA_DIR>/broker/config/

Plan replication configuration	com.ibm.tws.planner.monitor.subProcessors = 10
	com.ibm.tws.planner.monitor.filecachesize = 40000
	com.ibm.tws.planner.monitor.cachesize = 70000
	com.ibm.tws.planner.monitor.cachemaxage = 21600000
		Configuration file: TWSConfig.properties in <TWA_DATA_DIR>/usr/servers/engineServer/resources/properties/

Table 10. Dynamic Workload Broker recommended settings.

6 Capacity Plan Examples

In the context of this document, the number of key parameters used to identify the workload was kept to a minimum:

1. Number of DWC users.
2. Number of jobs to be scheduled.
3. Percentage of dynamic jobs to schedule.

With the above input, it is possible to forecast the resources needed to host version 10.1.0.x product. Internal fit functions were used to model the workload and resource usage relationship. A 65% CPU usage was the threshold considered before requesting additional cores.

In this section, some examples of capacity planning are reported. Remember that all the requirements are related to Linux X86 VM in a VMWare virtualization with reserved resources; nevertheless, this information could be used as a reference point for different platform architectures.

	NODE	Number of virtual vCPU cores	RAM Capacity (GB)
up to 10K jobs (50% FTA +50% DYN) per day (~8 jobs/min), 10 DWC users			
1 Node	**MDM, RDBMS, DWC**	4	16
up to 100K jobs (50% FTA +50% DYN) per day (~70 jobs/min), 50 DWC users			
2 Nodes	**MDM, DWC**	4	16
	RDBMS	4	16
up to 600K jobs (50% FTA +50% DYN) per day (~410 jobs/min), 100+ DWC users			
3 Nodes	**MDM**	8	32
	RDBMS	8	32
	DWC	5	20

Table 11. Capacity planning examples

7 Notices

7.1 *Trademarks*