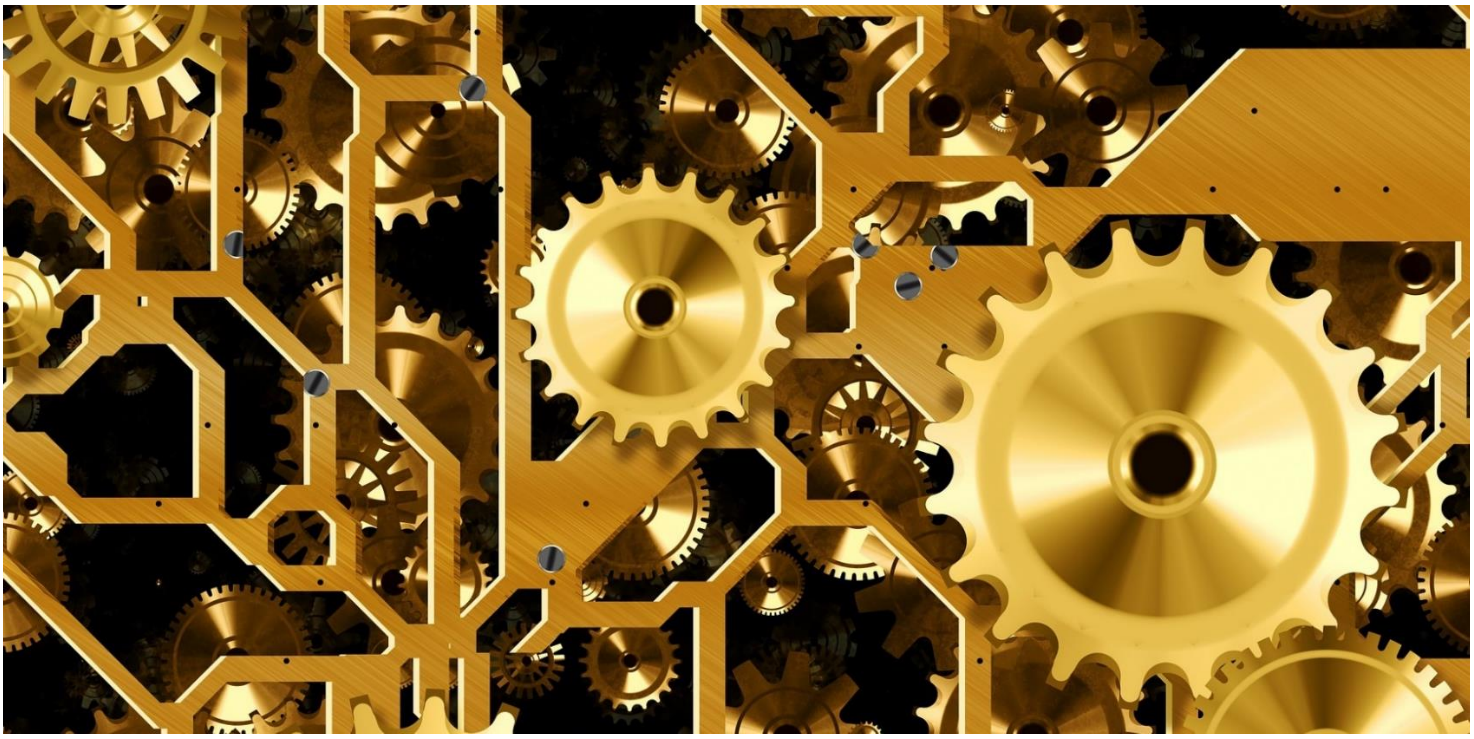


Workload Scheduler Version 9.5.0.2 Performance Report

an IBM + HCL product

Document version 1.0

Pier Fortunato Bottan
Giorgio Corsetti
Alessandro Tomasi
Workload Automation Performance Team - HCL 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 9.5

In the last year and half, the major release 9.5 and two different fix packs have been released.

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

- 9.5 GA
- 9.5 FP1
- 9.5 FP2

This new major release 9.5 and related fix packs 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 main focus areas from Performance Testing perspective have been:

- the replacement of IBM WebSphere® Application Server with IBM WebSphere Application Server Liberty Base for Master Domain Manager, Backup Domain Manager and Dynamic Workload Console
- the introduction of the *<DATA_DIR>* directory, where are located the product data and data generated by IBM Workload Scheduler, such as logs and configuration information
- Dynamic Workload Console evolution based on new architectural foundation of modern front-end technologies while maintaining previous workload logic and processes. With this refurbishment, Dashboard Application Services Hub (DASH) is replaced by a lean, high-performance in-house solution that is now based on a lightweight, highly composable, fast to start, dynamic application server runtime environment, WebSphere Application Server Liberty
- the new capabilities to organize scheduling objects in a hierarchy of folders. This new capability has been initially added for job and job streams only with 9.5 GA but the full support for all scheduling objects has been extended with 9.5 FP2
- deployment of IBM Workload Scheduler on IBM® Cloud Private, an integrated environment for managing containers that includes the container orchestrator Kubernetes, a private image repository, a management console, and monitoring frameworks

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, V9.5.0.2, executed in a test environment based on the VMWare ESX® - Linux x86 platform with the new middleware dependencies in comparison with previous versions (see Workload Scheduler v9.4 Fix Pack 3 performance report) .

The performance results could be summarized as follows:

- Consolidate previous performance achievements in terms of throughput
- Validate tunings for new middleware
- New key performance indicators for product deployment via containers

3 Performance Test

3.1 Test Approach

As specified in section 2.1, the main focus for Workload Scheduler 9.5 performance tests was to validate that the new middleware, supporting the product, allows the same results in terms of throughputs and capacity with respect previous releases; to achieve this the scheduling throughput, resource consumption and reliability are continuously 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 7.7 Kernel 3.10.0-1062.1.1.el7	
RDBMS	IBM DB2 v11.1.4.4 a	Oracle® 12c Enterprise Edition 12.2.0.1.0
J2EE	WebSphere Application Server Liberty Base 20.0.0.3 OpenJDK Runtime Environment (1.8.0_232-b09)	
LDAP	IBM Directory Server 7.2	

WA	9.5.0.2

Table 1. Software level of code

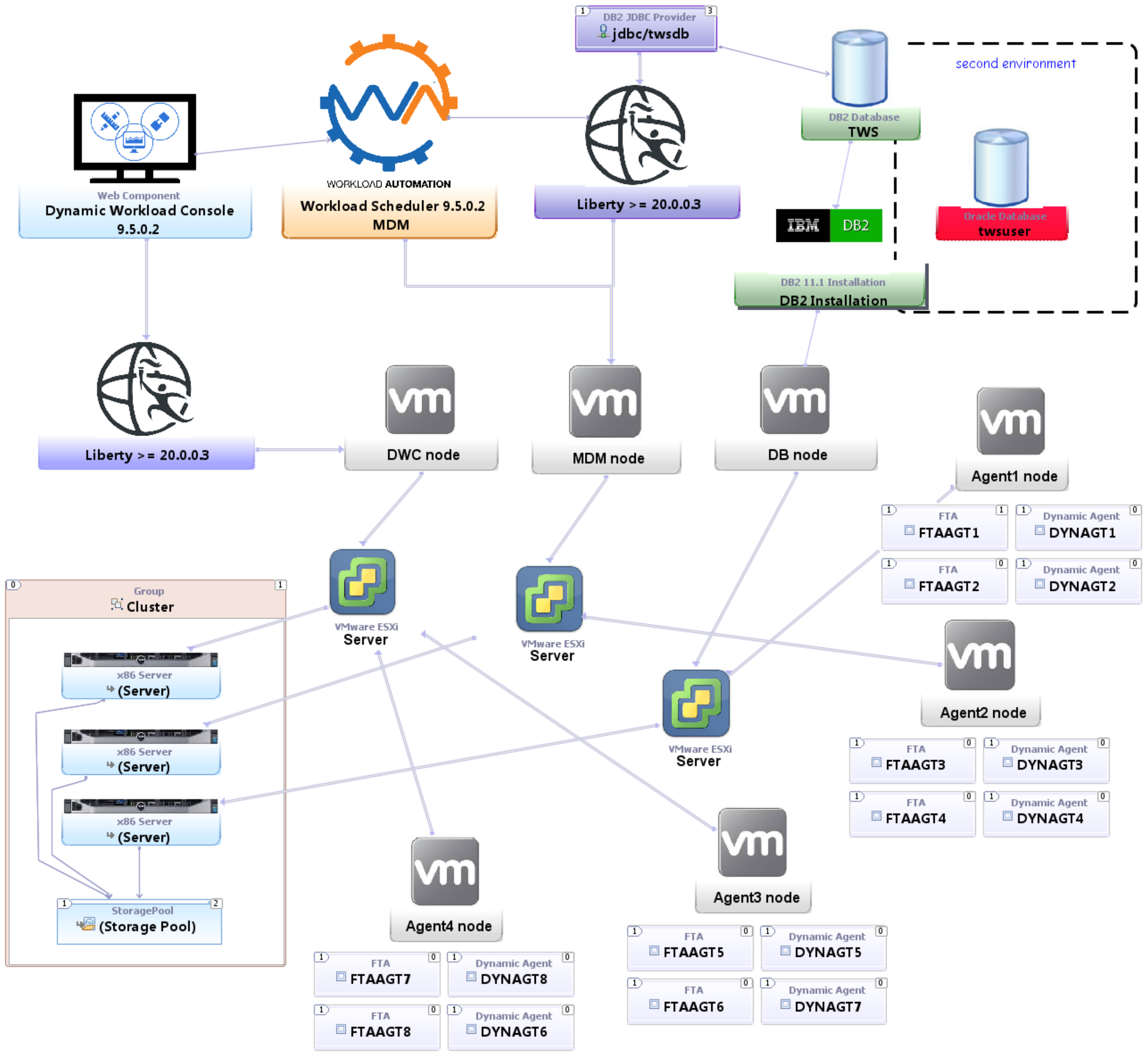
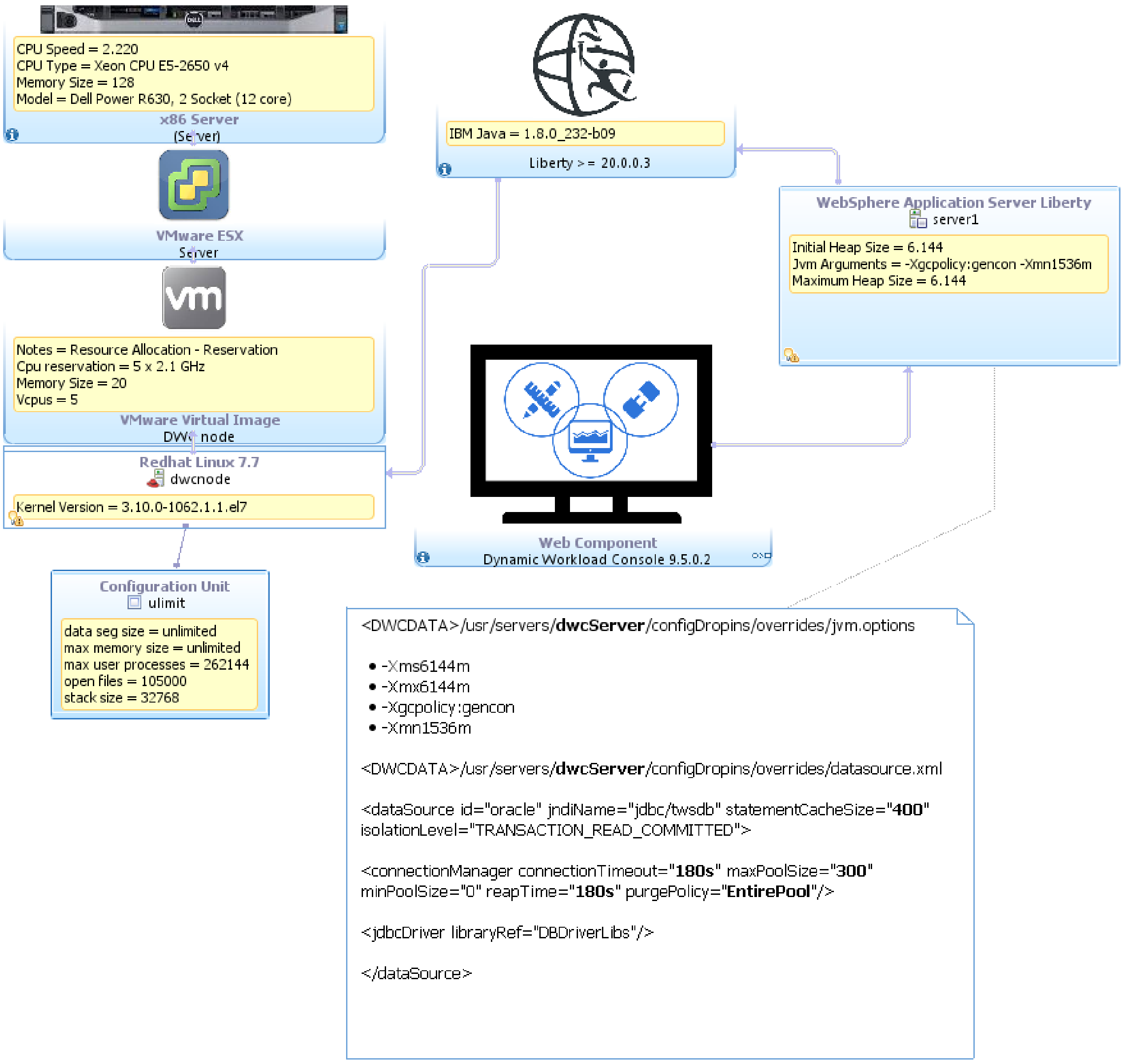


Figure 1. Overall deploy view of test environment



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

Notes = Resource Allocation - Reservation
Cpu reservation = 5 x 2.1 GHz
Memory Size = 20
Vcpus = 5

Kernel Version = 3.10.0-1062.1.1.el7

data seg size = unlimited
max memory size = unlimited
max user processes = 262144
open files = 105000
stack size = 32768

IBM Java = 1.8.0_232-b09

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

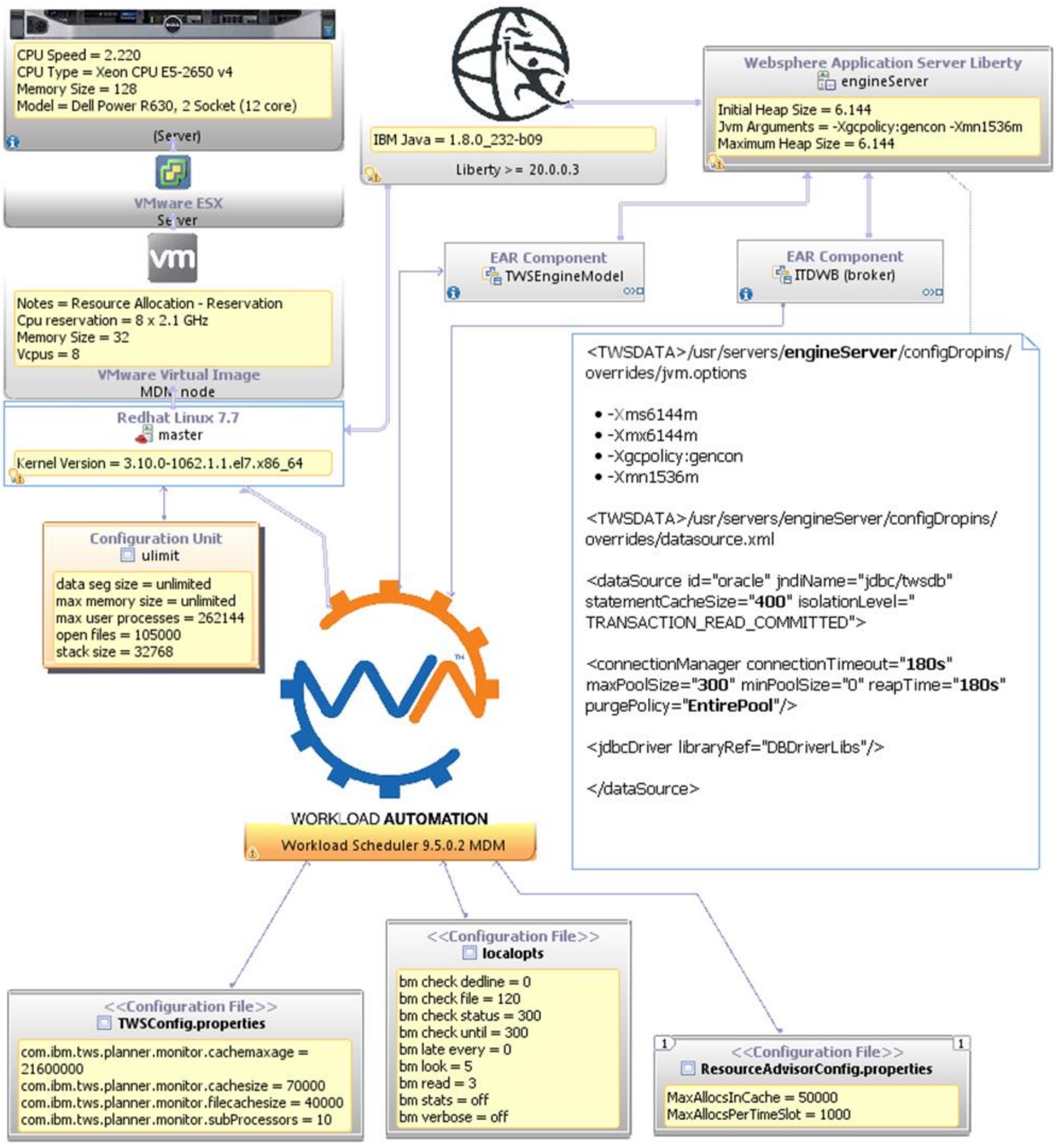
<DWCDATA>/usr/servers/**dwcServer**/configDropins/overrides/jvm.options

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

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

```
<dataSource id="oracle" jndiName="jdbc/twsdb" statementCacheSize="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
```

```
Kernel Version = 3.10.0-1062.1.1.el7.x86_64
```

```
data seg size = unlimited
max memory size = unlimited
max user processes = 262144
open files = 105000
stack size = 32768
```

```
IBM Java = 1.8.0_232-b09
```

```
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

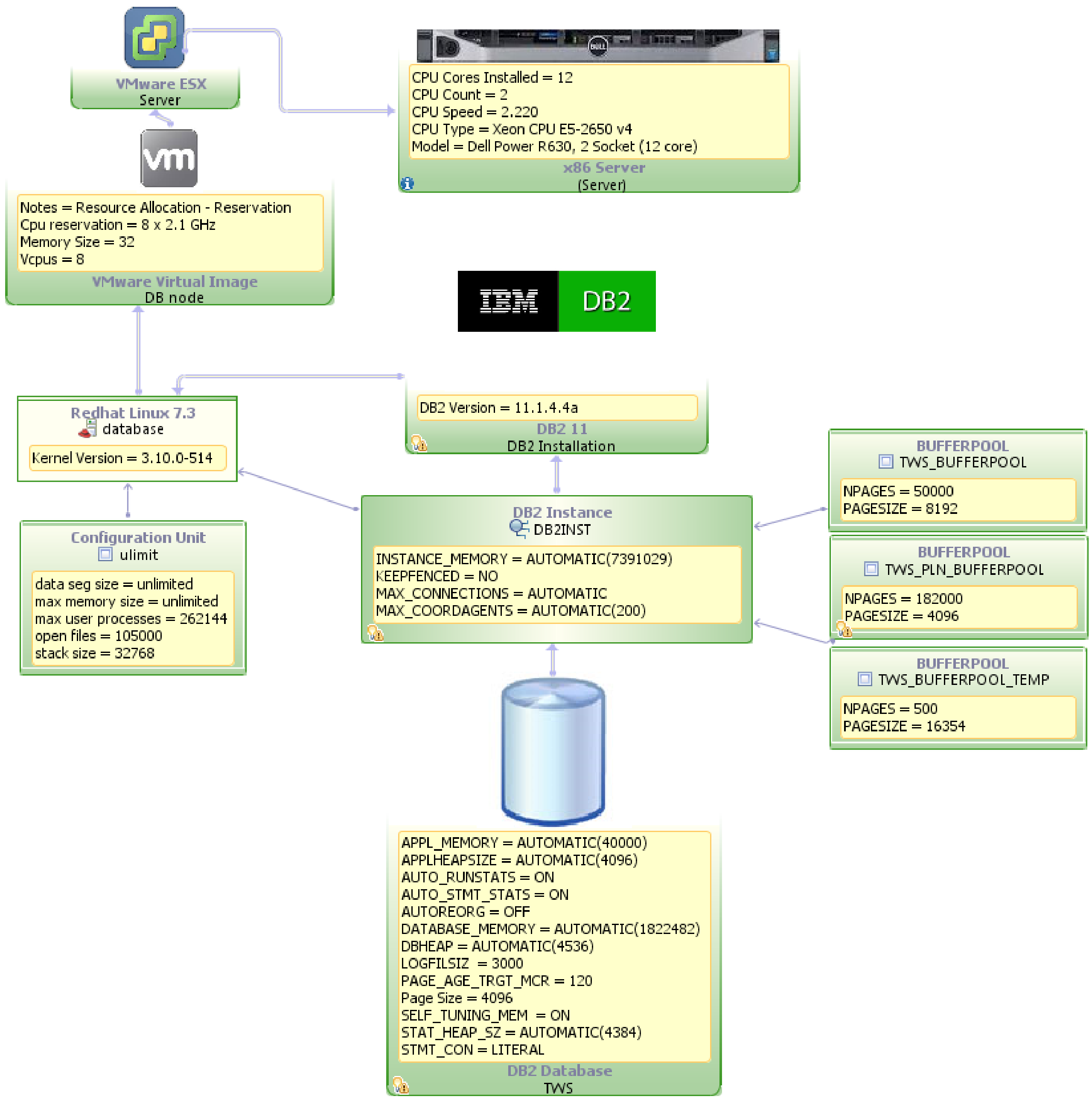
```
<dataSource id="oracle" jndiName="jdbc/twsdb"
statementCacheSize="400" isolationLevel="
TRANSACTION_READ_COMMITTED">

<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

Kernel Version = 3.10.0-514

DB2 Version = 11.1.4.4a

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 = 50000, PAGESIZE = 8192

TWS_PLN_BUFFERPOOL: NPAGES = 182000, PAGESIZE = 4096

TWS_BUFFERPOOL_TEMP: NPAGES = 500, PAGESIZE = 16354

APPL_MEMORY = AUTOMATIC(40000)
APPLHEAPSIZE = AUTOMATIC(4096)
AUTO_RUNSTATS = ON
AUTO_STMT_STATS = ON
AUTOREORG = OFF
DATABASE_MEMORY = AUTOMATIC(1822482)
DBHEAP = AUTOMATIC(4536)
LOGFILSIZ = 3000
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

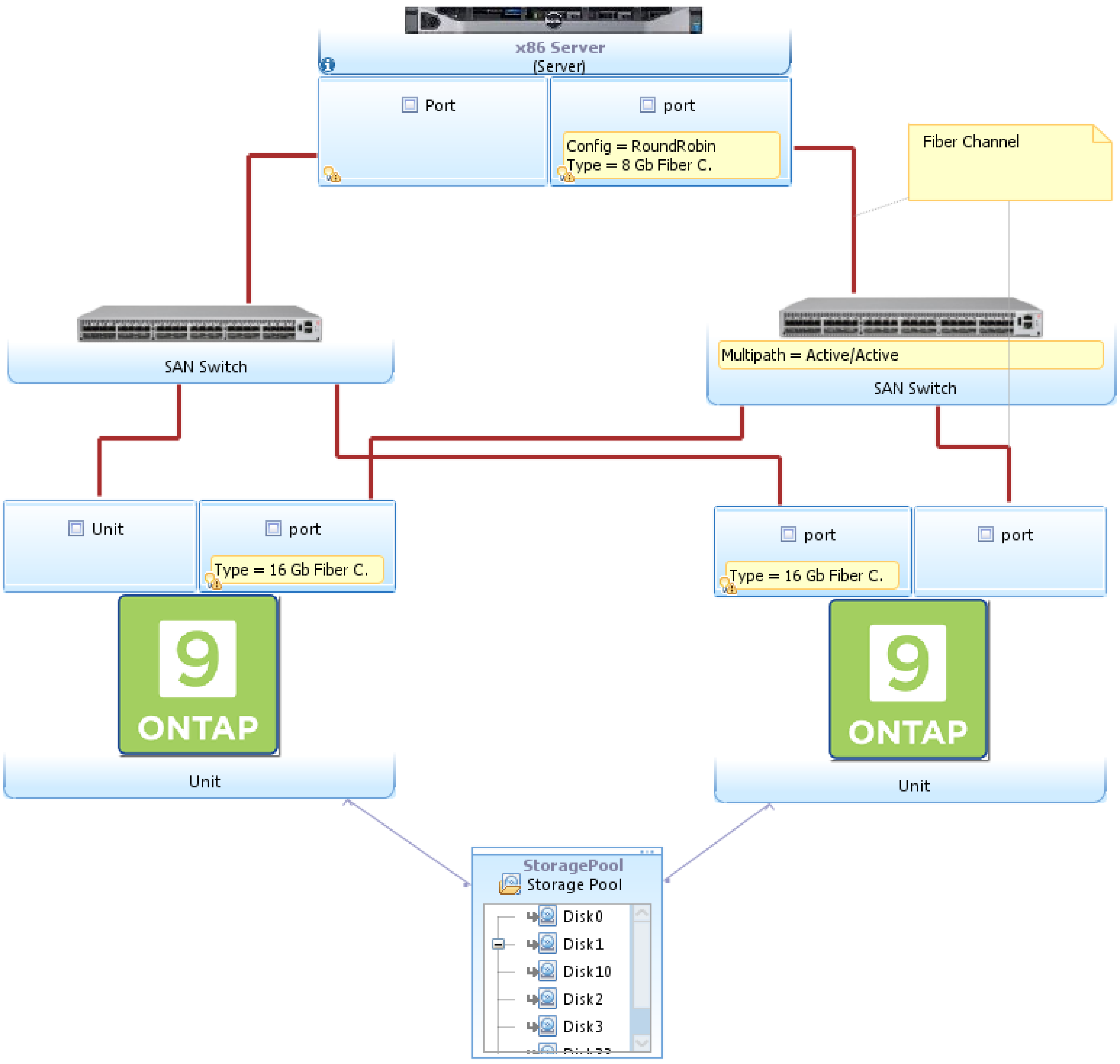


Figure 5. Storage Solution

3.3 Test tools

The main tools used in this performance test context are listed in the following table.

Tool	Scope	Version	Link
nmon	Resource usage	16g	https://www.ibm.com/developerworks/aix/library/au-nmon_analyser/index.html
IOzone	Benchmark I/O capacity	3.434	www.iozone.org
Perfanalyst	DB2 and WAS tuning	1.1.4	https://www.ibm.com/support/pages/ibm-performance-analyst-tool
ROOT framework	Data analysis and Presentation	6.0.8	https://root.cern.ch/
Rational Performance Tester	Concurrent DWC users test	9.5	https://www.ibm.com/us-en/marketplace/ibm-rational-performance-tester
IBM Support Assistant	Java Core, Garbage Collector and thread dump analysis	5.0.2.8	https://www.ibm.com/support/home/product/C100515X13178X21/other_software/ibm_support_assistant
netem	Network delays emulator of wide area network	Kernel 3.10.0-514	https://wiki.linuxfoundation.org/networking/netem

Table 2. Test tools

3.4 Test Benchmarks and Results

3.4.1 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 new feature doesn’t 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, that 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 FTAs . 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.

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.

Table 3. 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.

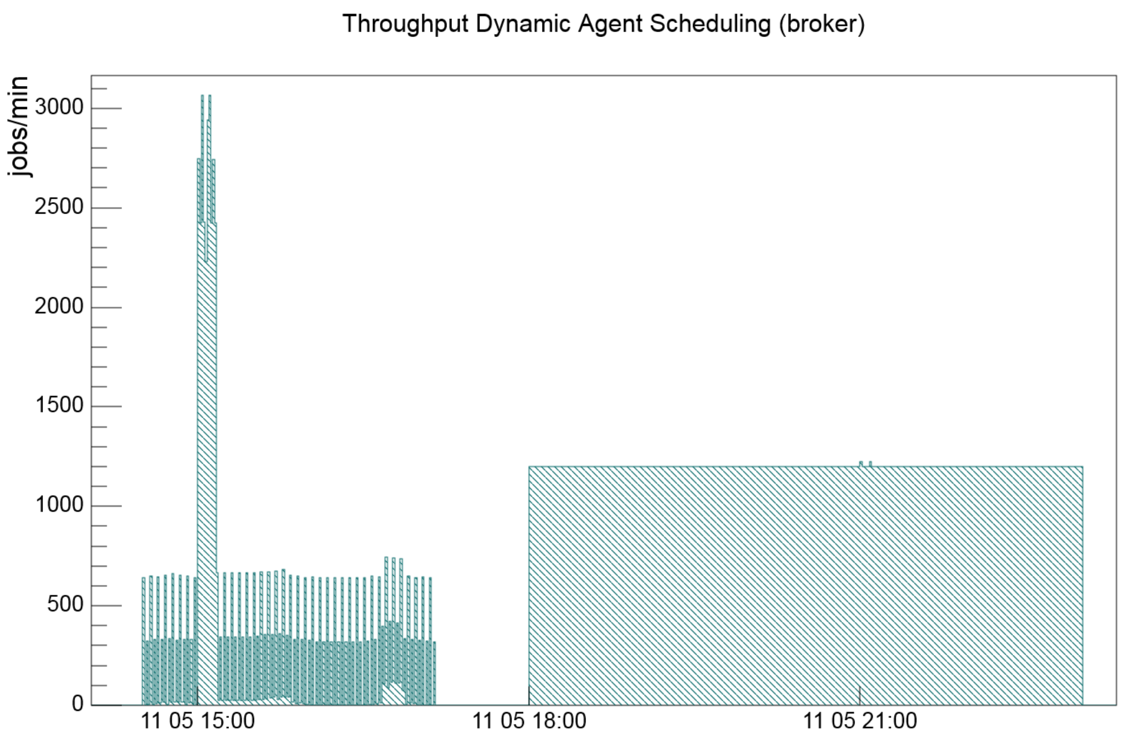


Figure 6. Dynamic agent daily throughput. The total number of jobs scheduled in a day is around 4.3 x10^5

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**.

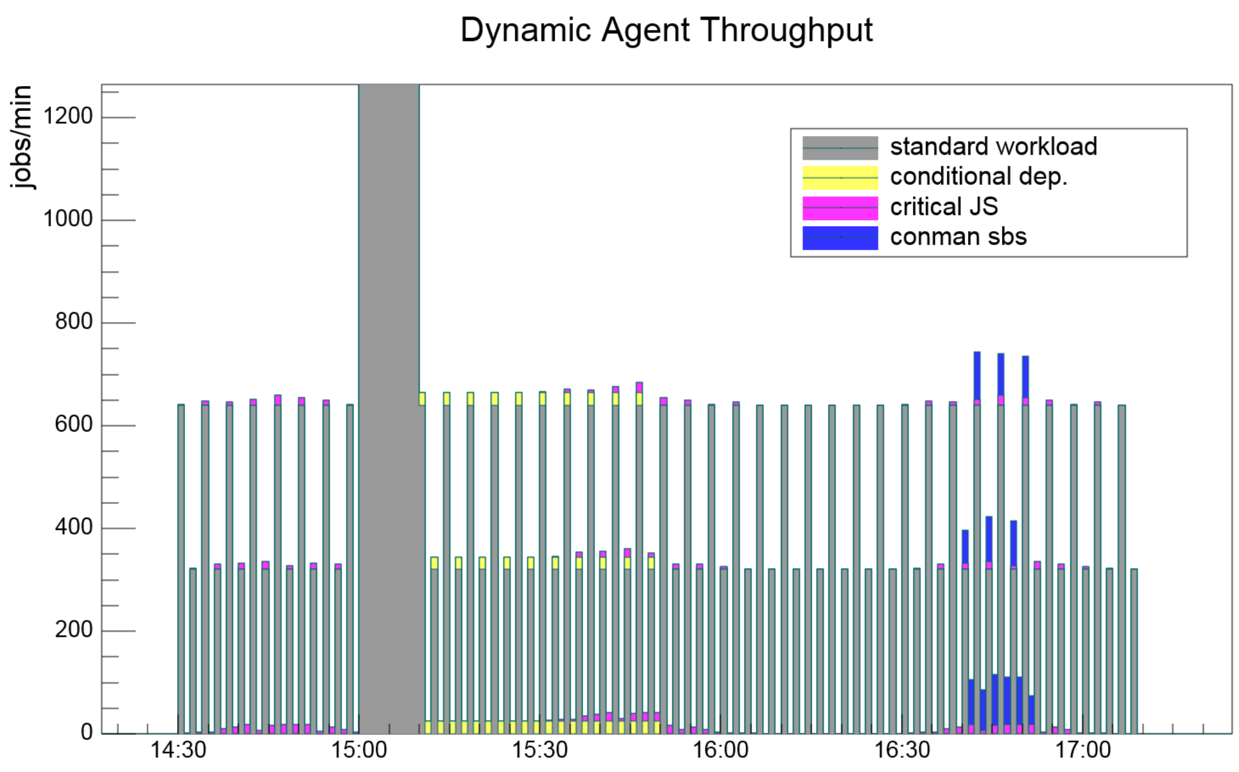


Figure 7. Zoomed view of dynamic agent scheduling

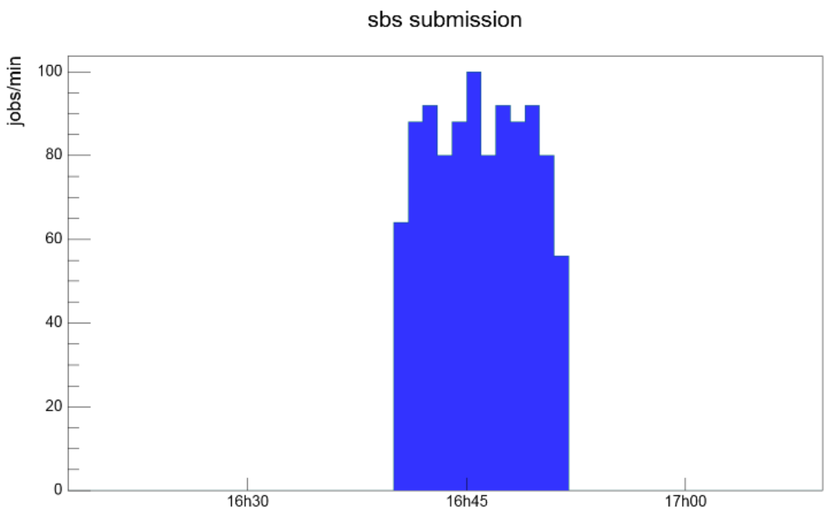


Figure 8. Dynamic agent schedule: ad hoc submission with 1000 scheduled jobs

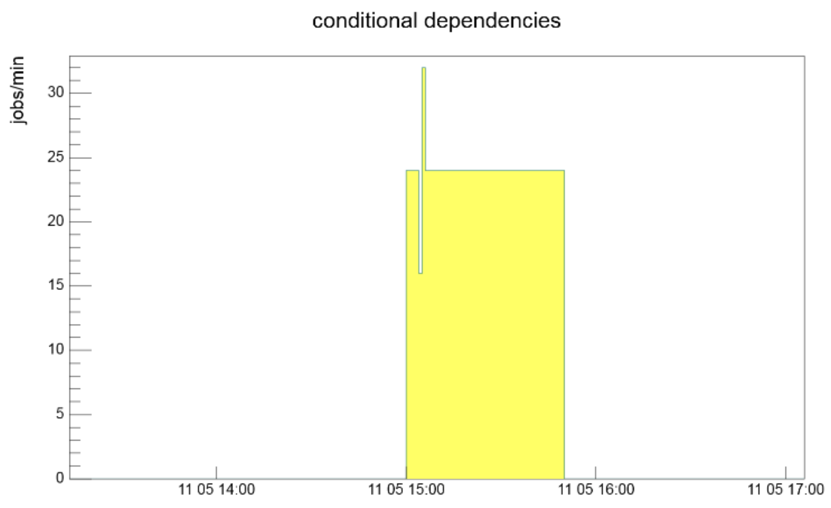


Figure 9. Dynamic agent schedule: jobs with internal and external conditional dependencies (3200 total jobs)

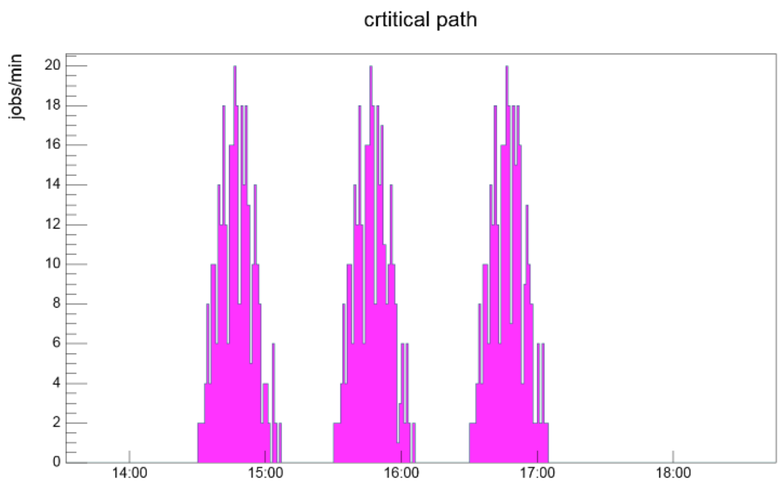


Figure 10. Dynamic agent schedule: jobs belonging to a critical path (Workload Service Assurance)

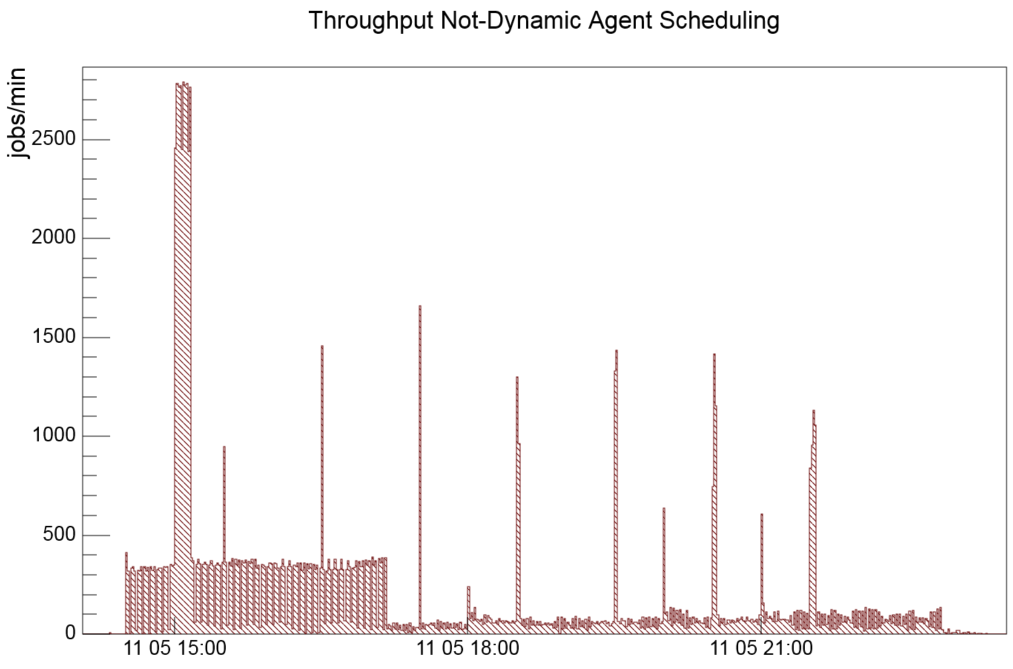


Figure 11. Fault tolerant agent schedule throughput in daily workload (10^5 jobs)

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 from the Dynamic Workload Console (see **Figure 13**).

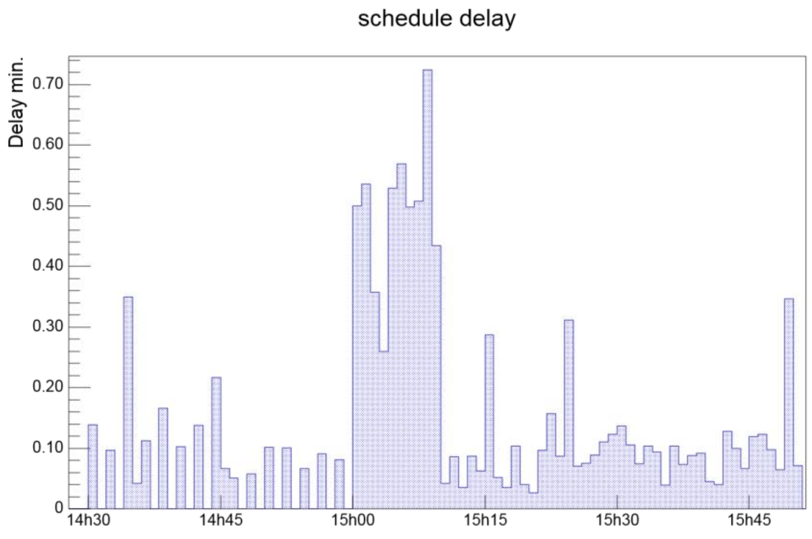


Figure 12. Average job schedule delay over time

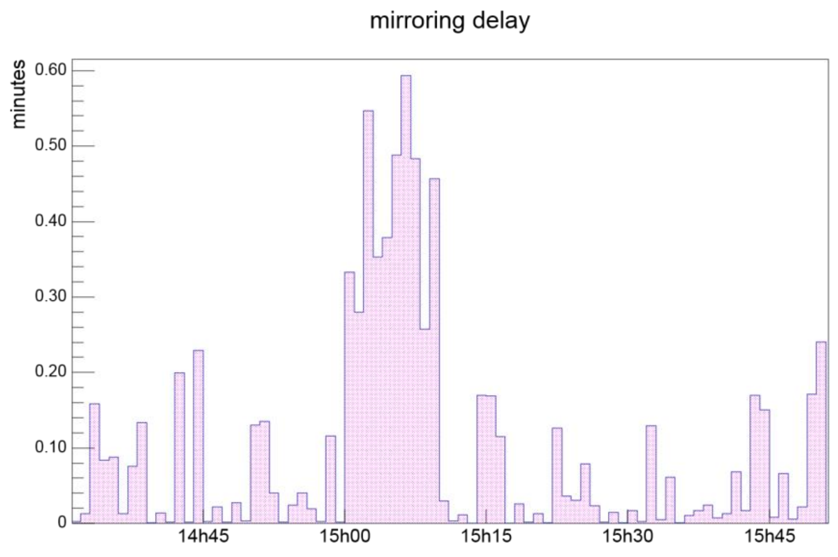


Figure 13. Job plan status update delay over time

The above results evidence the promptness of Workload Scheduler in the 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 is interesting to note how the average scheduling delay is around 4 seconds as expected from the `batchman` process configuration (`bm look`, `bm read` settings). 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 with respect to the outcoming throughput for dynamic agent and fault tolerant agents scheduling as described in section 3.4.1. It is interesting to underline how the MDM CPU footprint for 9.5.x is lower than 9.4.x. This means that the 9.5.x MDM spends less CPU than 9.4.x MDM to run the same workload over time.

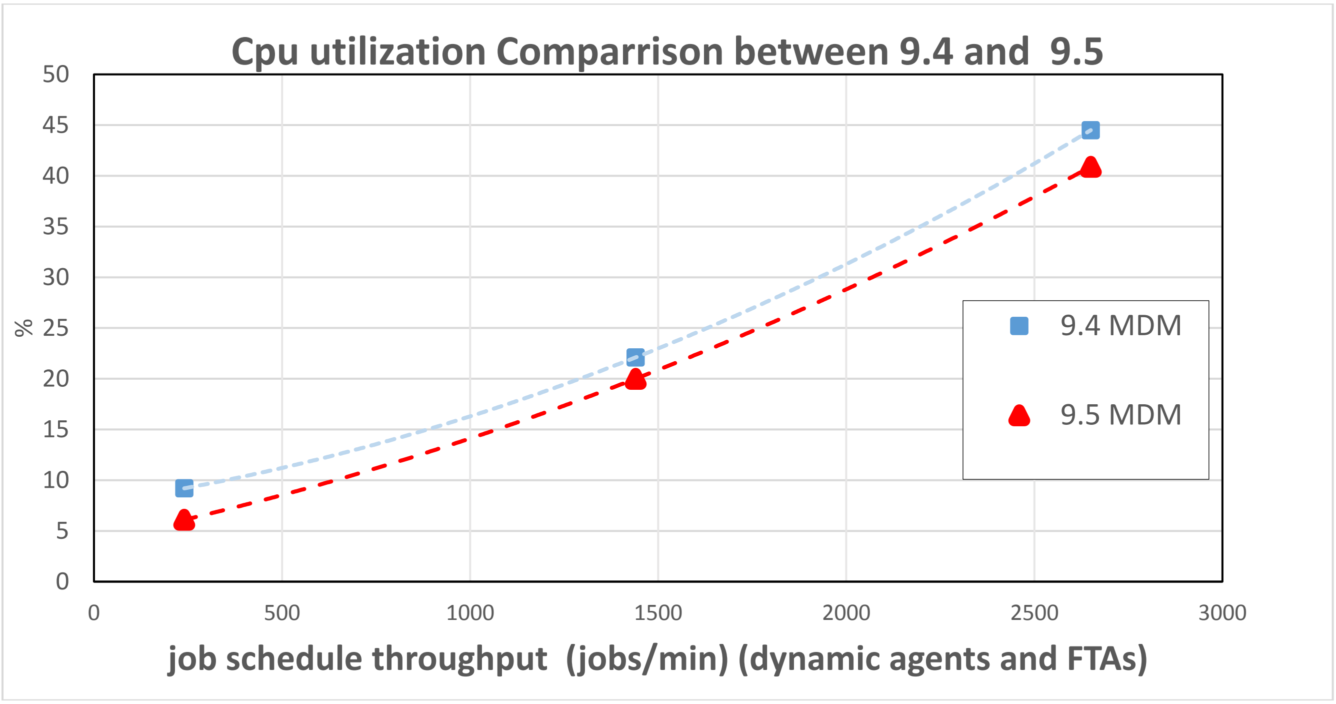


Figure 14. Total average CPU utilization at master domain manager vs different workload

3.4.2 Dynamic Workload Console Workload

The Dynamic Workload Console has been tested to probe the performance and scalability levels assured by the new architecture for which Dashboard Application Services Hub (DASH) has been replaced by a lean, high-performance in-house solution that is now based on a lightweight, highly composable, fast to start, dynamic application server runtime environment, WebSphere Application Server Liberty.

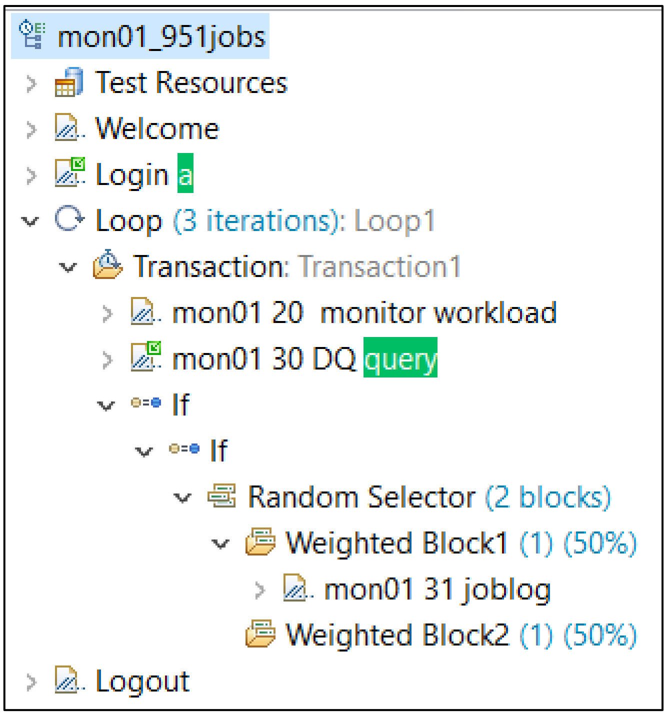
In particular, the concurrent user tests have been executed using a topology with 2 DWC in clusters (see **Figure *2*** for details on single DWC node configuration) configured to use the same DWC database created on Oracle 12c Enterprise Edition 12.2.0.1.0 Server.

The workload has been designed to replicate scenarios and transaction rates applied in previous tests. Table describes the scenarios and its weight in the overall payload including user percentage and single user transaction rate.

Scenario	Brief description	User %	Rate per user (transaction/min)
Mon01_951jobs	Query on job status (Figure 15)	40	0.5
Mon03_pv	Open plan view and drill down in job stream details (Figure 16)	10	0.5
Erule01_monitorER	Select an event rule, view details and monitor the triggering (Figure 17)	10	0.5
Mod01_CreateJ&JS	Create 2 jobs and add then to a new JS (Figure 18)	20	0.5
Mod02_EditJS	Modify JS (Figure 19)	20	0.5

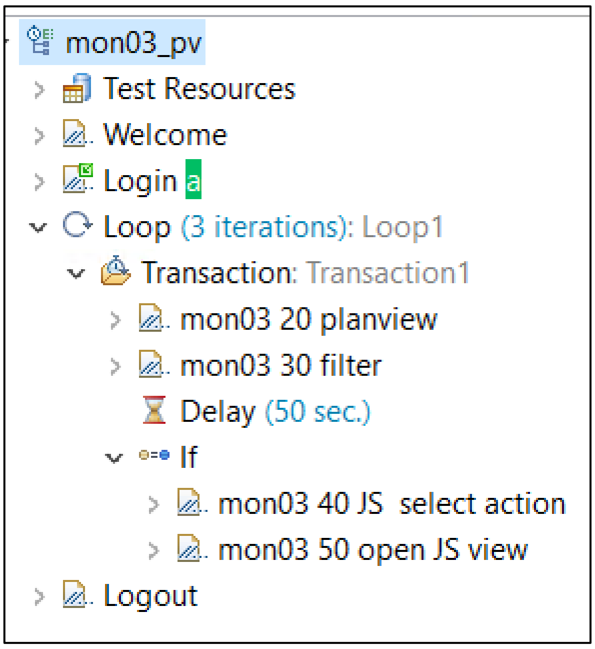
Table 4. Dynamic Workload Console scenarios with distribution (%) and rates per user

Figure 15-19 represent the details of each Dynamic Workload Console scenario, from login, transaction loop, driven by specific rate, and logout. Multiple data pool sets have been used to achieve variation in the payload, for example in the Mon_01_jobs scenario the plan queries vary filtering and producing different result set from few to thousands of jobs.



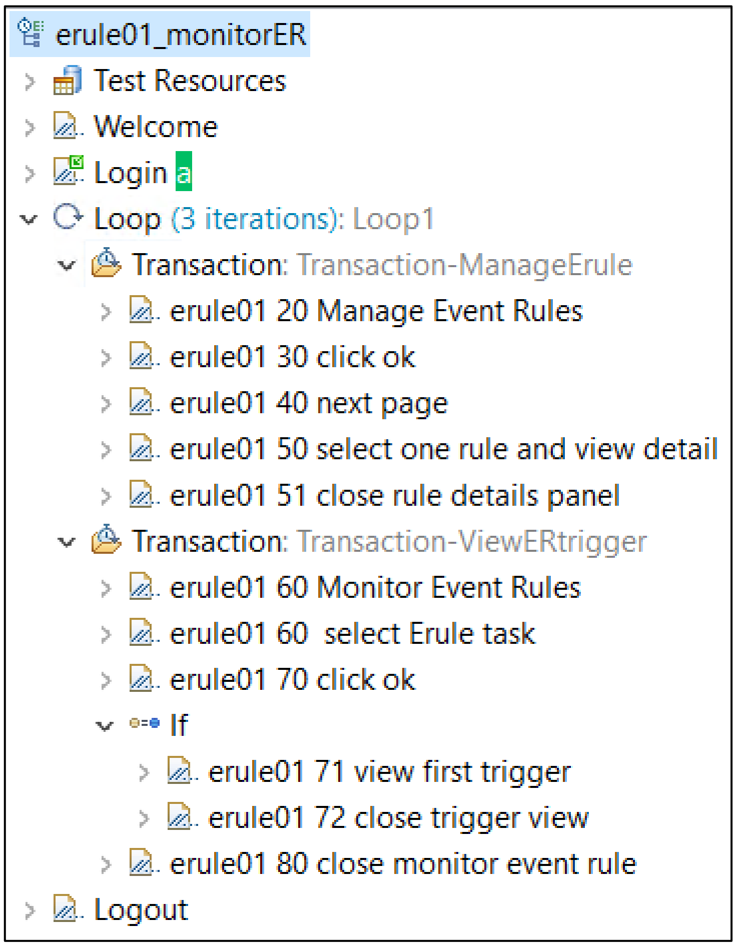
```
mon01_951jobs
  > Test Resources
  > Welcome
  > Login a
  v Loop (3 iterations): Loop1
      v Transaction: Transaction1
          > mon01 20  monitor workload
          > mon01 30 DQ query
          v If
              v If
                  v Random Selector (2 blocks)
                      v Weighted Block1 (1) (50%)
                          > mon01 31 joblog
                        Weighted Block2 (1) (50%)
  > Logout
```

Figure 15. Monitor Job Scenario with retrieve of job log



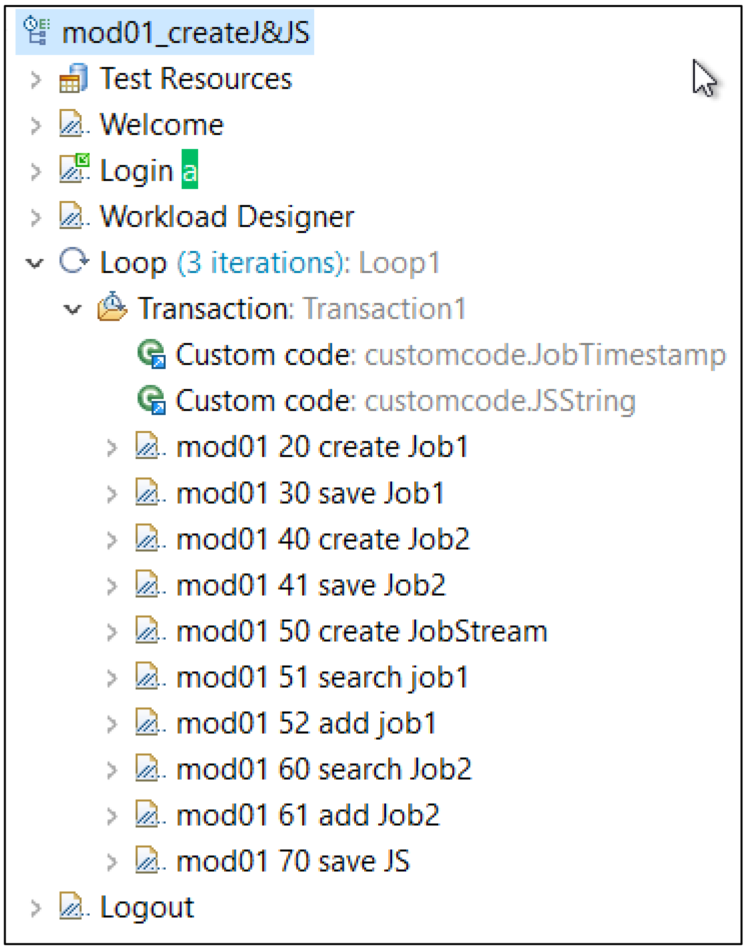
```
mon03_pv
  > Test Resources
  > Welcome
  > Login a
  v Loop (3 iterations): Loop1
      v Transaction: Transaction1
          > mon03 20 planview
          > mon03 30 filter
            Delay (50 sec.)
          v If
              > mon03 40 JS  select action
              > mon03 50 open JS view
  > Logout
```

Figure 16. Plan View and job stream view



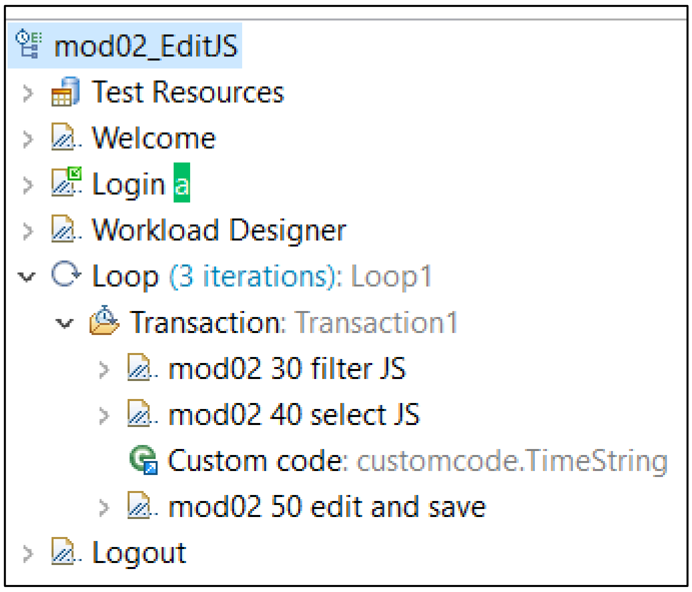
```
erule01_monitorER
> Test Resources
> Welcome
> Login a
v Loop (3 iterations): Loop1
    v Transaction: Transaction-ManageErule
        > erule01 20 Manage Event Rules
        > erule01 30 click ok
        > erule01 40 next page
        > erule01 50 select one rule and view detail
        > erule01 51 close rule details panel
    v Transaction: Transaction-ViewERtrigger
        > erule01 60 Monitor Event Rules
        > erule01 60  select Erule task
        > erule01 70 click ok
        v If
            > erule01 71 view first trigger
            > erule01 72 close trigger view
        > erule01 80 close monitor event rule
> Logout
```

Figure 17. Monitor event rule and triggers



```
mod01_createJ&JS
> Test Resources
> Welcome
> Login a
> Workload Designer
v Loop (3 iterations): Loop1
  v Transaction: Transaction1
      Custom code: customcode.JobTimestamp
      Custom code: customcode.JSString
    > mod01 20 create Job1
    > mod01 30 save Job1
    > mod01 40 create Job2
    > mod01 41 save Job2
    > mod01 50 create JobStream
    > mod01 51 search job1
    > mod01 52 add job1
    > mod01 60 search Job2
    > mod01 61 add Job2
    > mod01 70 save JS
> Logout
```

Figure 18. Create jobs and job streams



```
mod02_EditJS
> Test Resources
> Welcome
> Login a
> Workload Designer
v Loop (3 iterations): Loop1
  v Transaction: Transaction1
    > mod02 30 filter JS
    > mod02 40 select JS
      Custom code: customcode.TimeString
    > mod02 50 edit and save
> Logout
```

Figure 19. Edit existing Job Stream

As said before, the tests have been performed with two Dynamic Workload Consoles pointing to a database for clustering purpose. The workload balancing has been achieved by using the Rational Performance Tester capability to uniformly distribute workload between the two nodes. Tests have been scheduled with increasing payload in terms of concurrent users (**Figure 20**) with the specific rates and scenarios distribution (**Table 4**).

The test with 300 users generates an overall concurrency in the steady state of around:

19 pages/seconds

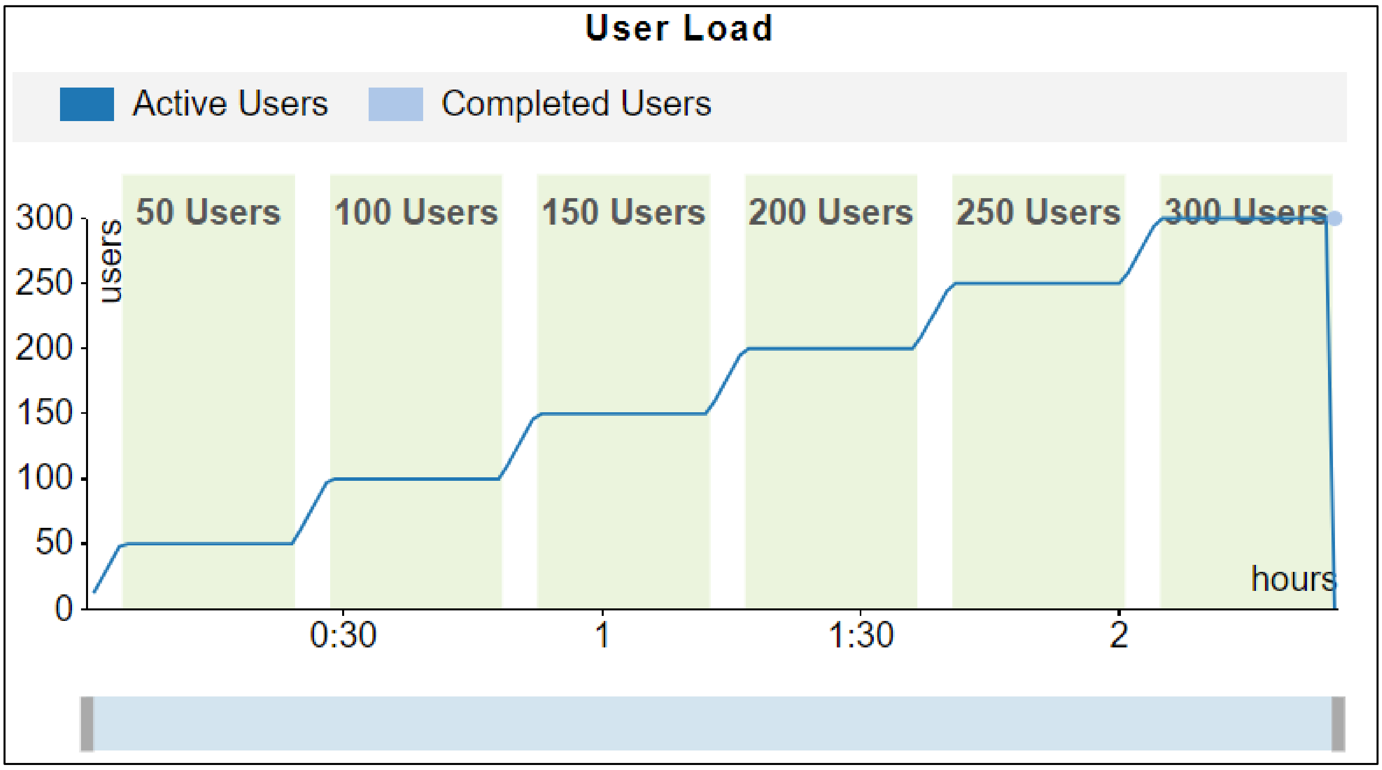


Figure 20. Concurrent user income

Results show a linear relationship for incoming throughput (concurrent users) and outgoing throughput (page requests fulfilled per second) as illustrated in **Figure 21**. Linearity has been also confirmed in resource usage.

The pages response time shown in **Figure *22***, **Figure *23*** and **Figure *24*** below are related to the time spent on the DWC Servers to reply to the Rational Performance Tester calls and it doesn't include the time spent on the browser to process and show data. Moreover, these results have been obtained in a situation for which no significant scheduling workload was running on the back end (Master Domain Manager, Domain Manager and Dynamic Domain Manager). If significant scheduling workload is running, the average pages response time would be worst considering the extra CPU-intensive work running on both the Master Domain Manager and Database machines.

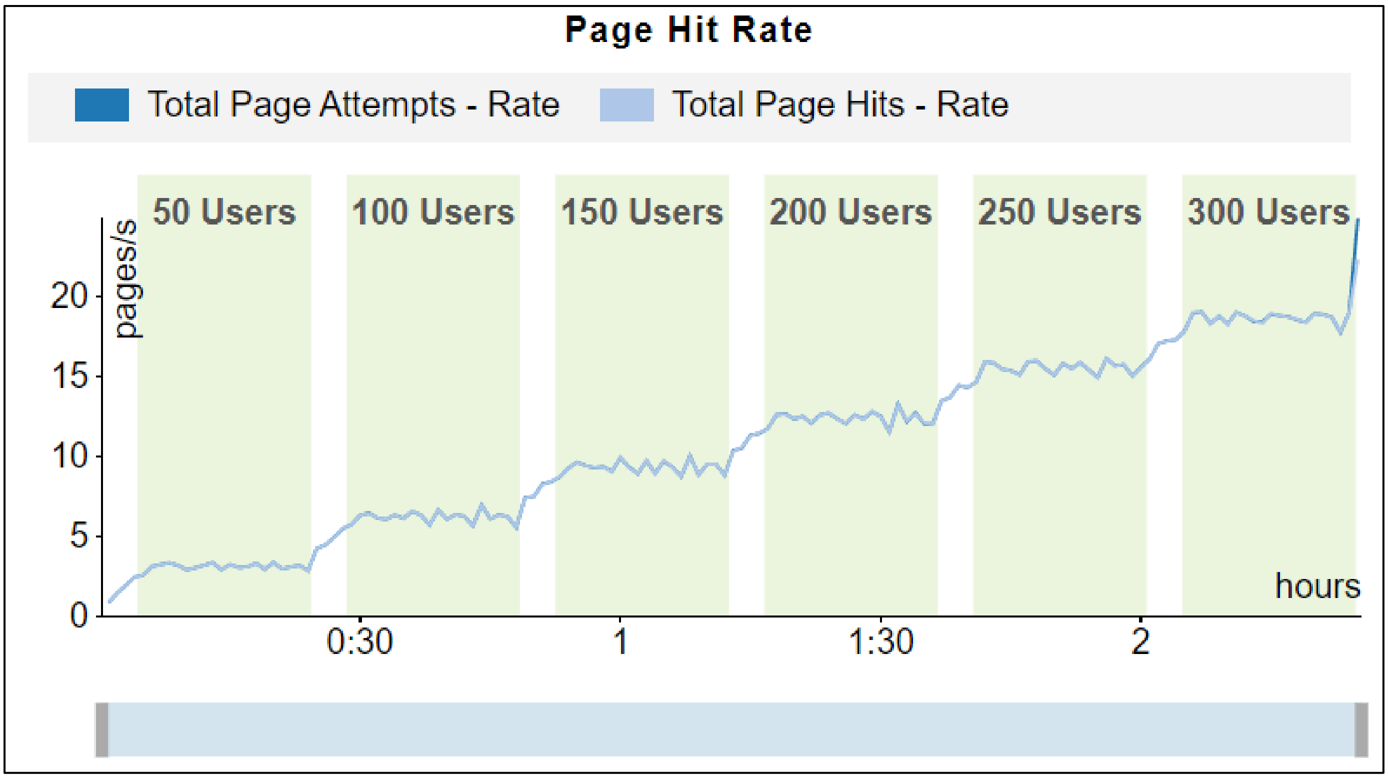


Figure 21. Pages per second attempted during workload

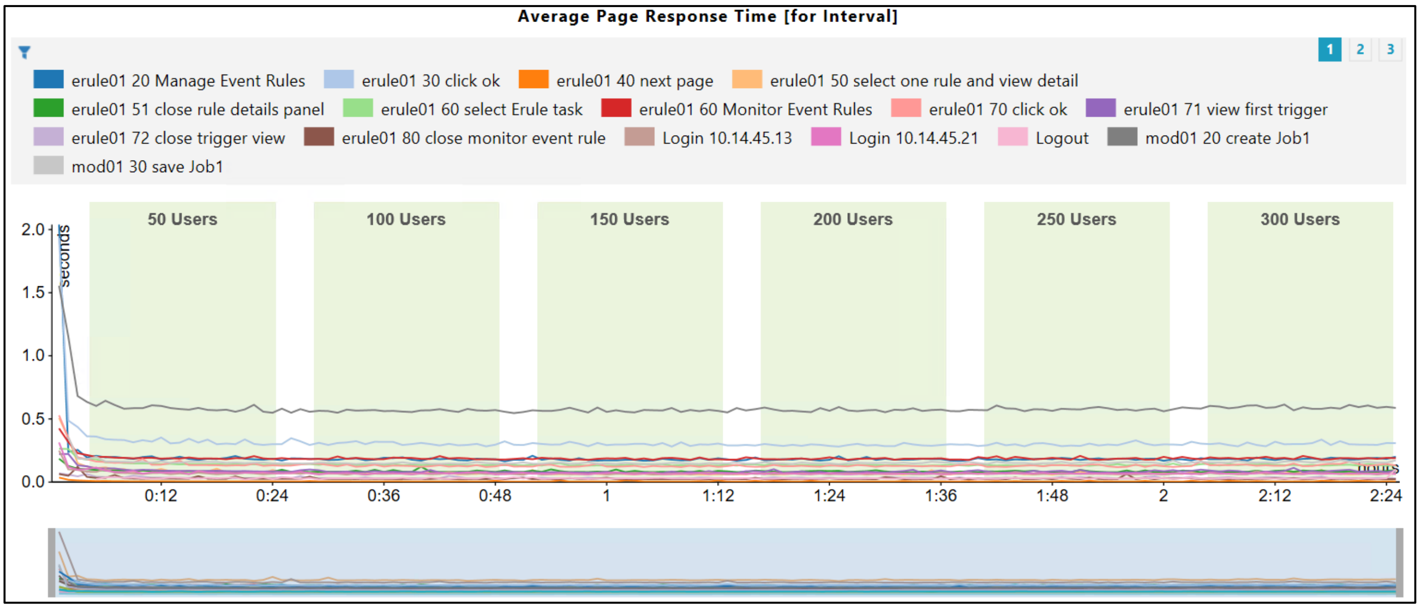


Figure 22. First set of response time trend during increasing workload

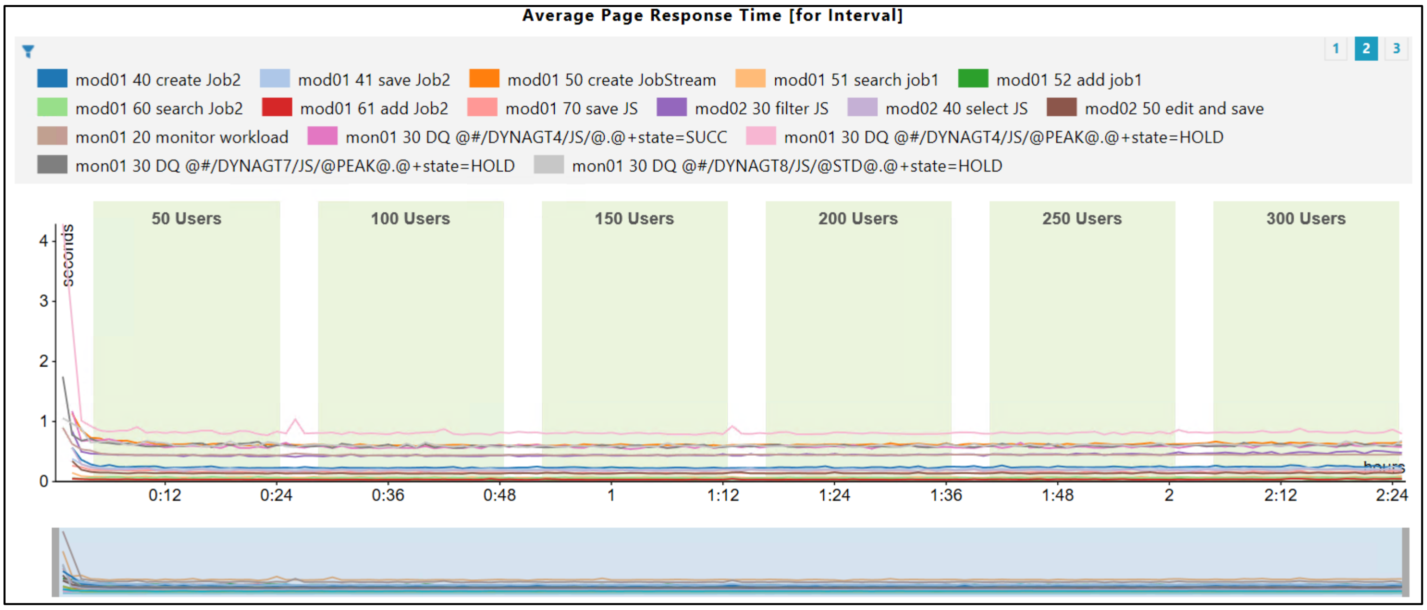


Figure 23. Second set of response time trend during increasing workload

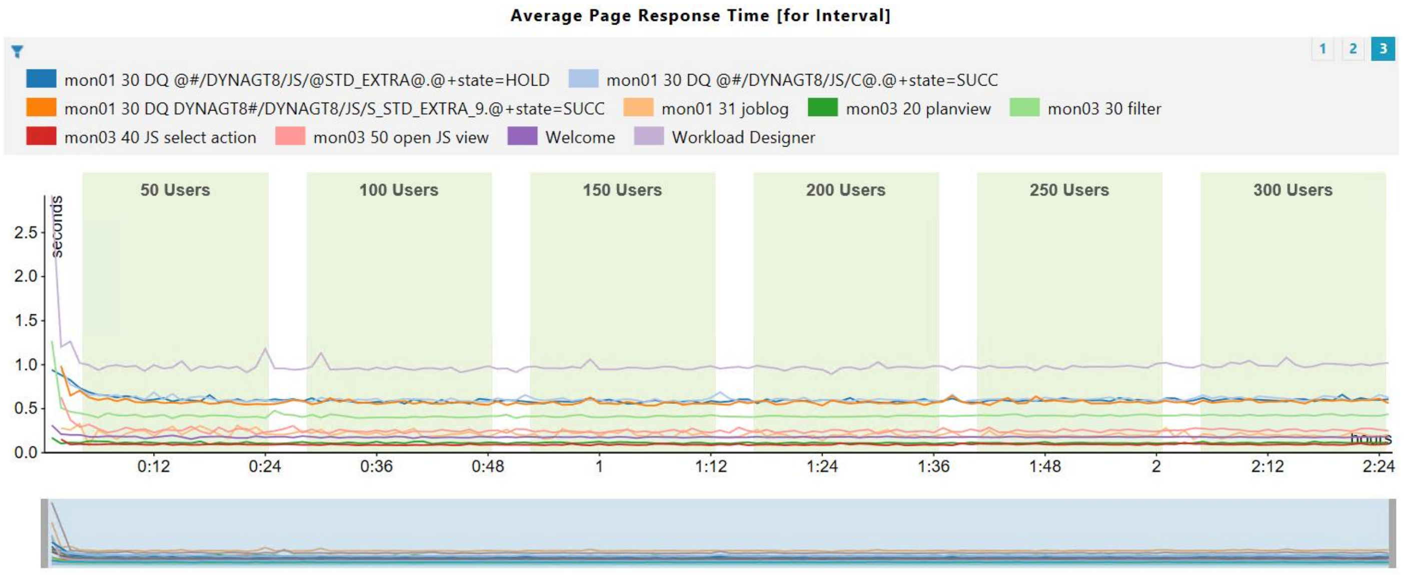


Figure 24. Third set of response time trend during increasing workload

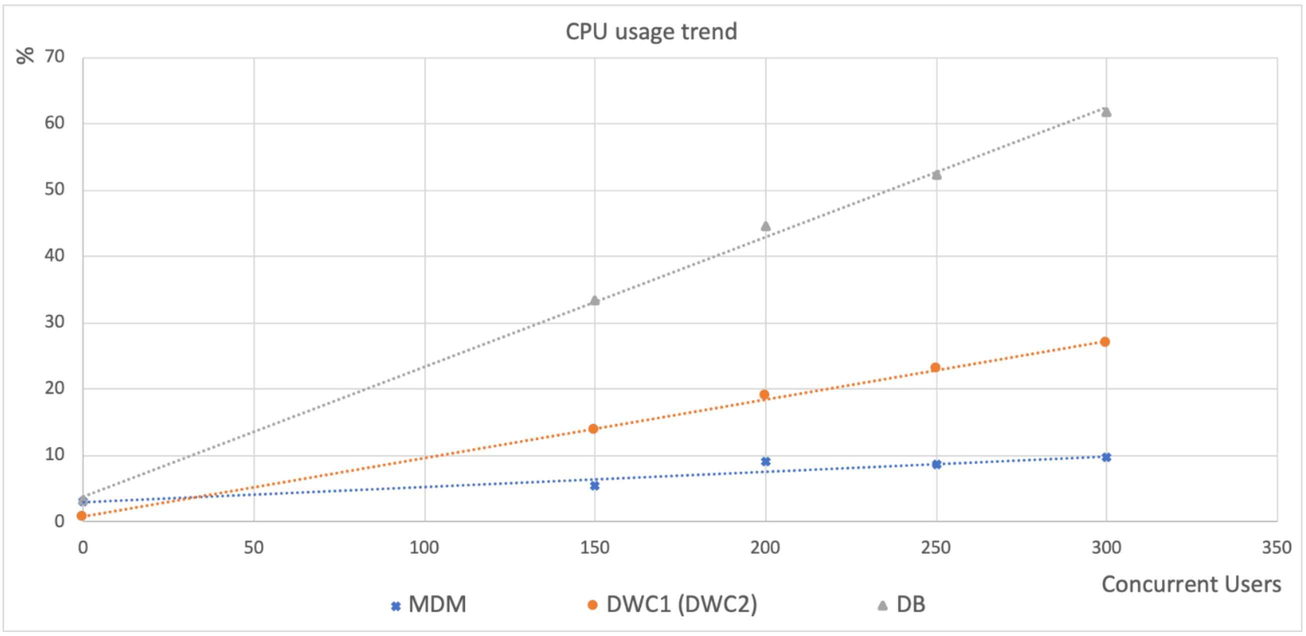


Figure 25. CPU utilization during DWC concurrent users test. The workload is distributed over two DWC nodes

The above results show an equivalent behavior in terms of throughput and resource utilization behavior with respect previous releases based on Dash technologies. Moreover, it confirms how the CPU usage is linear within the increasing number of concurrent users on all the main topological nodes.

It is also interesting to highlight how some performance improvements have been added in the Workload Scheduler 9.5 FP2 if compared with 9.5 GA and 9.5 FP1 versions, especially for the average response time of the pages related to the Mon01_951jobs scenario (Monitor Job Scenario with retrieve of job log).

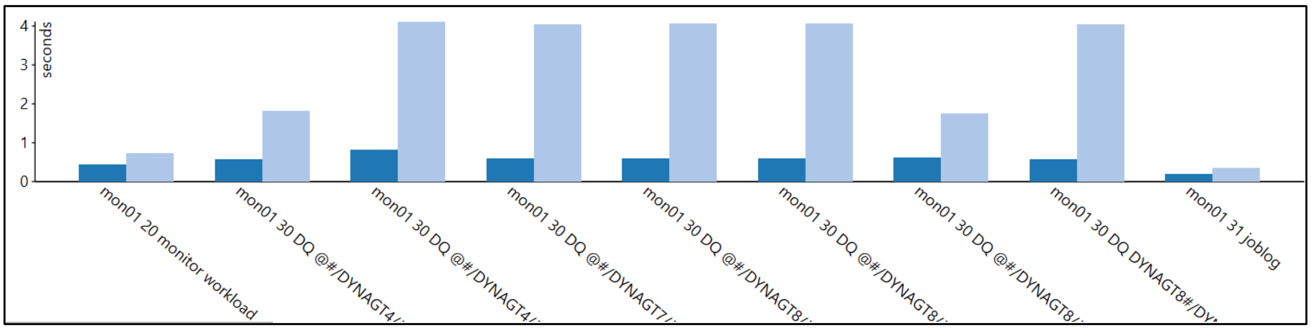


Figure 26. Monitor Jobs scenario - average response time comparison between 9.5 FP2 and 9.5 FP1

where the blue bars are related to the 9.5 FP2 run while the light blue bars are related to the 9.5 FP1.

3.4.3 Scheduling Workload for IBM Workload Scheduler on IBM Cloud Private

Starting from 9.5 version, the deployment of Workload Scheduler is available also for IBM Cloud Private.

IBM Cloud Private provides an integrated environment for managing containers that includes the container orchestrator Kubernetes, a private image repository, a management console, and monitoring frameworks. With

IBM Cloud Private, it is possible to deploy the IBM Workload Scheduler components as Helm charts to quickly configure and run them as Docker container applications in a Kubernetes cluster. Moreover, it is possible to manage the IBM Workload Scheduler components from the IBM Cloud Private dashboard or from the command-line interface. For further details about the IBM Cloud Private solution refer to online documentation.

The main scope of the current section is to document which are the overall scheduling performance of an IBM Workload Scheduler installation on IBM Cloud Private solution in a configuration with persistent storage defined on NFS Server solution.

The IBM Cloud Private environment and topology used for the performance scheduling test is described in the table below:

ICP Role	OS	CPU	RAM	HDD
Master Node	Ubuntu 18.04.1 LTS	8	32 GB	500 GB
Proxy Node		2	16 GB	500 GB
Worker Node 1		6	32 GB	500 GB
Worker Node 2		6	32 GB	500 GB

Table 5. IBM Cloud Private topology used to host IBM Workload Scheduler

Also in this case, all tests described in this section were executed on VMs with virtual CPUs and RAM assigned exclusively (reserved resources) and the database used for both the Master Domain Manager and the Dynamic Workload Console components was DB2 v11.1.4.4a installed on a dedicated VM with 8 vCPUs and 32GB RAM.

As persistent volumes for all the product components (Master Domain Manager, Dynamic Workload Console and Dynamic Agent) the NFS Server solution has been used. To avoid issues during the daily activity, it is highly suggested:

- to configure the persistent volume to use the version 3 of the NFS server
- to set the local lock parameter to all
- to assure to have a limited network latency between the NFS Server and the IBM Cloud Private components (in the test environment used, the average network latency between the Worker Nodes and the NFS Server was less than 0.3 ms The considerations done about the network latency impacts in a workload scheduler environment with a heavy workload available in the section 3.4.4 of Workload Scheduler v9.4 Fix Pack 3 performance report are applicable to 9.5.x version too.

The following is an example of .yaml file that shows how the persistent volumes have been configured in this environment:

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: <release_name>-data
  labels:
    <volname>: <volume_name>
spec:
  capacity:
    storage: 15Gi
  accessModes:
    - ReadWriteOnce
  mountOptions:
    - nfsvers=3
    - local_lock=all
  nfs:
    path: <nfs_server_path>
    server: <nfs_server_IPaddress>
```

Figure 27. yaml file example for persistent storage

In this configuration, a daily plan of around 46000 jobs distributed among tens of dynamic agents was executed. Note that fault-tolerate agents are not supported by Workload Scheduler installed on IBM Cloud Private.

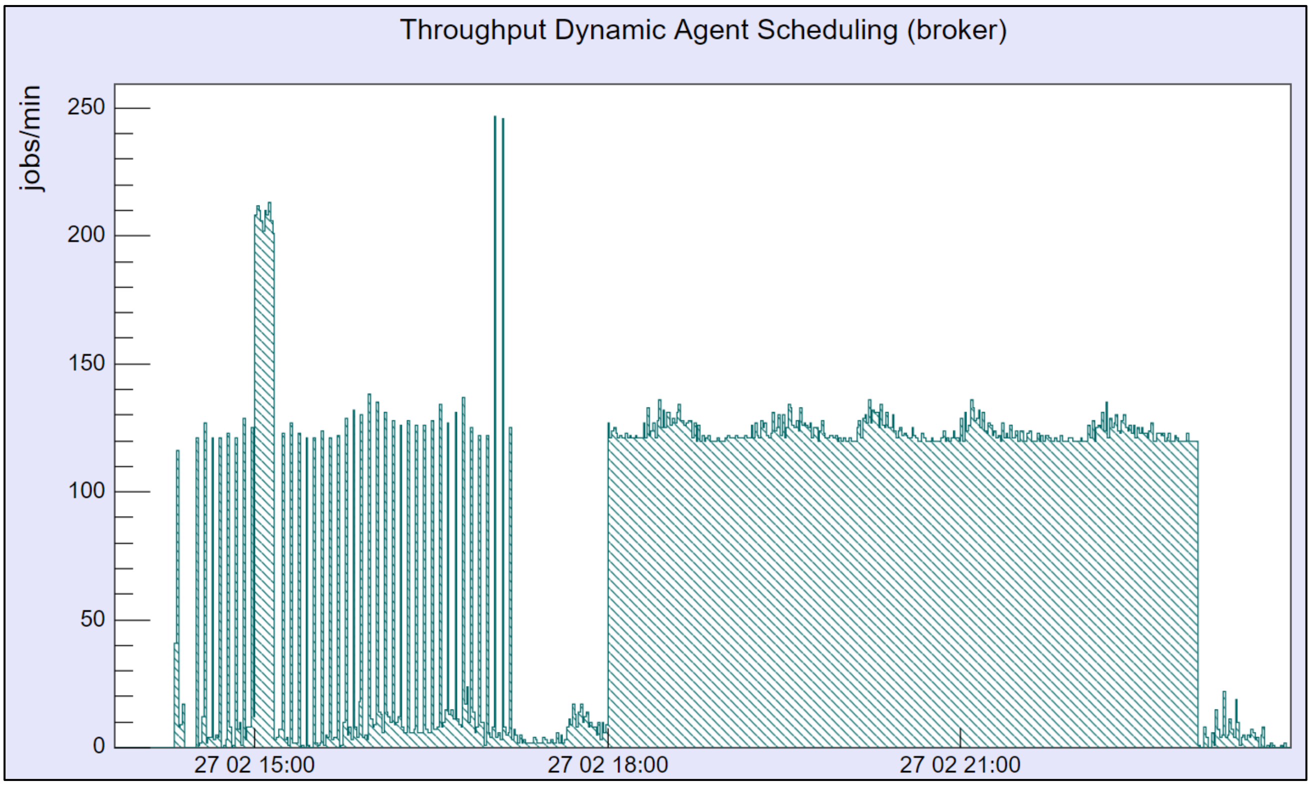


Figure 28. Dynamic agent daily throughput on ICP environment

The workload represented in **Figure 28. Dynamic agent daily throughput on ICP environment** 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 scheduling throughput represented in this section did not suffer any queuing phenomenon (incoming and outgoing throughput are equivalent).

From the workload scheduler user perspective that means, again, no substantial delay in the scheduling of a job when the same job is ready to start (see **Figure 29. Average job schedule delay over time on ICP environment**) and no substantial latency in the job and job stream status update from the Dynamic Workload Console (see **Figure 30. Job plan status update delay over time on ICP environment**), except for a sporadic unexpected peak.

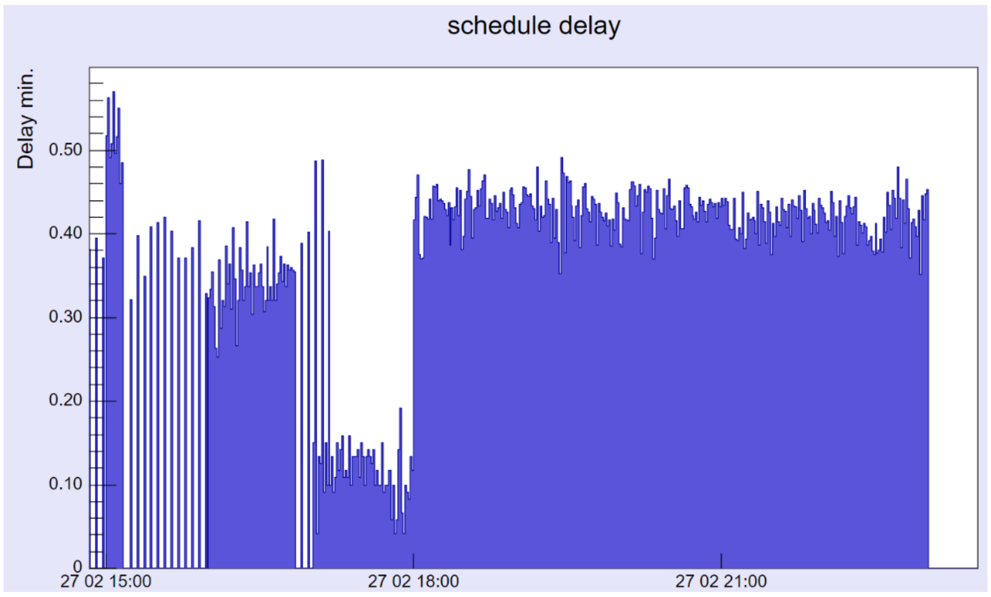


Figure 29. Average job schedule delay over time on ICP environment

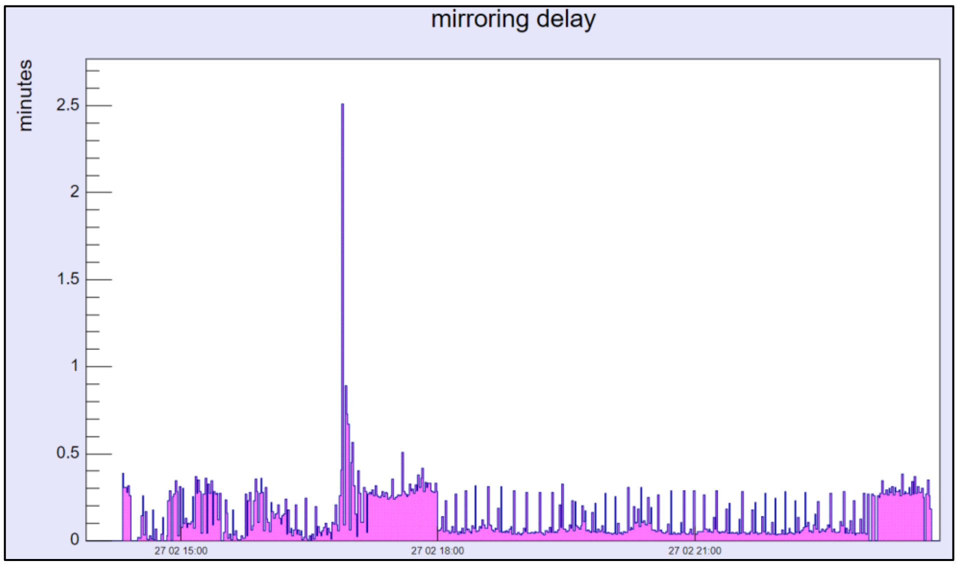


Figure 30. Job plan status update delay over time on ICP environment

The above results evidence the promptness of Workload Scheduler installed on ICP in the case of a workload with a peak of around 200 jobs/min with a dynamic scheduling delay of less than 1 minute.

It is interesting to highlight that the dynamic scheduling delay and job plan status update delay start to grow if the incoming workload is increased, especially during the job submission peak. This behavior depends on the fact that the Master Domain Manager `DATA_DIR` (where the Workload Scheduler queues and Symphony file reside) is located in the persistent volume, that is a network-based file system. In this specific configuration, the network bandwidth and the network latency play a key role in limiting the product throughput capabilities if compared with an on-prem installation for which the Master Domain Manager `DATA_DIR` is typically defined on a local attached disk.

4 Best Practices

4.1 Scheduling

Workload Scheduler software offers many 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 give a higher level of versatility to accomplish different business scenarios, there are some recommendations that must be considered before planning to adopt them to orchestrate the scheduler completely in case of a heavy 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 of 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 of 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.

The benchmark was based on 6000 file creation rules, defined for 4 dynamic agents with more than 1.2x10^5 files created in one hour. The file creation rate was increased during the test to saturate the event processor capacity. It is meaningful to compare the event processor capacity in two different architectures (**Table 6**).

Environment architecture	Events per minute
Power7 AIX env	70
VMWare Linux env	390

Table 6. Event processor capacity in terms of maximum throughput (events per minute) comparison

The test was executed during a 240 x 2 jobs/min schedule for dynamic agents and fault tolerant agents. **Figure 31** shows the number of actions that the event processors was able to trigger and, in this case, they are mapped (one to one) to job submission (see **Figure 32**). The focus of this section was to demonstrate the event processor throughput stressed with the file creation rule. It was evaluated that the type of rule does not impact the throughput. There is a specific scenario, status change event rules, whose throughput is constrained by the total number of event rules (see *IBM Workload Scheduler 9.3.0.1 Capacity Planning Guide*) causing enqueuing on the *`monbox`*.

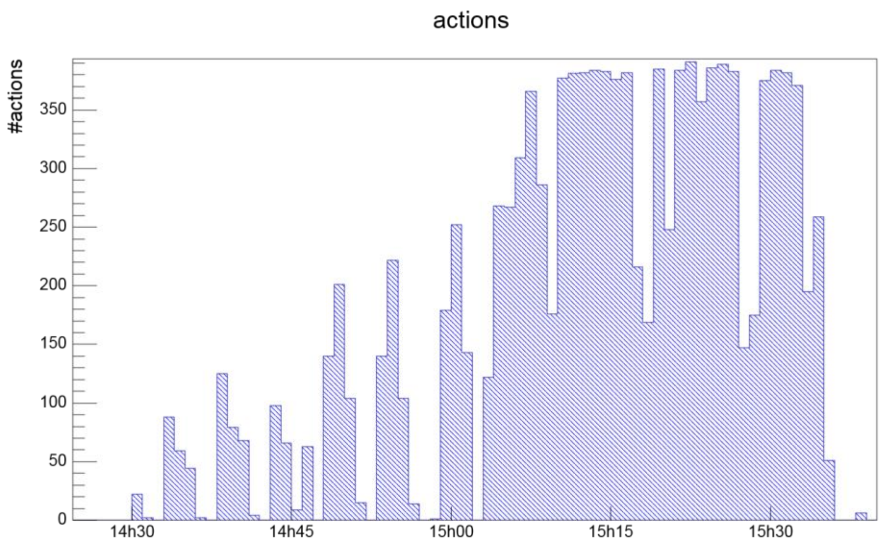


Figure 31. Number of actions triggered by the event processor

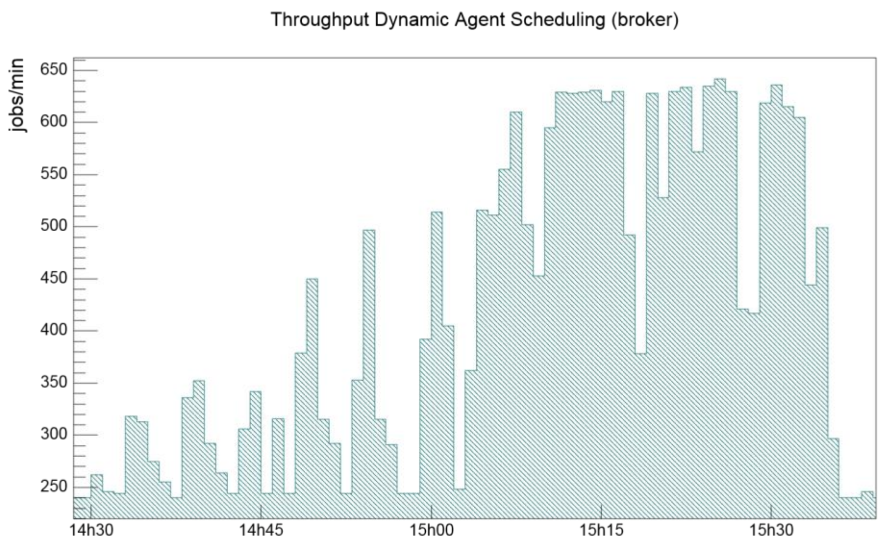


Figure 32. Dynamic Domain Manger job submission throughput in the file creation event rule scenario

While planning this kind of solution, not only the event processor capacity must be considered but also its additional resource utilization in terms of CPU.

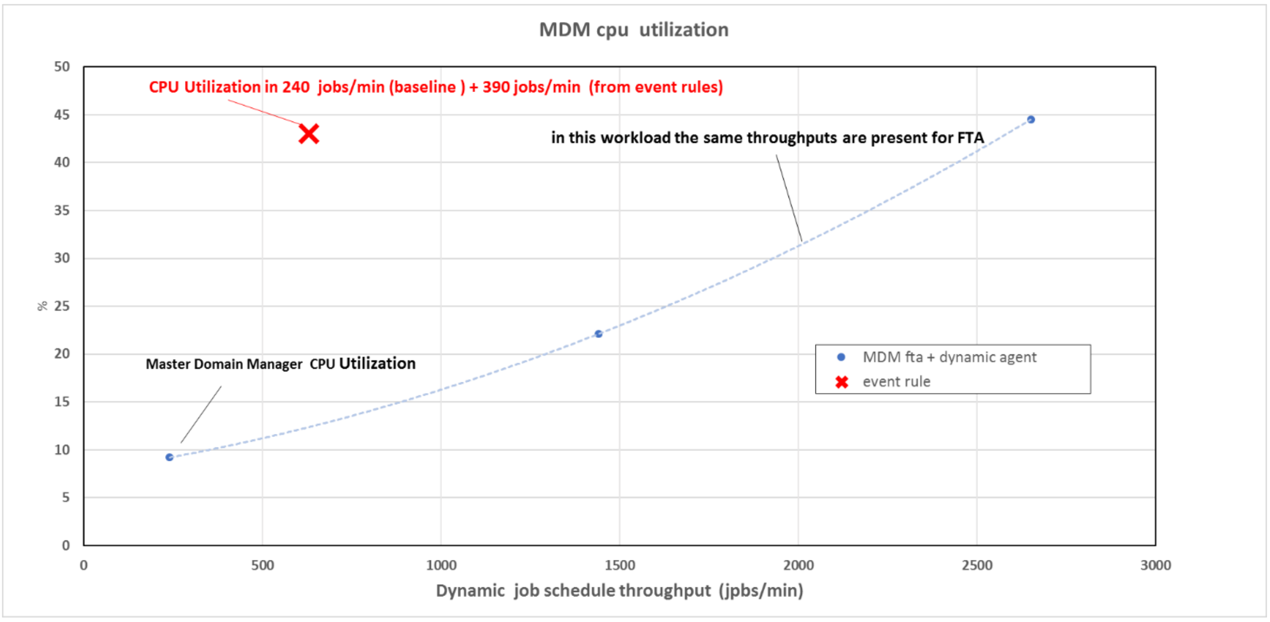


Figure 33. MDM CPU utilization comparison with and without event rule processing

It could be noted that to obtain the same throughput there is an additional CPU consumption of about 30% on the master domain machine.

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 3.4.1 already includes a component of this type acted against fault tolerant agents. This feature has a different impact on the performance if used with a dynamic agent. In 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 server about file status. The server workload throughput is ruled by three 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 around network latency 0.1 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) a 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.

This scenario was applied to 2000 jobs scheduled with 2000 different file dependencies, for which `T = bm check file = 300`, as shown in the example above. File dependency release was triggered in three blocks of 400, 800 and 800 files when the system was charged with 1200 jobs/min job submission (see **Figure *34***).

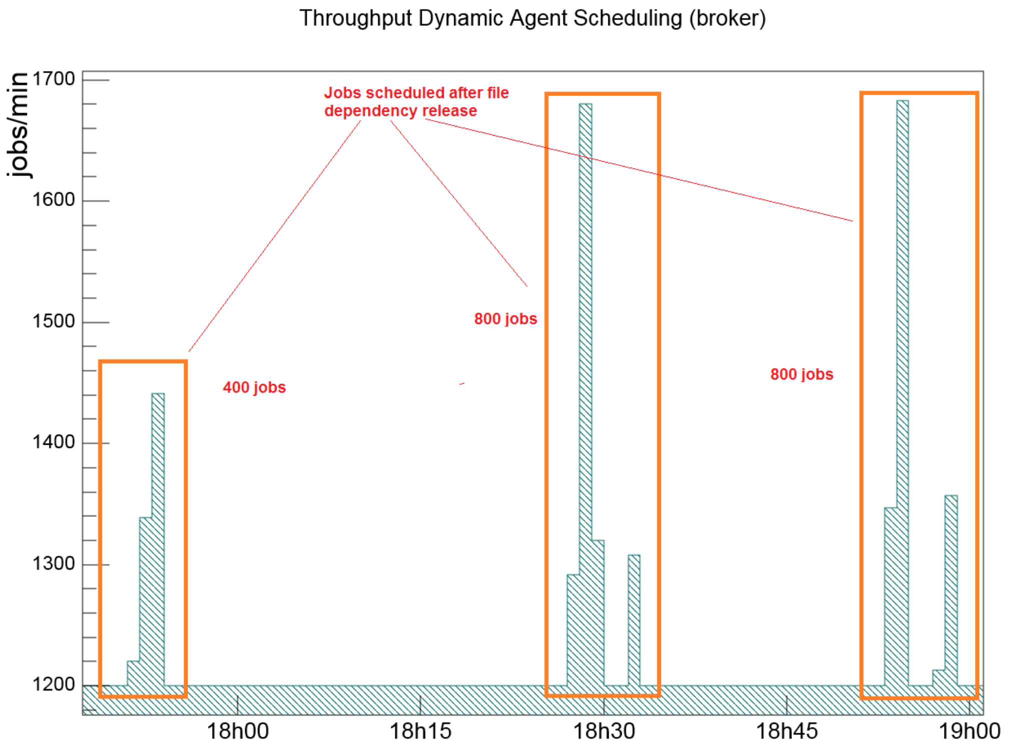


Figure 34. File dependency releases in a 1200 jobs/min workload as baseline

4.1.3 Scheduling using a start condition

It is also possible to schedule jobs using file detection as a trigger handled a by job mechanism. This is known as the Start Condition feature: a job called continues to run until a file match is detected.

The capacity of this job stream submission is strictly related to the global schedule capacity. The advantages include leveraging of job control and monitoring.

4.1.4 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 jobs in the job streams and internal and/or external dependencies.

4.1.5 Scheduling using “every” option

“Every” feature allow 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 (few minutes) to avoid to increase the number of jobs in a job stream (few hundreds of job could impact performances). There are several methodologies that could help the driving of business need:

- 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 to perform job log retrieval for archived plans. The following parameters, in the `<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 boot and removes jobs from the table accordingly with 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, as shown by the example below.

During 1200 jobs/min constant workload, the dynamic job cleaning starts deleting almost 7x10^5 rows of jobs that were 10 days old and in successful state. The delete operation in this case took almost 7 minutes to complete, causing an intensive I/O activity on the database, which impacted overall product throughput capabilities.

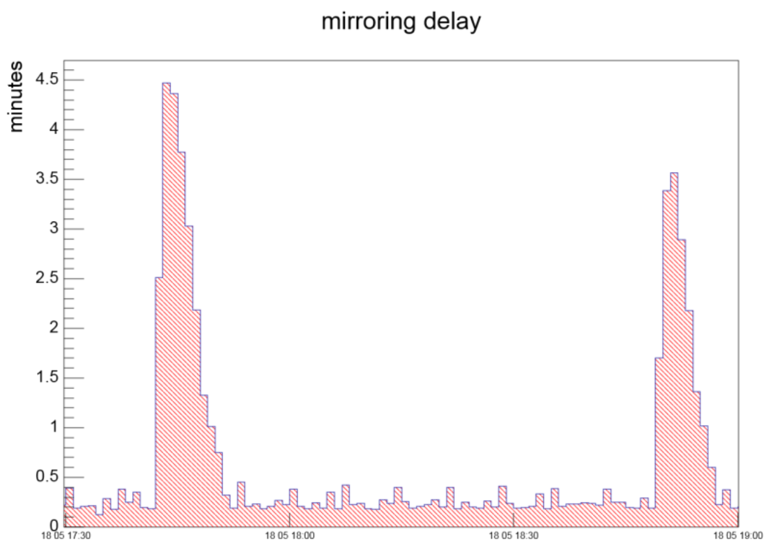


Figure 35. Plan update delay while dynamic job table is being cleaned up

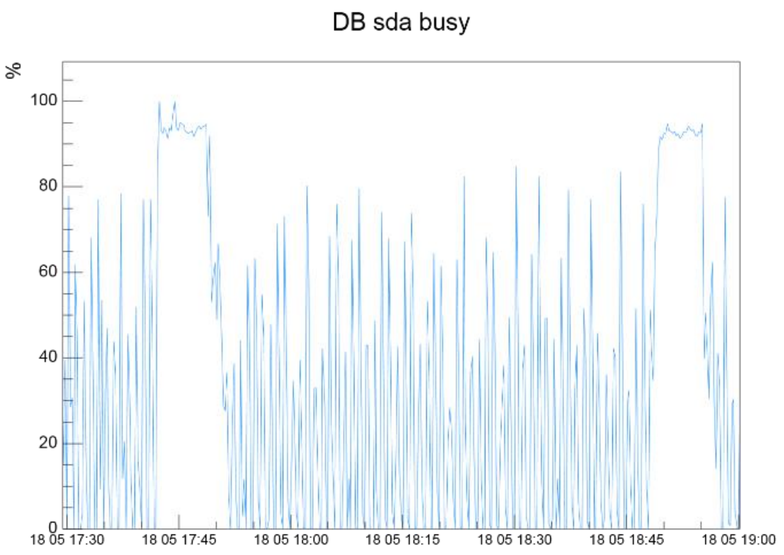


Figure 36. Database Server disk busy while dynamic job table is being cleaned up

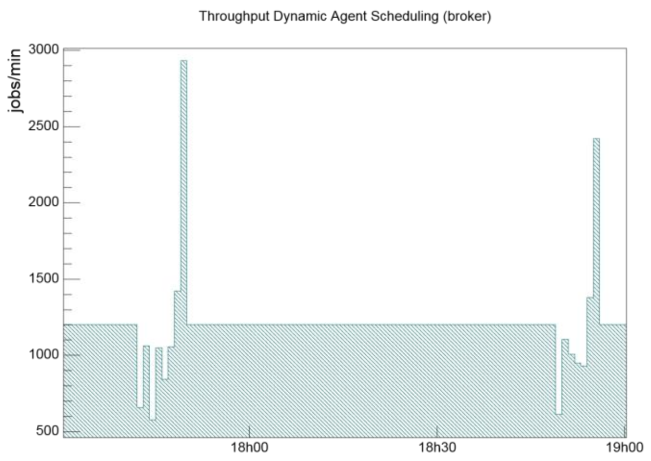


Figure 37. Impact Dynamic Domain Manager throughput capabilities while dynamic job table is being cleaned up

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. In this case, the following configuration should be applied in the `<DATA_DIR>/broker/config/JobDispatcherConfig.properties`:

```
SuccessfulJobsMaxAge = 360 (15 days)
```

```
MoveHistoryDataFrequencyInMins = 720 (12 hours)
```

```
SCHEDULE MDMWS#CLEANUP_DWB_DB
DESCRIPTION "Added by composer."
ON RUNCYCLE RULE1 "FREQ=DAILY;INTERVAL=1"
AT 2345
:
EU-HWS-LNX47#CLEANUP_DWB_DB
 SCRIPTNAME "/opt/IBM/TWA/TDWB/bin/movehistorydata.sh -dbUsr db2user -dbPwd password
-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
```

Figure 38. Example of Job Stream that handles the cleanup of Dynamic Domain Manager table entries

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 configuration. 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 10** 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 39** could be used as a reference while planning a solution, because they are the output of I/O Industry standard benchmark, such as IOzone, and they can be considered key performance indicators to be used for comparison.

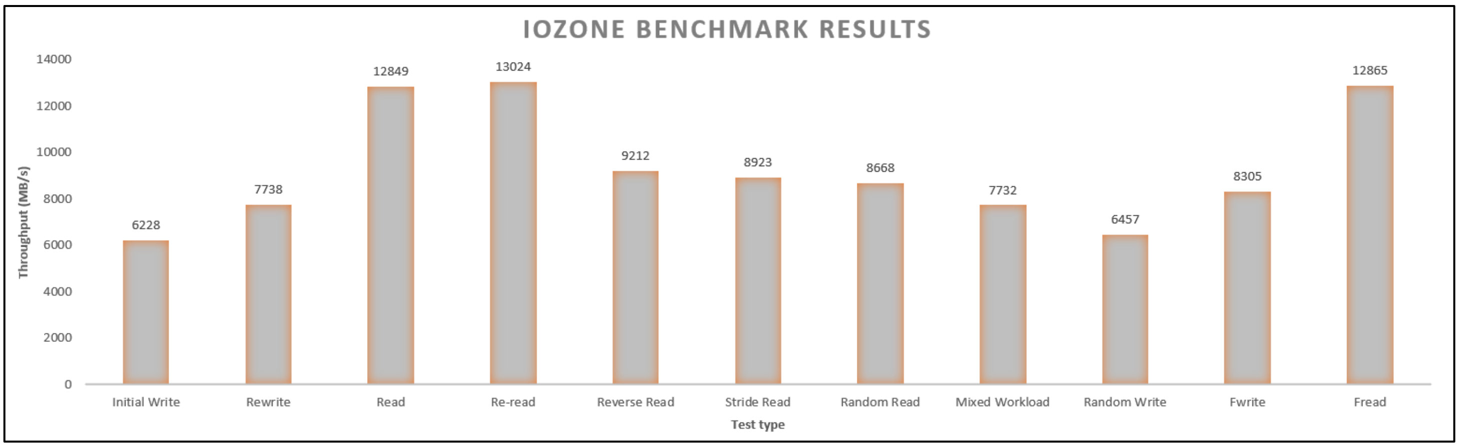


Figure 39. IOzone benchmark output run with "-R -l 5 -u 5 -r 4k -s 100m –F file1 ...file5" options

Moreover, we have monitored VMWare ESXi server disk latency over time while Workload Scheduler daily production plan was running. In average the disk latency was negligible, with some sporadic peaks lower than 5 ms. Disk latency needs to be monitored constantly 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:

Concurrent users range x DWC node	1 – 10	10 -50	50 - 150
DWC heap size	2 GB	4 GB	6 GB

Table 7. Dynamic Workload Console WebSphere Application Server Liberty Base heap configuration

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

Table 8. Engine 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.

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

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

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 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 by means of a parallelism (multithreading) and caching. The former is 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 configuration (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:
`<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 12c Enterprise Edition 12.2.0.1.0 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.

5.4.4 Comprehensive configuration and tuning

	Parameter		Value	Comment
Dynamic Workload Console	WebSphere Application Server JVM max heap = min heap		Required: 4096 for [10, 50] users per node Suggested: 6144 for [50, 150] users per node	
	WebSphere Application Server JVM options		-Xgcpolicy:gencon –Xmn1024m	-Xmn parameter value should be ¼ of total heap size. This parameter should be set to 1536m if heap = 6144
	WebSphere Application Server Liberty Base JDBC max Connections		300	datasource.xml
Master Domain Manager	<DATA_DIR>/localopts	batchman settings	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	
	WebSphere Application Server Liberty Base Configuration	JVM arguments	-Djava.awt.headless=true -Xdisableexplicitgc -Xgcpolicy:gencon –Xmn 1024m	- Xmn 1536m if heap size = 6144
		Heap	Required: 4096 for [100, 200] jobs/min Suggested: 6144 for >200 jobs/min	
		Data Source	JDBC Type = 4	datasource.xml
			Connection Timeout = 180	
			Max Connections = 300	
			Min Connections = 0	
			Purge Policy = EntirePool	
			Reap Time = 180	
			Statement Cache Size= 400	
DB (db2)	LOGPRIMARY		200	MB total transaction log space
	LOGFILSIZ		3500	
	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	182000	
		PAGESIZE	4096	
	TWS_BUFFPOOL_TEMP	NPAGES	500	
		PAGESIZE	16384	
	TWS_BUFFPOOL	NPAGES	50000	
		PAGESIZE	8192	
Dynamical Workload Broker	<DATA_DIR>/broker/config/JobDispatcherConfig.properties	Historical data management	`MoveHistoryDataFrequencyInMins=720` `SuccessfulJobsMaxAge = 360`	
	<DATA_DIR>/broker/config/ResourceAdvisorConfig.properties		`MaxAllocsPerTimeSlot = 1000`	
			`TimeSlotLength = 10`	
			`MaxAllocsInCache = 50000`	
	<DATA_DIR>/usr/servers/engineServer/resources/properties/TWSConfig.properties	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`	

Table 9. Main configurations and tunings

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 concurrent 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 the version 9.5.0.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 core.

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)
10K jobs (50% FTA +50% DYN) per day (8 jobs/min), 10 concurrent users			
1 Node	**WS Engine, RDBMS, DWC**	4	16
250K jobs (50% FTA +50% DYN) per day (175 jobs/min), 50 concurrent users			
2 Nodes	**WS Engine, DWC**	4	16
	RDBMS	4	16
500K jobs (50% FTA +50% DYN) per day (350 jobs/min), 100+ concurrent users			
3 Nodes	**WS-Engine**	8	32
	RDBMS	8	32
	DWC	5	20

Table 10. Capacity planning samples

7 Notices

7.1 *Trademarks*