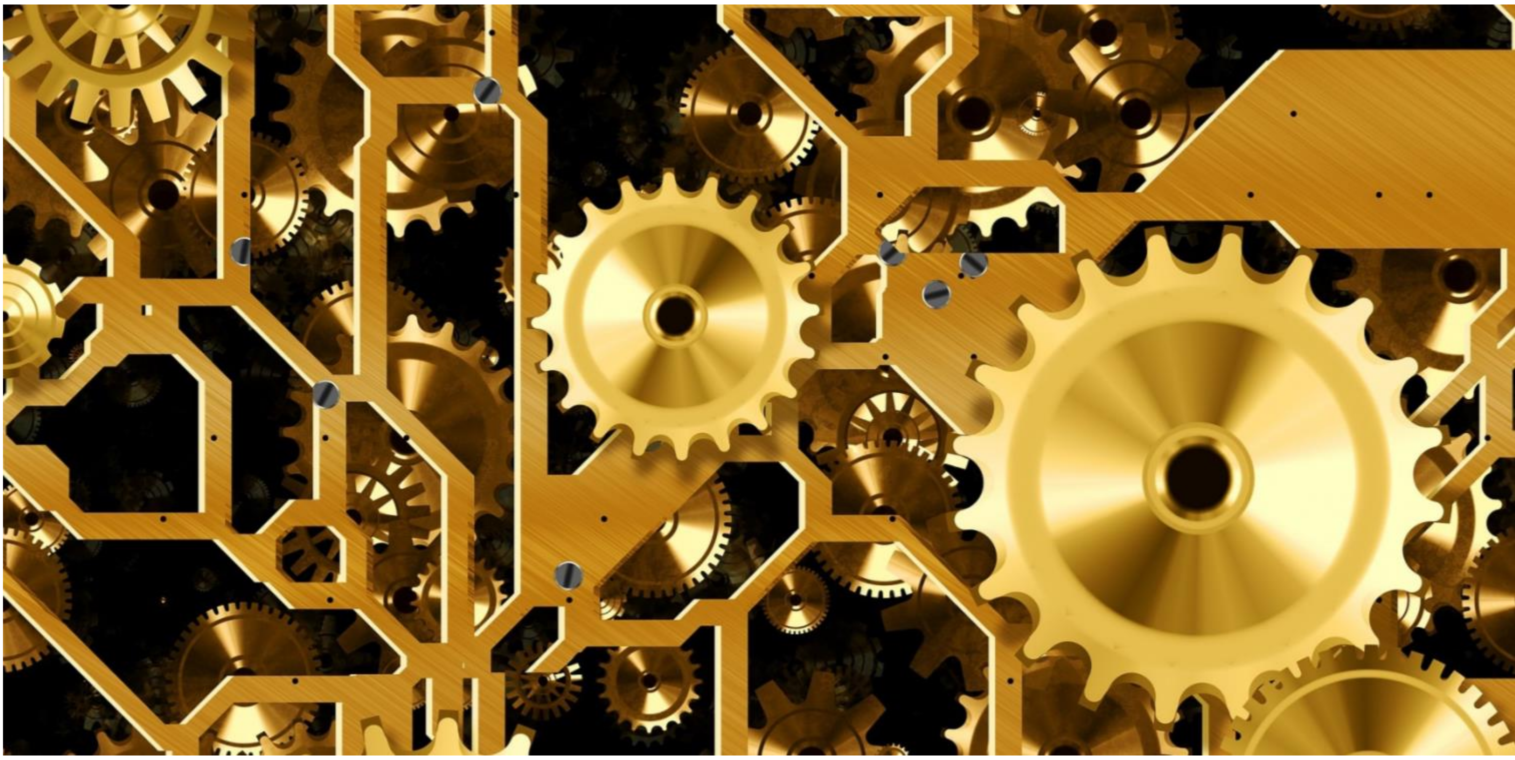

Workload Scheduler Version 9.4 FP3 Performance Report

an IBM + HCL product

Document version 1.0

Pier Fortunato Bottan
Giorgio Corsetti
Workload Automation Performance Team - HCL Rome Lab

This edition applies to version 9, release 4, fix pack 3 of Workload Scheduler and to all subsequent releases and modifications until otherwise indicated in new editions

CONTENTS

Contents 3
List of Figures 4
List of Tables 4
1. Introduction 5
1.1. What's new since version 9.4 5
2. Scope 5
2.1. Executive summary 5
3. Performance Test 5
3.1. Test Approach 5
3.2. Environment 6
3.3. Test tools 11
3.4. Test Benchmarks and Results 12
3.4.1. Scheduling Workload 12
3.4.2. Plan View 17
3.4.3. Job Stream View 18
3.4.4. High network latency test 18
4. Best Practices 19
4.1. Scheduling 19
4.1.1. Scheduling using event rules: Event Processor Throughput 20
4.1.2. Scheduling using file dependencies 21
4.1.3. Scheduling using start condition 22
4.1.4. Scheduling using conman sbs 22
4.2. Dynamic domain manager table cleanup policy 23
5. Recommendations 25
5.1. CPU capacity 25
5.2. Storage 25
5.3. Memory 26
5.4. Tunings and settings 26
5.4.1. Data Source 26
5.4.2. Plan Update (mirroring) 27
5.4.3. Comprehensive configuration and tuning 27
6. Capacity Plan Examples 29
7. Notices 31
7.1. Trademarks 32

LIST OF FIGURES

Figure 1. Overall deploy view of test environment 7
Figure 2. Dynamic Workload Console node configuration 8
Figure 3. Master Domain Manager node configurations 9
Figure 4. Database node configurations 10
Figure 5. Storage Solution 11
Figure 6. Dynamic agent daily throughput. The total job scheduled in a day is around 4.3 x10^5 13
Figure 7. Zoomed view of dynamic agent scheduling to distinguish the compound of different workload type 13
Figure 8. Dynamic agent schedule: ad hoc submission with 1000 scheduled jobs 14
Figure 9. Dynamic agent schedule: jobs with internal and external conditional dependencies (3200 total jobs) 14
Figure 10. Dynamic agent schedule: jobs belonging to a critical path (Workload Service Assurance) 14
Figure 11. Fault tolerant agent schedule throughput in daily workload (10^5 jobs) 15
Figure 12. Average jobs schedule delay over time 15
Figure 13. Jobs plan status update delay over time 16
Figure 14. CPU utilization at master domain manager and database nodes sides vs different workload 16
Figure 15. Job Stream network for the plan view scenario 17
Figure 16. Comparisons of rendering time for graphical view reload 17
Figure 17. Network latency impacts on Dynamic Domain Manager throughput capabilities 19
Figure 18. Network latency impacts on jobs plan status update 19
Figure 19. Number of action triggered by event processor 20
Figure 20. Dynamic Domain Manger jobs submission throughput in the file creation event rule scenario 21
Figure 21. MDM CPU utilization comparison with and without event rules processing 21
Figure 22. File dependency releases in a 1200 jobs/min workload as baseline 22
Figure 23. Plan update delay while dynamic jobs table is being cleaned up 23
Figure 24. Database Server disk busy while dynamic jobs table is being cleaned up 24
Figure 25. Impact Dynamic Domain Manager throughput capabilities while dynamic jobs table is being cleaned up 24
Figure 26. Example of Job Stream that handles the cleanup of Dynamic Domain Manager table entries 25
Figure 27. IOzone benchmark output run with “-R -l 5 -u 5 -r 4k -s 100m –F file1 ...file5” options 25

LIST OF TABLES

Table 1. Software level of code 6
Table 2. Daily plan workload composition 12
Table 3. Job Stream filtered value for the Plan View scenario 17
Table 4. Event processor capacity in terms of max throughput (events per minute) comparison 20
Table 5. Dynamic Workload Console WebSphere Application Server heap configuration 26
Table 6. Engine WebSphere Application Server heap configuration 26
Table 7. Main configurations and tunings 29
Table 8. Capacity planning samples 30

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 are making it increasingly difficult to maintain the same 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 9.4

In the last year and half, three different fix packs have been released for 9.4 major release.

For more details about Workload Scheduler new features, see the Summary of enhancements in the online product documentation in IBM Knowledge Center:

- 9.4 FP1
- 9.4 FP2
- 9.4 FP3

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, V9.4.0.3, executed in a test environment based on VMWare - Linux x86 platform having comparable resources assignment with respect previous performance environment (see Workload Scheduler v9.4 performance report) based on Power7 - AIX platform.

Those performance results could be summarized:

- Consolidate previous performance achievements in terms of throughputs
- New tunings for software performance improvements considering new test environment architecture
- New key performance indicators

3. Performance Test

3.1. Test Approach

As specified in section 2.1 most of performance test focus was specific for new performance test environment used to validate Workload Scheduler 9.4.0.3 release. The guideline was to keep the performance benchmark results, collected in previous releases and different platform, as key performance indicators. Scheduling throughputs, resources consumption and reliability are continuously certified assuring no degradation with respect previous releases. Specific tests have been implemented for validating performance improvements like the “Plan View” and “Job Stream View” applications.

In addition, accordingly with continuous Workload Scheduler Customer interactions, other specific workloads have been benchmarked in the performance environment. The latter is continuously tested with daily plan running every day on 2 production-like environments (one with DB2 and one with Oracle).

In this context it has been applied a continuous monitoring with special focus on key performance indicator (scheduling and mirroring throughput, average delays, internal queues sizes) and on HW main resources (CPU, memory, disk busy) to prevent memory leaks, unexpected HW consumptions 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(R) 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 7.3 Kernel 3.10.0-514	
RDBMS	IBM DB2 v11.1.2.2	Oracle 12c Enterprise Edition 12.1.0.1.0
J2EE	IBM WebSphere® Application Server 8.5.5.13 with IBM Java 8.0.5.6	
LDAP	IBM Directory Server 7.2	
Jazz™ for Service Management	JazzSM 1.1.2.1 DASH 3.1.3 CP6	
WA	9.4.0.3	

Table 1. Software level of code

The HTTPS protocol was used and an IBM HTTP Server with IHS WebSphere Application Server Plugin acted as a load balancer with “Random” policy to distribute user load on the Dynamic Workload Console servers. The procedure described at the following link:

http://www.ibm.com/support/knowledgecenter/SSGSPN_9.4.0/com.ibm.tivoli.itws.doc_9.4/distr/src_ad/ctip_config_ha_ovw.htm

was followed to set up a high availability configuration (also known here as cluster).

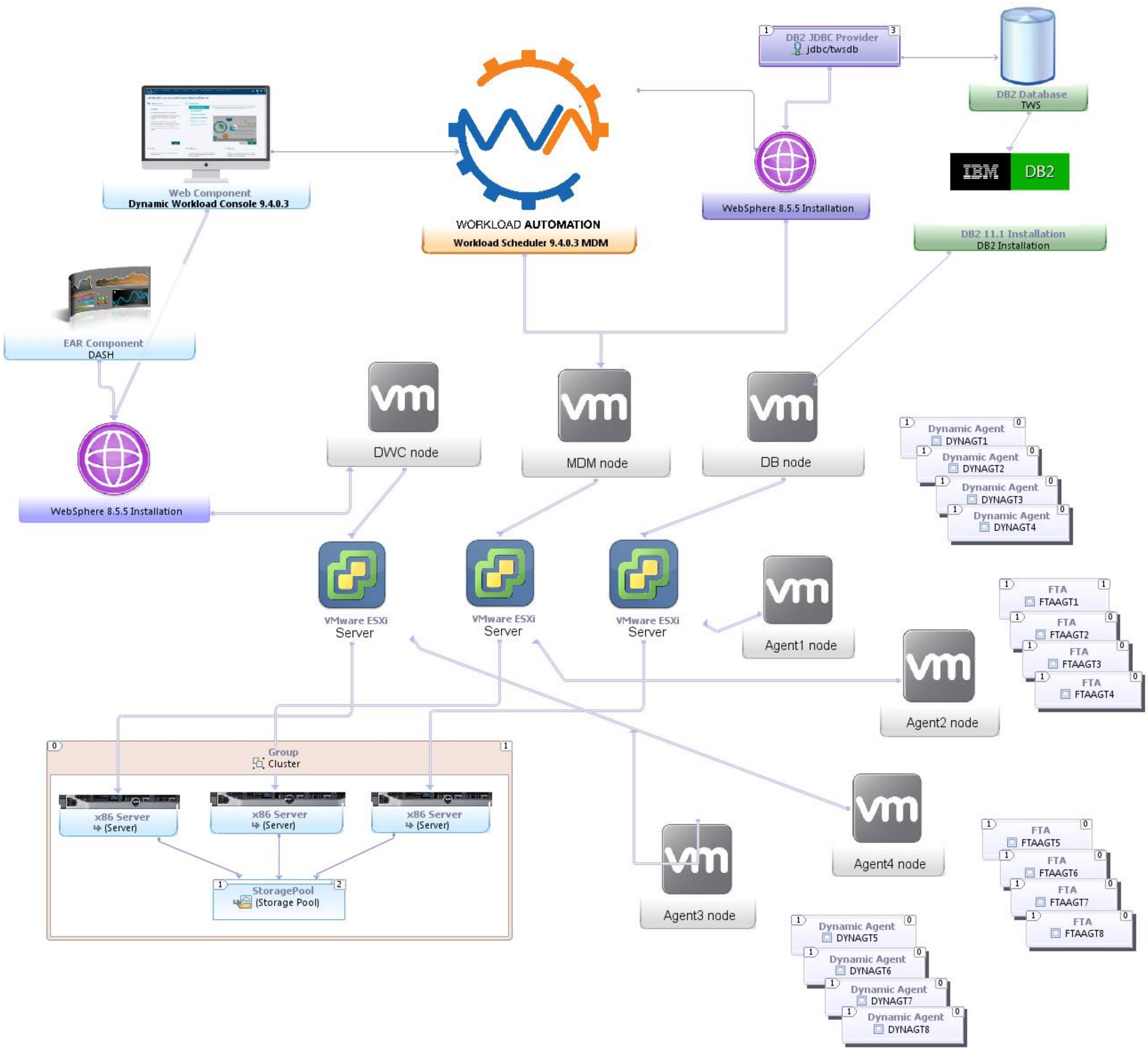

Figure 1. Overall deploy view of test environment

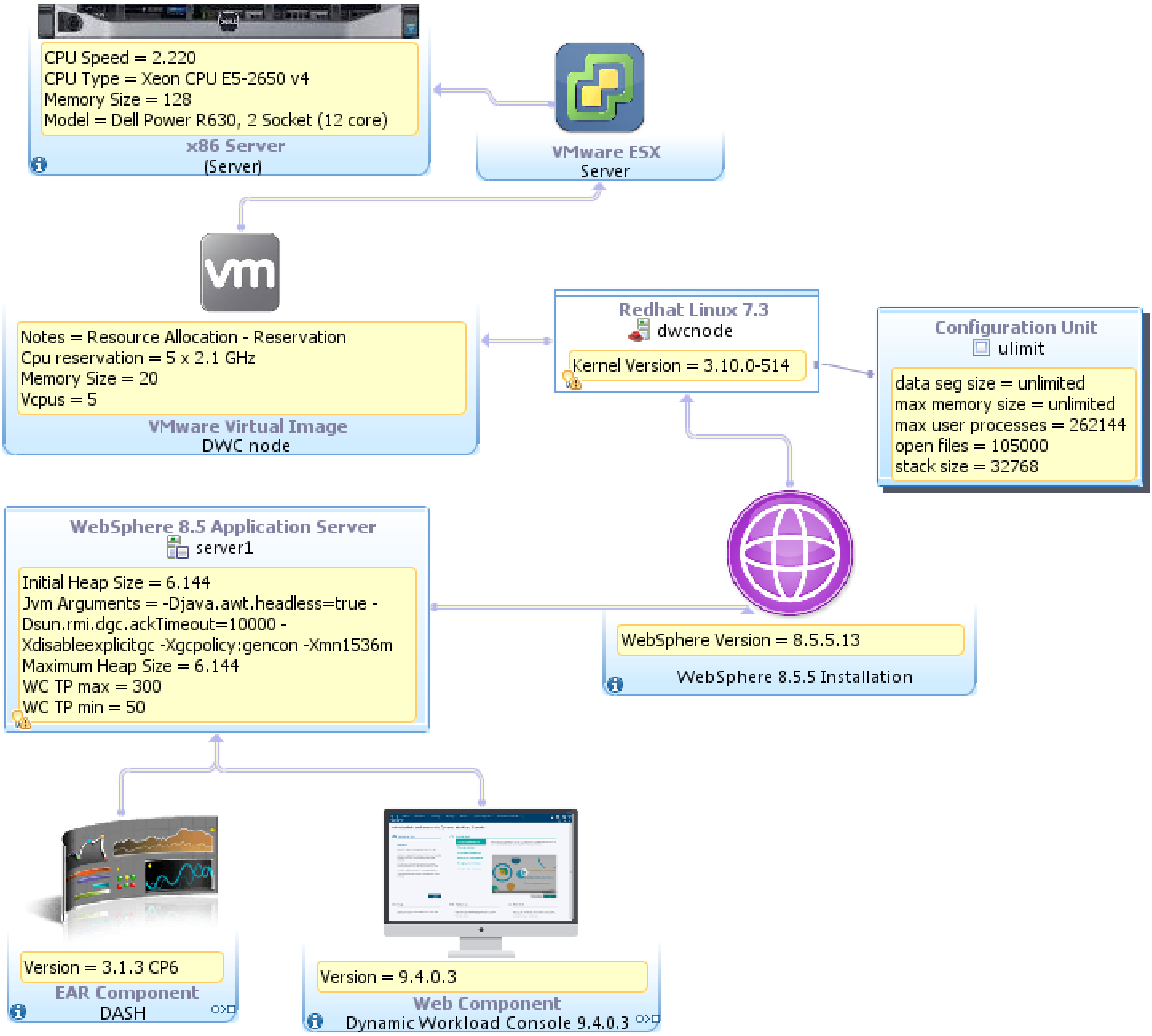

Figure 2. Dynamic Workload Console node configuration.

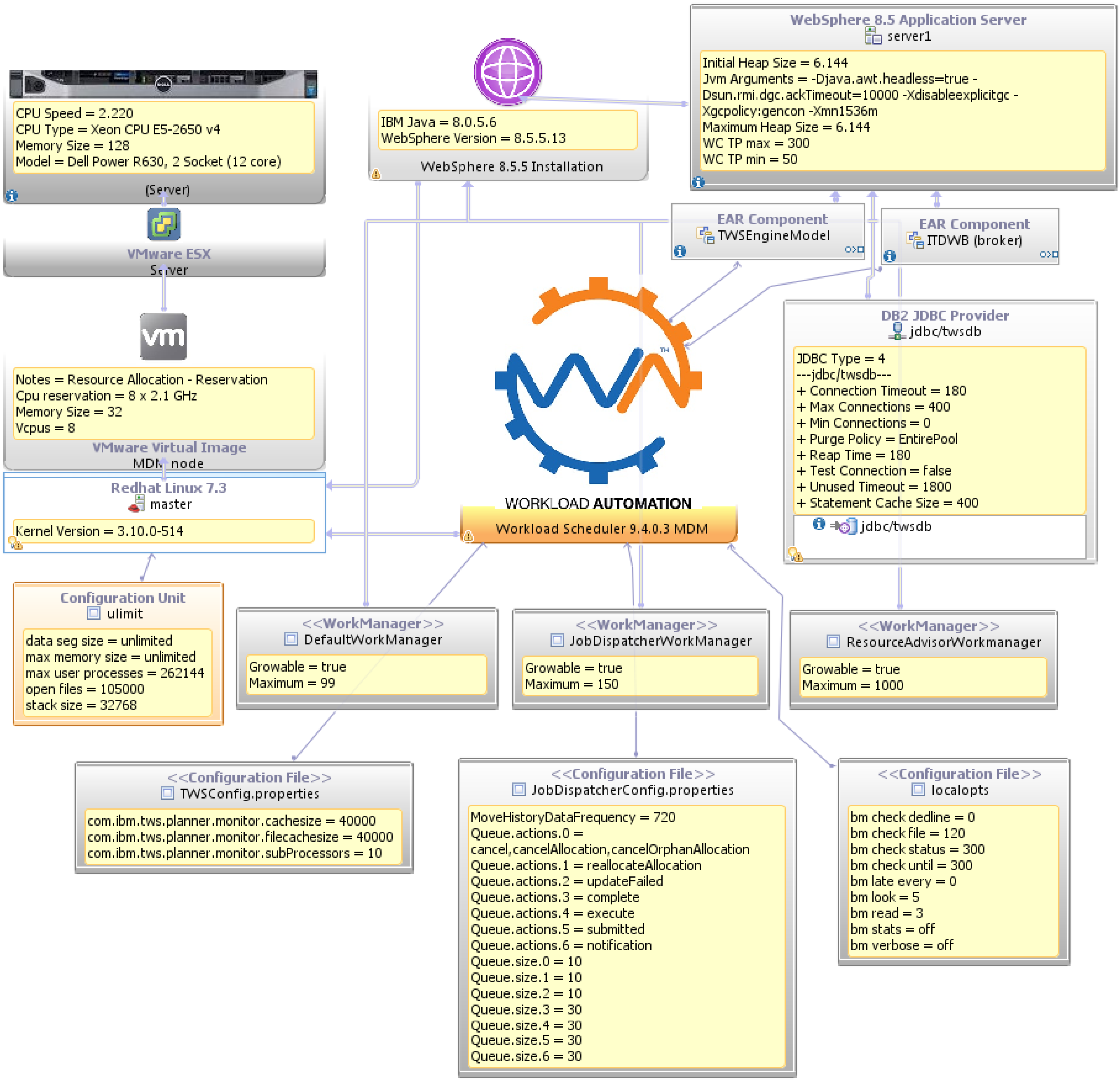

Figure 3. Master Domain Manager node configurations

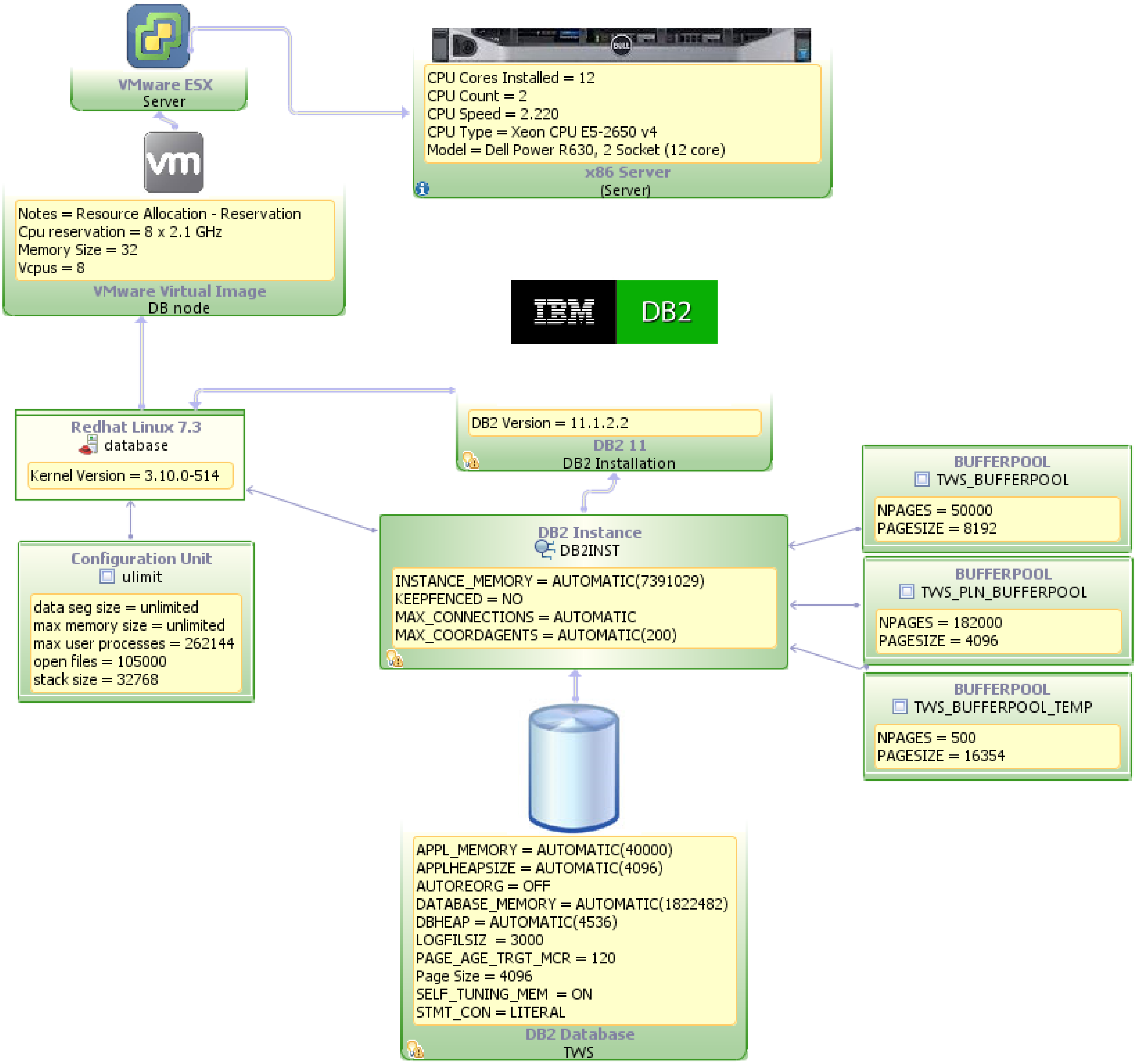

Figure 4. Database node configurations

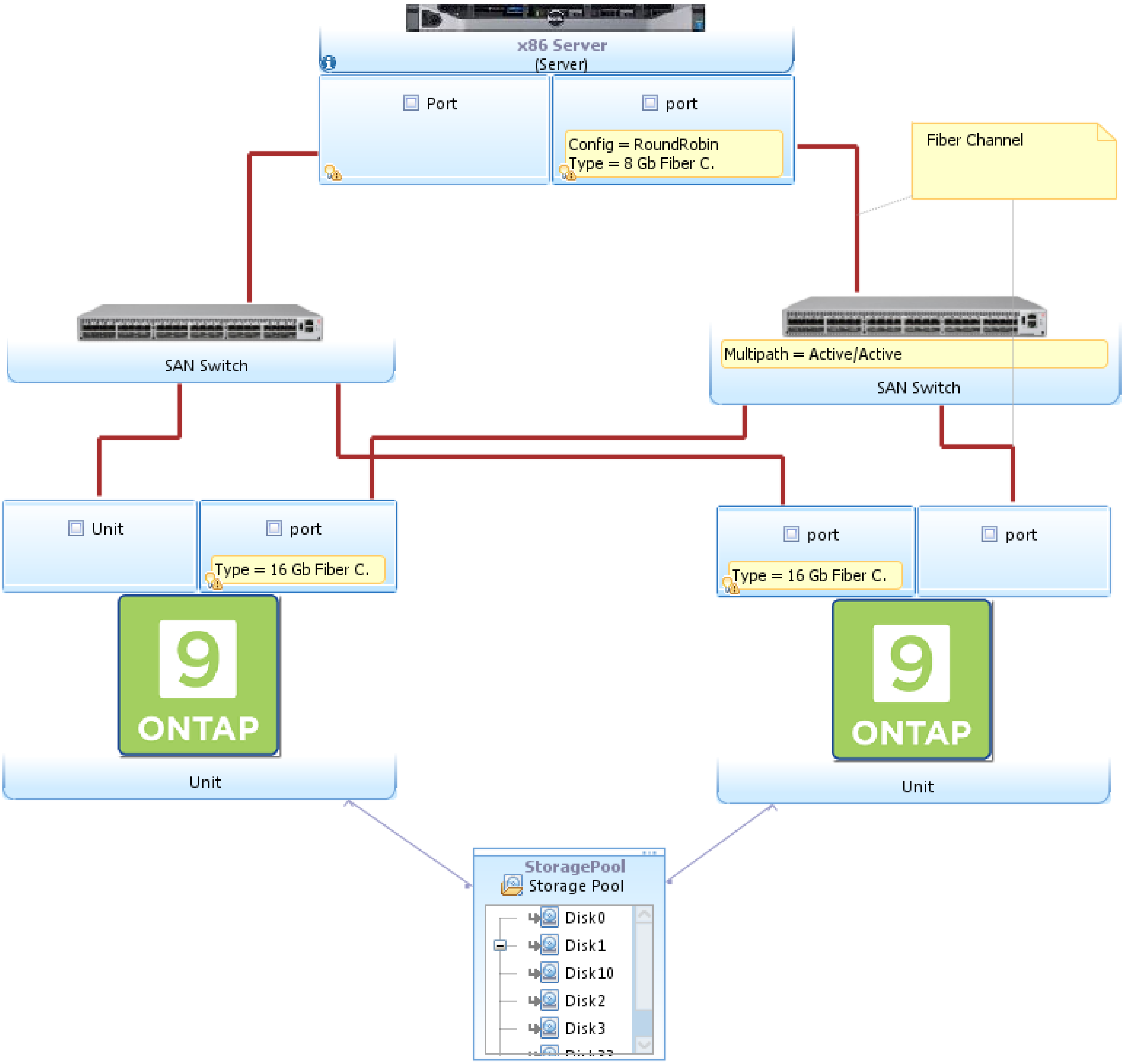

Figure 5. Storage Solution

3.3. Test tools

Rational Performance Tester (RPT) version 9.1.1 was used to generate traffic and run a multiple user scenario. RPT also provides a response time for each HTTP action on the browser by reporting the time spent on the server to process the request. RPT cannot determine the time spent by the browser to process data to be interpreted.

Standard monitoring tools and methodologies were used, such as nmon and IBM Support Assistant – Garbage Collection and Memory Visualizer. IOzone version 3.434 has been used to benchmark storage throughput.

The Perfanalyst tool v. 11.4 was used to control the middleware configuration and to analyse the DB2 snapshot.

WebSphere Application Server Performance Tuning Toolkit v.2.0 is an intelligent toolkit which helps in tuning the performance of WebSphere Application Server.

Statistical data processing and presentation have been done using the ROOT Data Analysis Framework (https://root.cern.ch/)

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 daily executed is around 530000 jobs per day.

Standard FTA and Dynamic Agent schedule

- Plan includes **124800 jobs scheduled in around 3 hours**. In particular, there are **48000 jobs** scheduled to be started in a 10 minutes peak. In addition, only for dynamic agent, around 361000 jobs scheduled in 5 hours (1200 jobs/min).

WSA - Critical path

- 48 complex patterns, composed by 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 generates 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) have been generated each hour, that means 1 event every 4 minutes on each of 16 agents.

Conditional Dependencies

- 5% of additional workload Additional 3200 jobs/800 job streams over 4 dynamic agents and 4 FTAs . This means that there are 100 JS for each agent, half of which has Internal Dependencies and half of which has External Dependencies. In the case with conditional dependencies, there are overall also 800 Join conditions.

***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 one from the others in a chain. In total, we had 1000 dynamic jobs 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 PVT Test environment (4375 jobs per agent). These 35000 job streams have a time dependency that is different FTA per FTA: the single block of 4375 job streams per agent is scheduled to start one hour after the other for 8 hours long

Table 2. Daily plan workload composition

This workload is used as standard benchmark for establishing key performance indicators whose baseline is continuously verified to track performance enhancements.

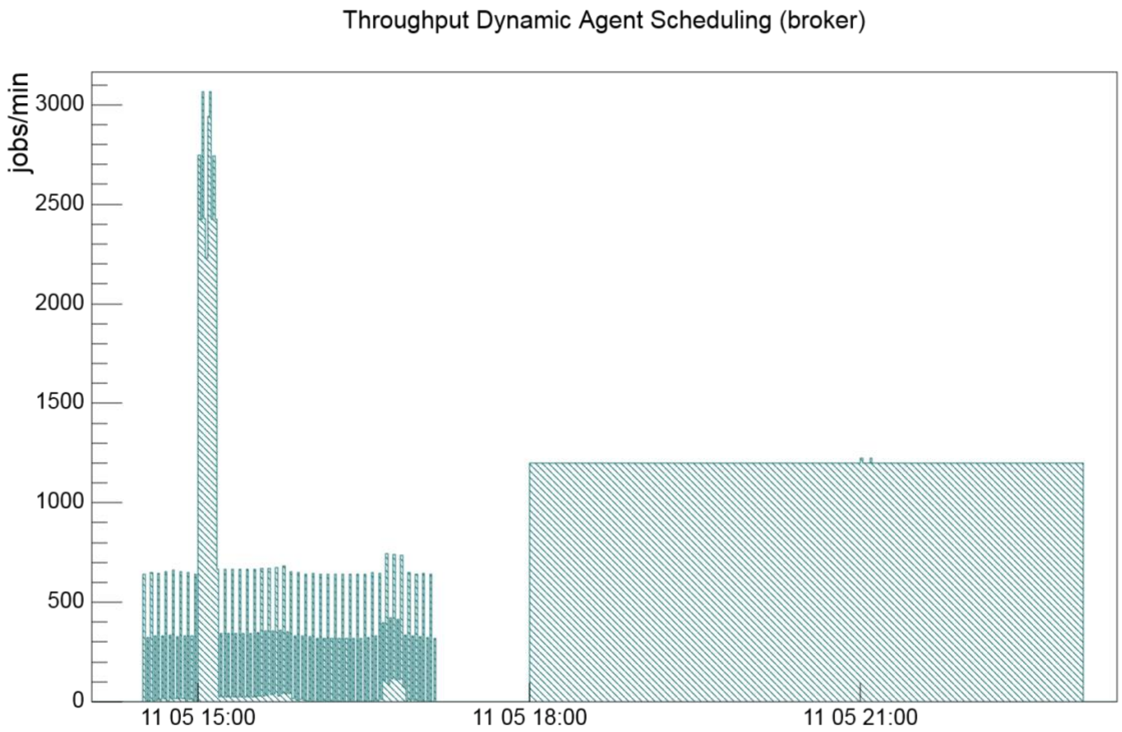

Figure 6. Dynamic agent daily throughput. The total job scheduled in a day is around 4.3 x10^5

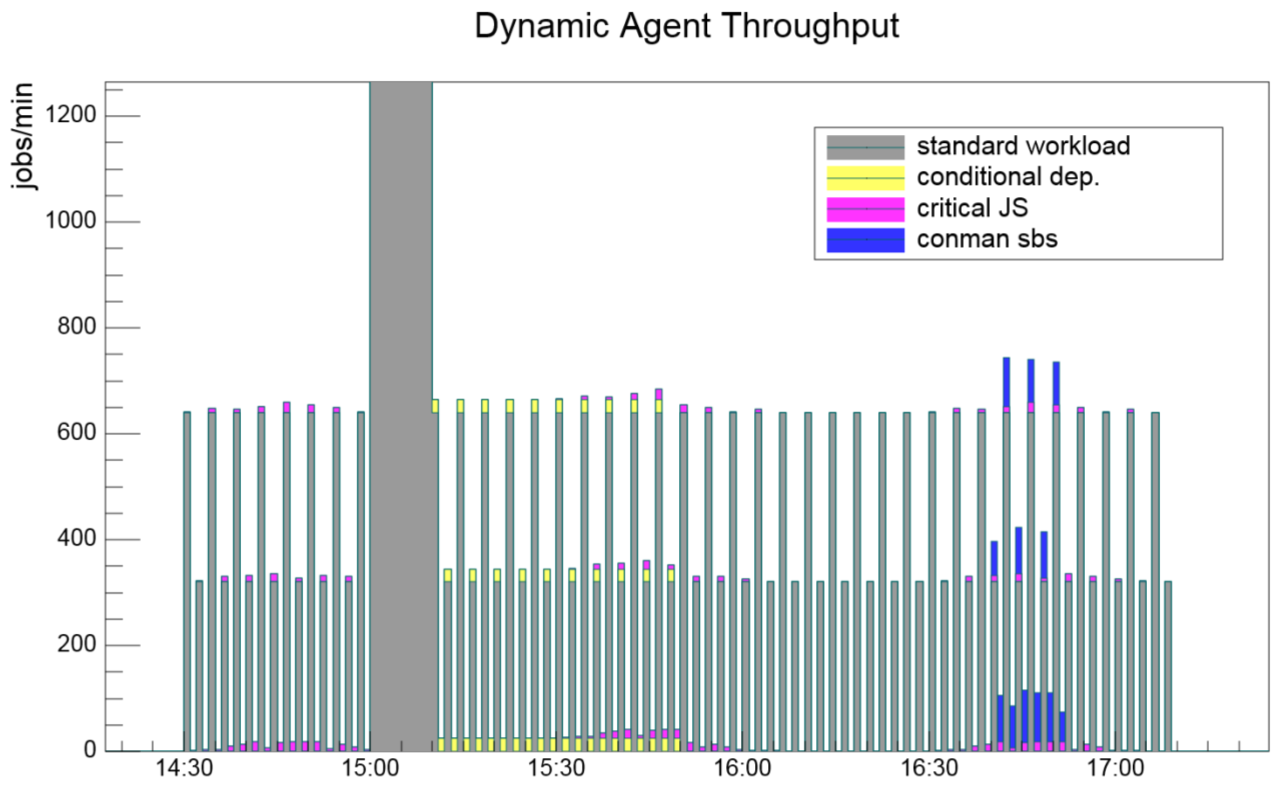

Figure 7. Zoomed view of dynamic agent scheduling to distinguish the compound of different workload type

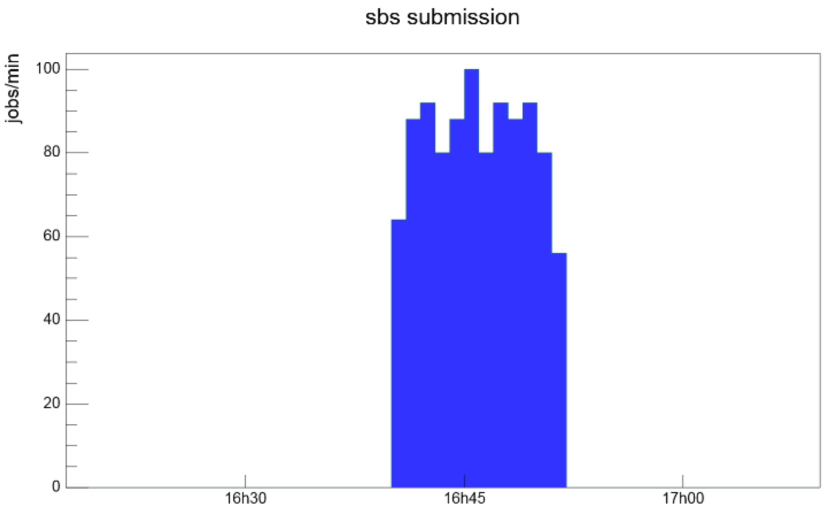

Figure 8. Dynamic agent schedule: ad hoc submission with 1000 scheduled jobs

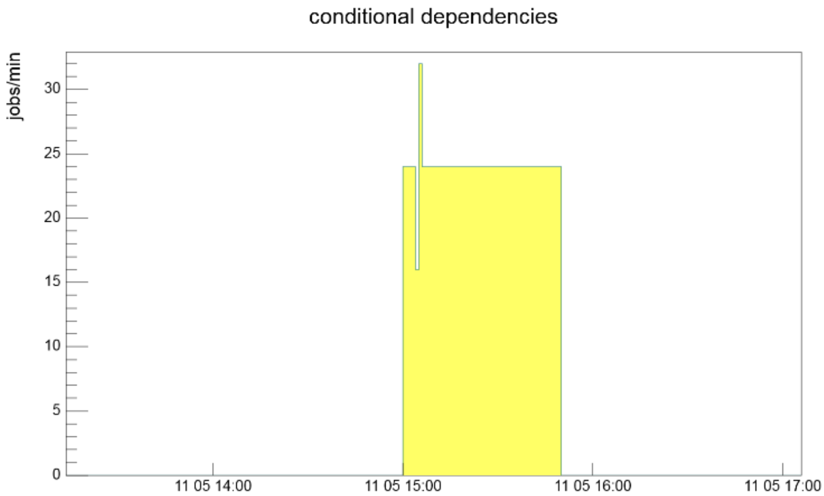

Figure 9. Dynamic agent schedule: jobs with internal and external conditional dependencies (3200 total jobs)

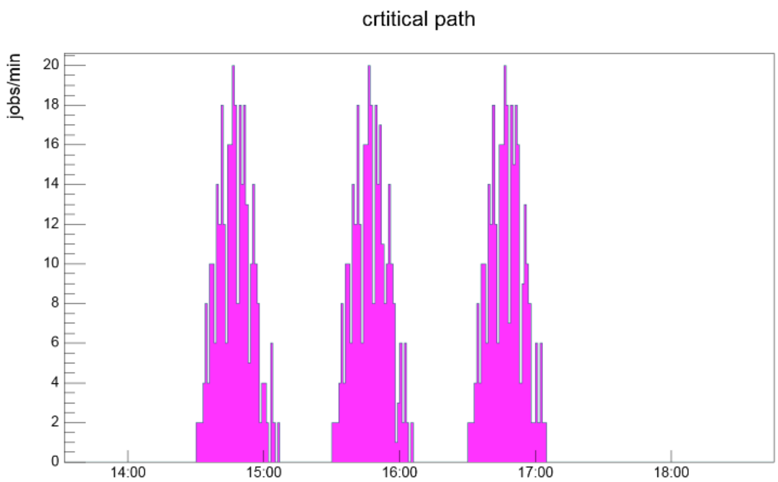

Figure 10. Dynamic agent schedule: jobs belonging to a critical path (Workload Service Assurance)

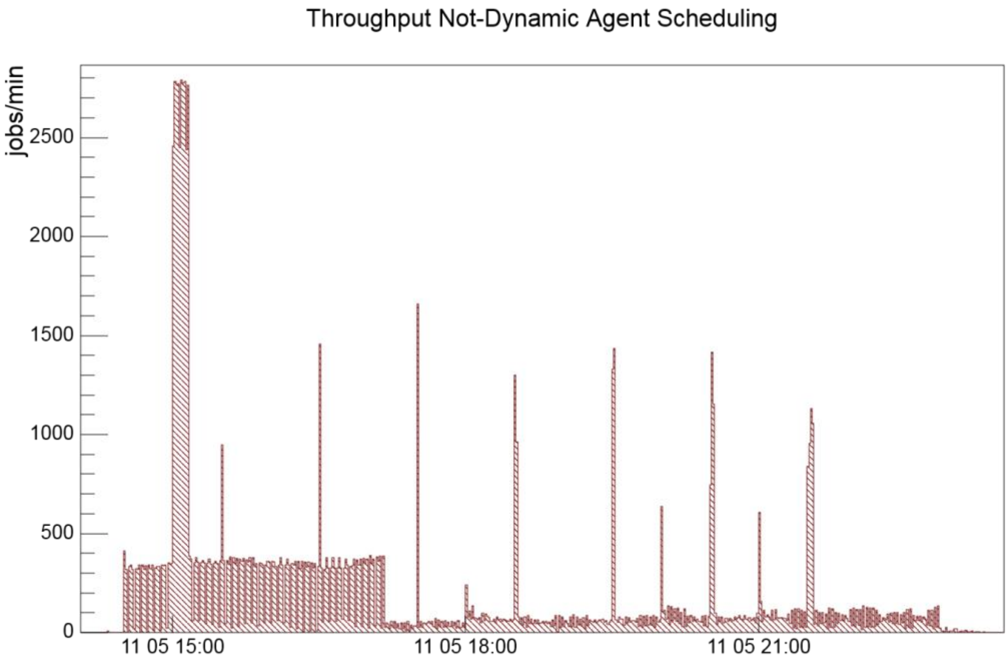

Figure 11. Fault tolerant agent schedule throughput in daily workload (10^5 jobs)

The scheduling throughputs represented in this section didn't suffer any queuing phenomenon (incoming and outgoing throughput are equivalent).

In fact, the throughput analysis confirms the performance and scalability levels assured in previous releases. From workload scheduler user perspective that means, for instance, no substantial delay in the scheduling a job when the same jobs 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***).

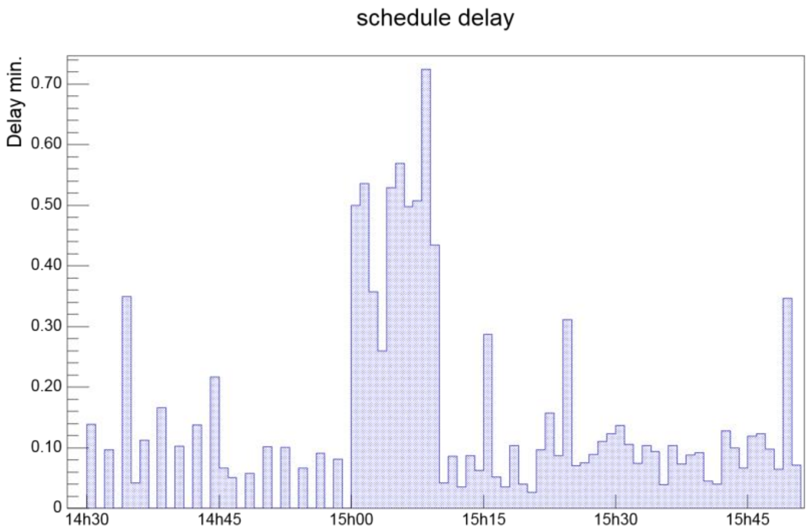

Figure 12. Average jobs schedule delay over time

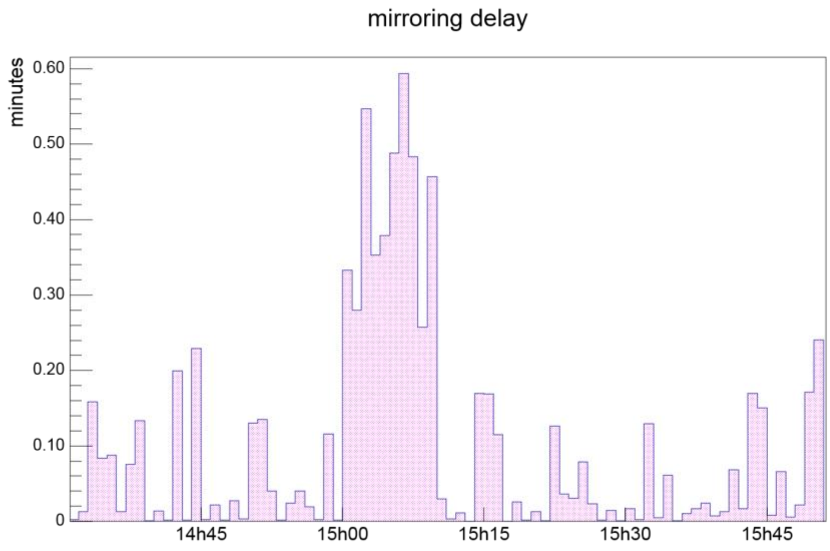

Figure 13. Jobs plan status update delay over time

Above results evidence the promptness of workload scheduler also in case of intensive workload (2600 jobs/min both for dynamic and fault tolerant agents) with dynamic schedule delay 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 to 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 resources utilization trend with respect the outcoming throughputs at dynamic agent scheduling, including two scenarios, one with exclusively dynamic agent scheduling and another one with both dynamic agents and fault tolerant agent (in the latter case the throughput must be considered double). As already revealed in previous performance reports, most of computation on master domain server node is related to dynamic job scheduling. The rest of computation is related to plan updates and, in this case, also fault tolerant agent job result notifications weight in resources utilization.

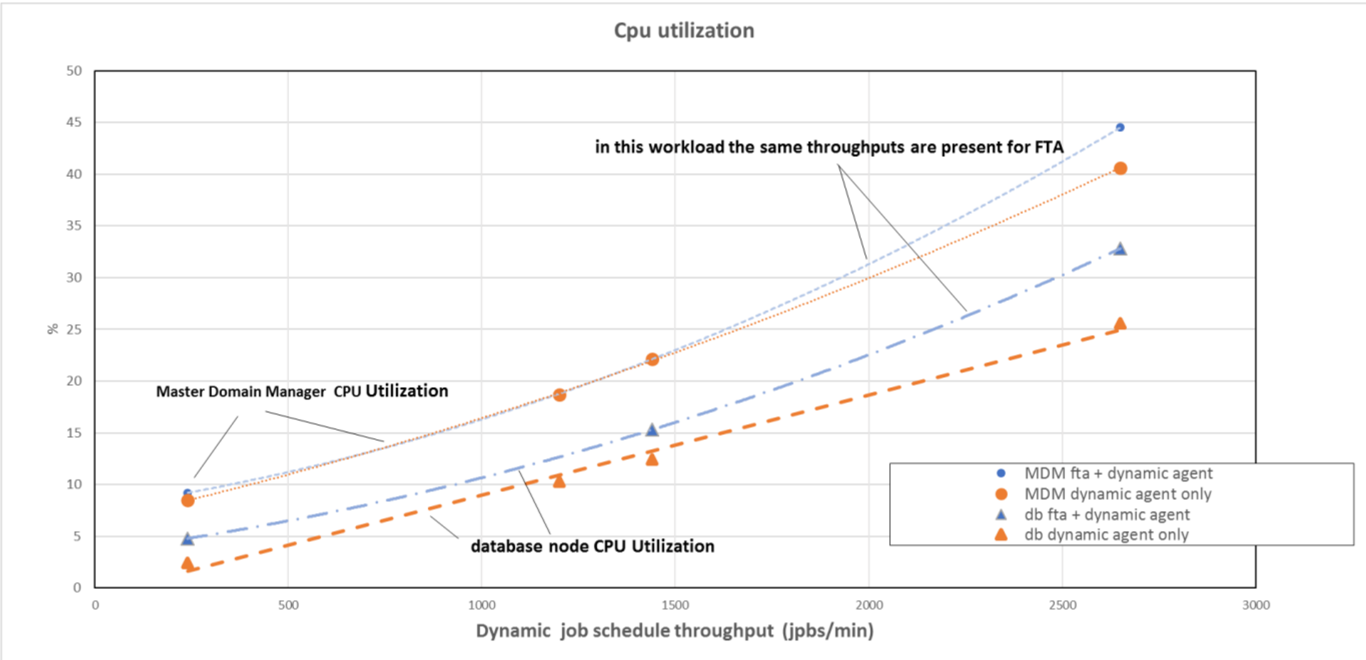

Figure 14. CPU utilization at master domain manager and database nodes sides vs different workload

3.4.2. Plan View

The Plan view has been redesigned to enhance the user experience (UX). The new design helps the user to accomplish the tasks easily and efficiently. Simple shapes to easily identify objects have been used, new icons to improve the interaction and quickly identify actions have been created, new colors and background to better visualize the objects have been applied.

The new graphical views (including the Plan View) has been implemented within a new client base framework. Most of the previous master workload has been moved on the client browser. That includes objects relationship computation and graphical rendering. This important architectural change allows to increase concurrency for Workload Scheduler operators accessing new graphical views.

Several job streams type have been tested as reported in the following table:

Workload	Number of Job Streams	Number of Job Streams dependencies
JS1	954	2895

Table 3. Job Stream filtered value for the Plan View scenario

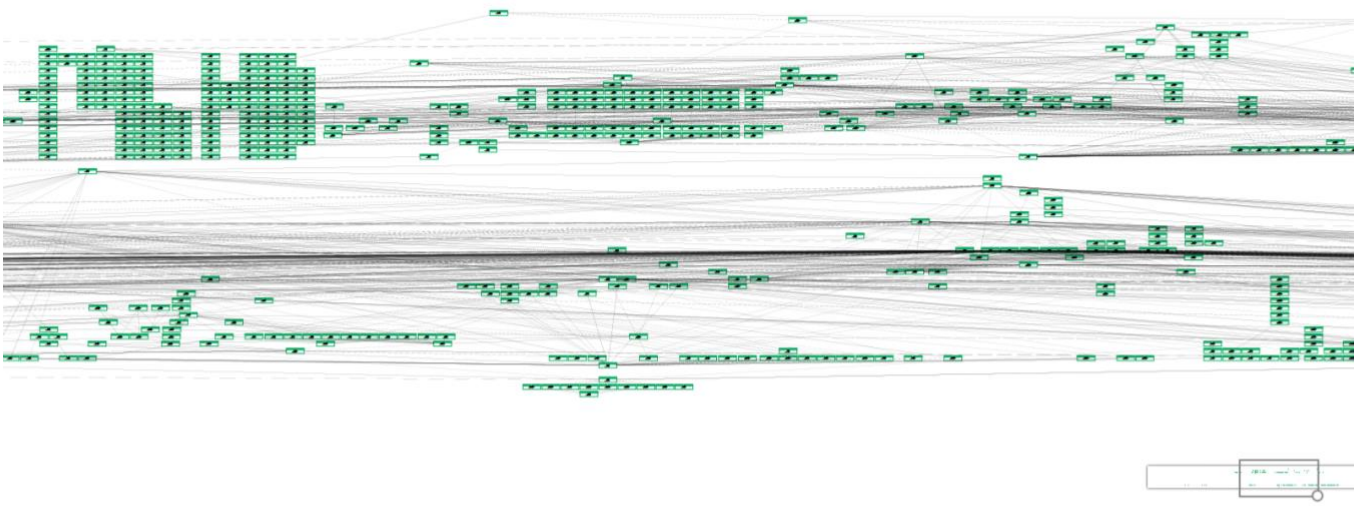

Figure 15. Job Stream network for the plan view scenario

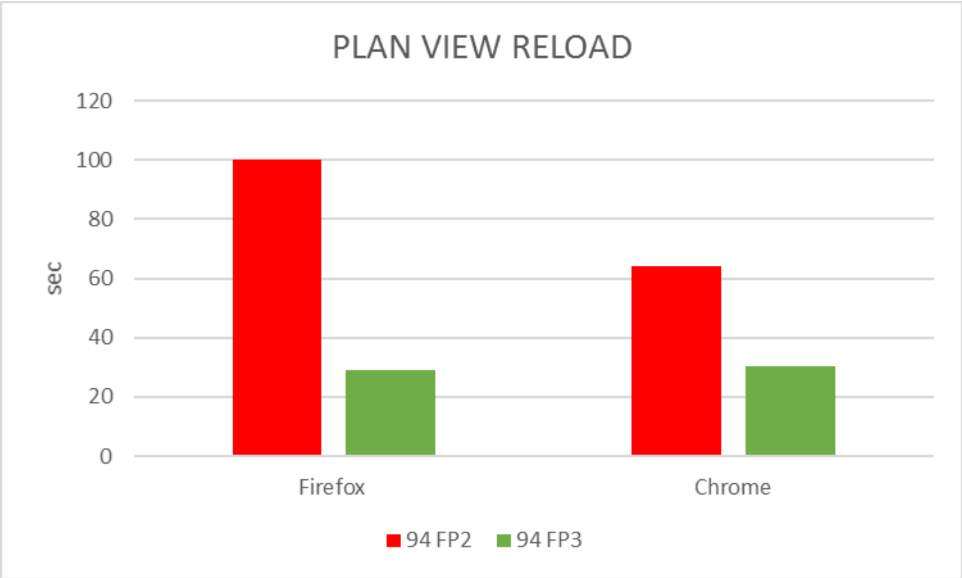

Figure 16. Comparisons of rendering time for graphical view reload

Given several jobs included in a job stream the rendering time appear to be almost linearly dependent from the total number of dependencies. This is due to the layout computational time.

Test have been performed on an Intel Core i7-4710MQ CPU @ 2.50 GHz ---- 8 logical cores host with the following browser levels:

- Firefox ESR 45.6
- Chrome 55.0

Results show how performance are strictly related to the specific browser: Chrome has better performances for this scenario and in general for all the end user scenarios for which the new graphical views are involved.

Note that the memory consumption for the browser process is also not negligible (around 700 MB).

Figure *15* give a rough idea of layout complexity of the job streams used in this benchmarks. It could be argued the feasibility of handling such objects in a graphical framework, anyway, tests where intended to stress the capability.

3.4.3. Job Stream View

In the Workload Scheduler 9.4.0.3 release, the job stream view scenario in the dynamic workload console has been optimized to avoid database resource consumption and high page response time in case of users concurrency. To validate the expected performance improvements, a test with 60 concurrent users (using Rational Performance Tester) has been done.

All these 60 concurrent users performed the same scenario (perform a monitoring job stream query to search for a specific job stream to retrieve the job log and to show the Job Stream graphical view) for 2 hours long, acting on different job steams:

- 30 users opened the Job Stream view for a Job Stream with 1000 jobs and 1000 dependencies and kept the Job Stream View opened for 5 minutes, with auto-refresh rate set to 5 seconds. In this case, the Job Stream live update call didn't return any change
- 30 users opened the Job Stream view for a Job Stream with 200 jobs and 235 dependencies and kept the Job Stream View opened for 5 minutes, with auto-refresh rate set to 5 seconds. In this case, the Job Stream live update call returned always some changes

The improvements vs the previous fix packs are meaningful:

- Dynamic Workload Console pages average response time comparison
  the response times for all the pages exercised during the scenario have been significantly improved. In particular the average response time of the live update call for the job Stream with 200 jobs and 235 dependencies has been reduced of 93%
- Average CPU usage on the database server machine
  the amount of CPU used in average on the database server has been reduced of 99%

3.4.4. High network latency test

Network latency has significant impact in a high loaded workload scheduler environment. This section has the objective to give a row quantitative estimation of performance impact due to network latency.
It could be easily understood that adding a delay while accessing the TCP layer means to increase the serving time at each internal queue that is related to network. This fact could be negligible in case of few transaction per unit of time but can have a disruptive effect in a context of workload like the ones described in this document.

The Linux kernel capability *netem* has been used to simulate large area network behavior and to add latency between nodes in the topology.
Using the command

tc qdisc add dev "net interface" root netem delay "value"ms "variance"ms

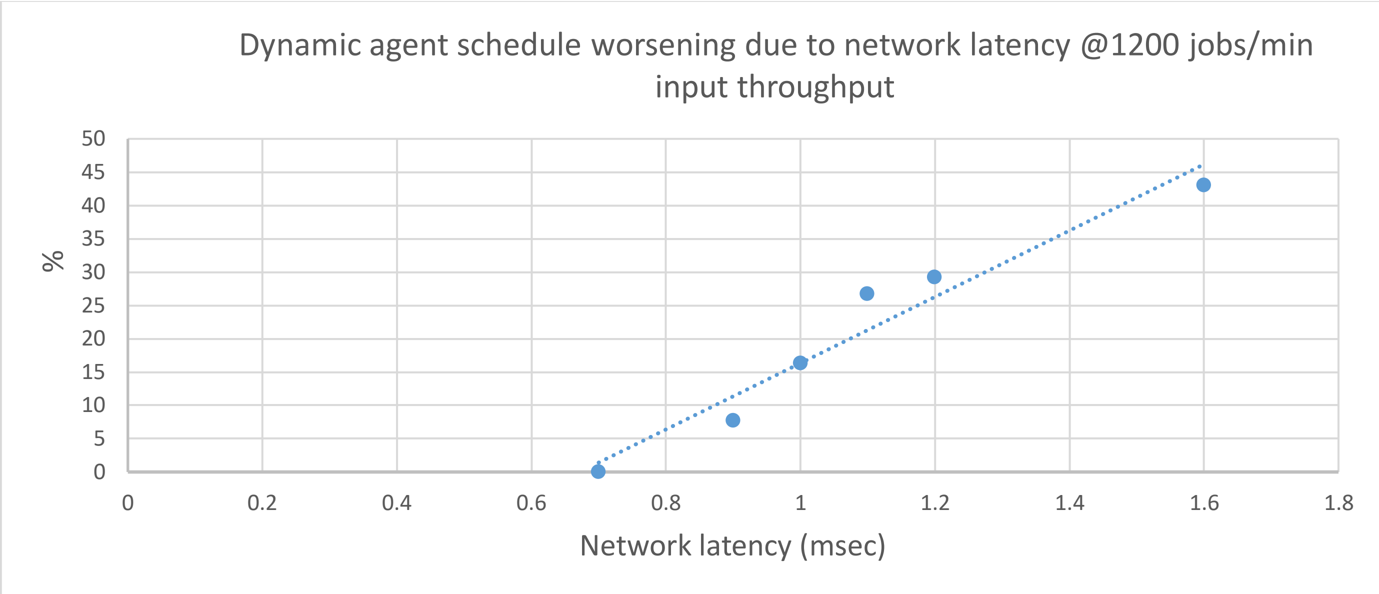

Figure 17. Network latency impacts on Dynamic Domain Manager throughput capabilities

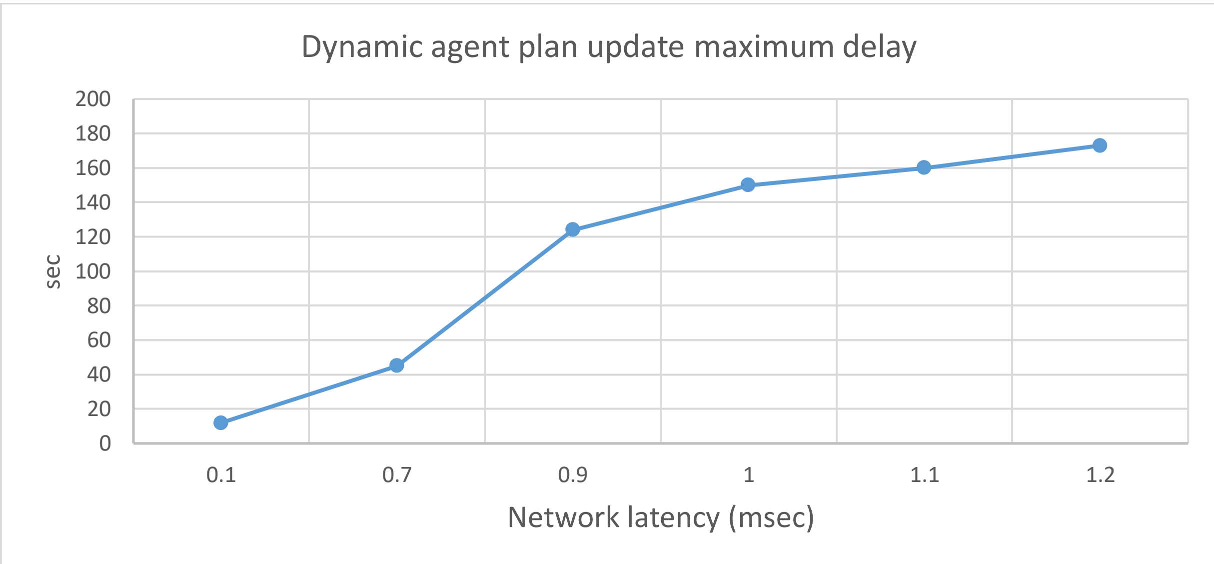

Figure 18. Network latency impacts on jobs plan status update

Network latency impact is directly proportional to the workload (in this case the baseline was 1200 jobs/min) and inversely to the system capacity and for this reason it is mandatory to correctly take it into consideration during workload scheduler deployment plan.

From the 2 graphs above, it is clear that, in our test environment and running the workload of 1200 jobs/min for at least 5 hours, we started to have meaningful impacts of Workload Scheduler performance capabilities starting from a network delay of 0.7 msec.

4. Best Practices

4.1. Scheduling

Workload Scheduler software offers 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 plan is running, using, for example, event rules, conman, start condition, and file dependencies. Even if the latter give high level of versatility to accomplish different business scenario, there are some recommendations that must be considered before planning to adopt them to orchestrate completely the schedule in case of 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 intercept, for example, job status change or file creation events. In all these cases the events have been sent to the event processor queue (cache.dat). In the case of 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 the cases the final submission throughput strictly depends on event processor throughput capability.

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 could not scale horizontally.

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

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

Table 4. Event processor capacity in terms of max 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 19** 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 20**). The focus of this section was for event processor throughput stressed with file creation rule. It has been evaluated that the type of rule is not impacting the throughput. There is a specific scenario, status change event rules, whose throughputs are constrained by the total number of event rules (see *[IBM Workload Scheduler 9.3.0.1 Capacity Planning Guide](#)*) causing enqueuing on the *monbox*.

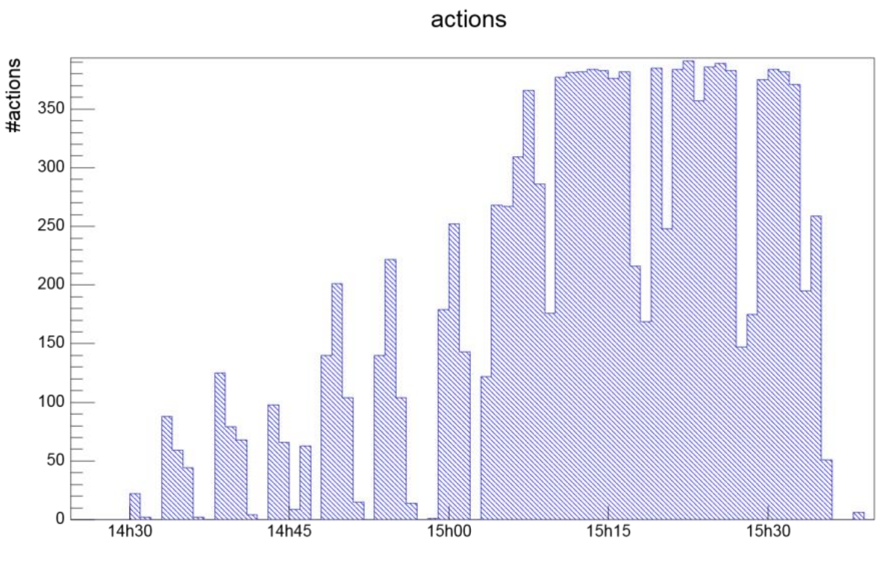

Figure 19. Number of action triggered by event processor

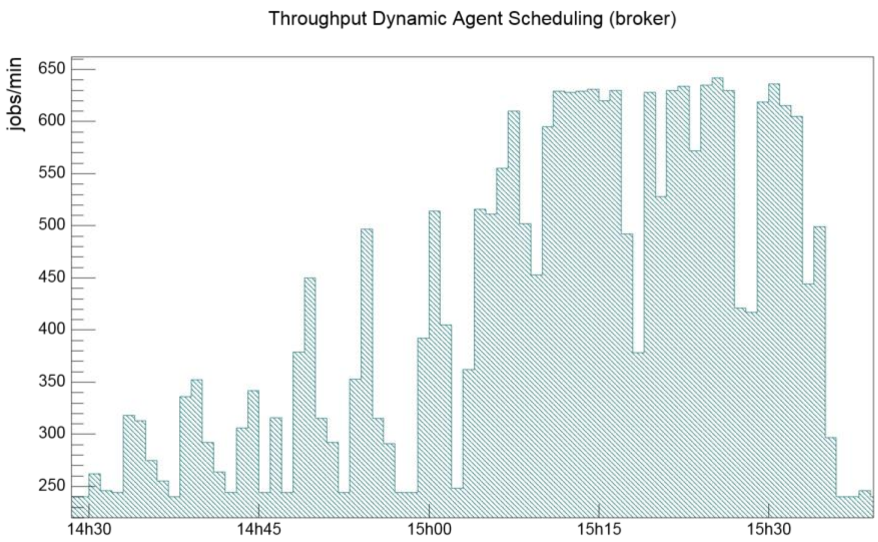

Figure 20. Dynamic Domain Manger jobs submission throughput in the file creation event rule scenario

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

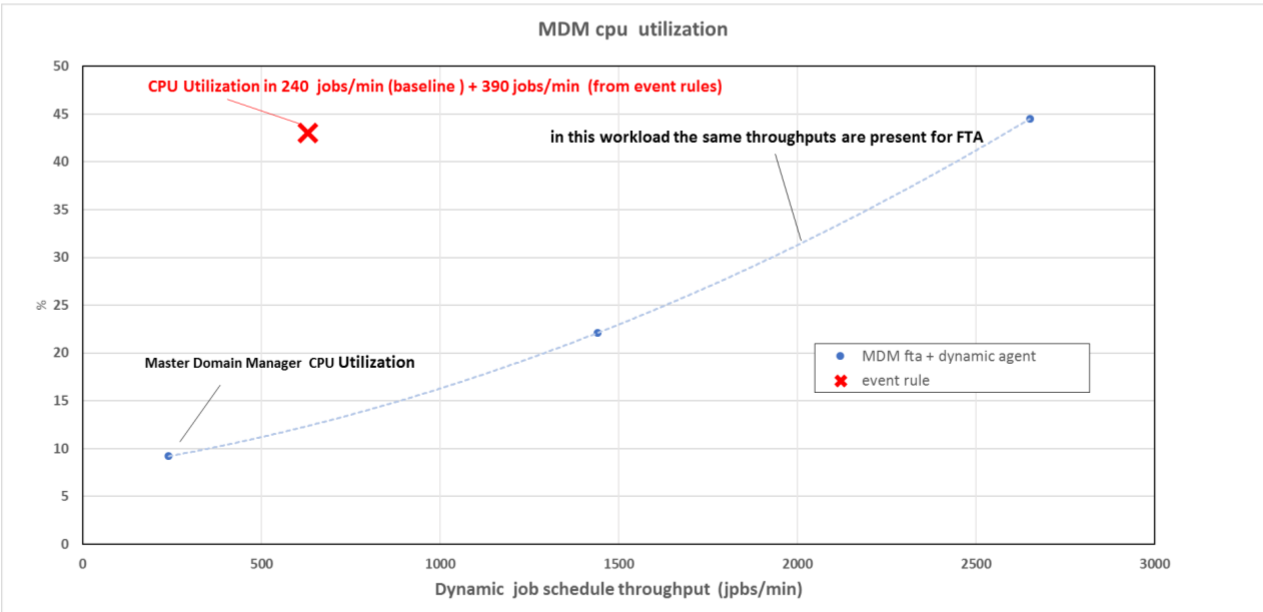

Figure 21. MDM CPU utilization comparison with and without event rules processing

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 dependencies release 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 section 3.4.1 already includes a component of this type acted against fault tolerant agents. This feature has different impacts on the performance if used with dynamic agent. In case of dynamic agent all the mechanism is driven by the dynamic domain manager that has in charge the continuous check of file existence status. The polling period is driven by the localopts property present on the Dynamic Domain Manager:

bm check file = 300 (5 minutes is the default).

It defines the frequency 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

In the test environment (with around latency 0.1 ms) the file check throughput has been evaluated to be around 44 seconds to check 1000 files.

It is suggested to keep the ratio (number of file dependencies)/(bm check file) less than 0.7.

This scenario has been applied with 2000 jobs scheduled with 2000 different file dependencies. File dependency release has been triggered in bunches of 400, 800 and 800 files when the system was charged with 1200 jobs/min job submission.

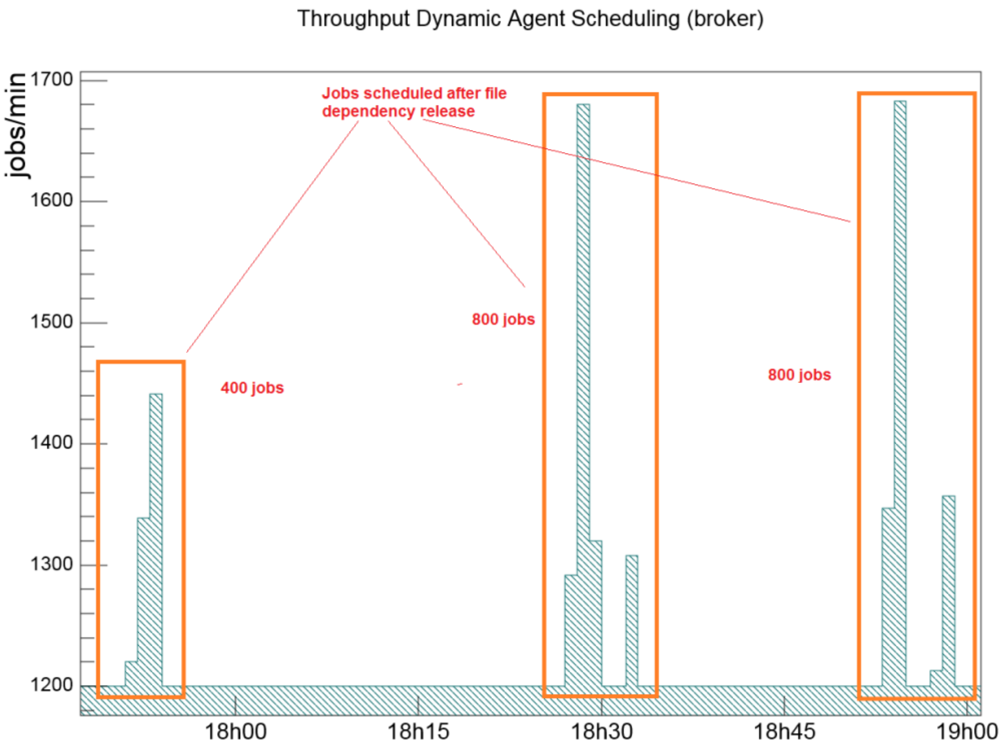

Figure 22. File dependency releases in a 1200 jobs/min workload as baseline

4.1.3. Scheduling using start condition

It is also possible to schedule job using file detection as trigger handled with job mechanism. This is known as Start Condition feature: a job called is kept on running until a file matches.

The capacity of this job stream submission is strictly related to global schedule capacity. The advantages consist of leverage of job control and monitoring.

4.1.4. Scheduling using conman sbs

The "*conman sbs*" (or equivalent RESTful calls) command allows to add jobstream to the plan on the flight. If the added jobstreams network was significantly complex both in terms of dependencies and cardinality it could cause a general delay in plan update mechanism. In this scenario, due to scheduling coherence, all the initial updates pass through the main thread queue (mirrobox.msg) missing the benefit of multithreading. It is extremely difficult to identify the complexity of network that would cause this kind of queueing, anyway the order of magnitude is of several hundreds of jobs in the job streams and internal and/or external dependencies.

4.2. Dynamic domain manager table cleanup policy

While increasing the workload in terms of number of dynamic jobs executed per day, the dynamic domain manager historical tables increase accordingly. Data persistency in this table allows to perform jobs log retrieve also for archived plans. The following parameters, in the JobDispatcherConfig.properties, define the cleanup policy:

- SuccessfulJobsMaxAge
- UnsuccessfulJobsMaxAge
- MoveHistoryDataFrequencyInMins

By default, the cleanup thread starts after “MoveHistoryDataFrequencyInMins” time laps since last occurrence completion or application server boot and remove jobs from table accordingly with their status and age. If job table is large (magnitude 10^6 rows) and the number of records to delete high (magnitude 10^5), this activity impact Workload Schedule performances and throughputs capabilities, as shown by the example below.

During 1200 jobs/min constant workload, the dynamic jobs cleaning started to delete almost 7x10^5 rows of jobs that whose 10 days aged 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:

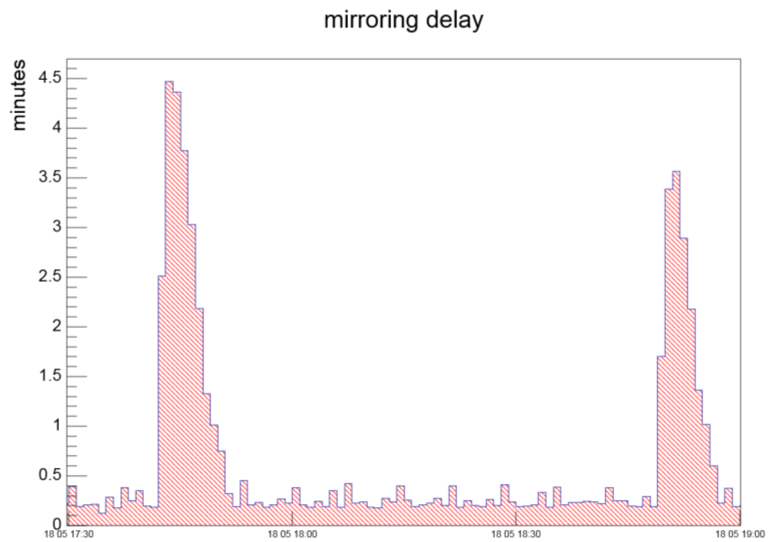

Figure 23. Plan update delay while dynamic jobs table is being cleaned up

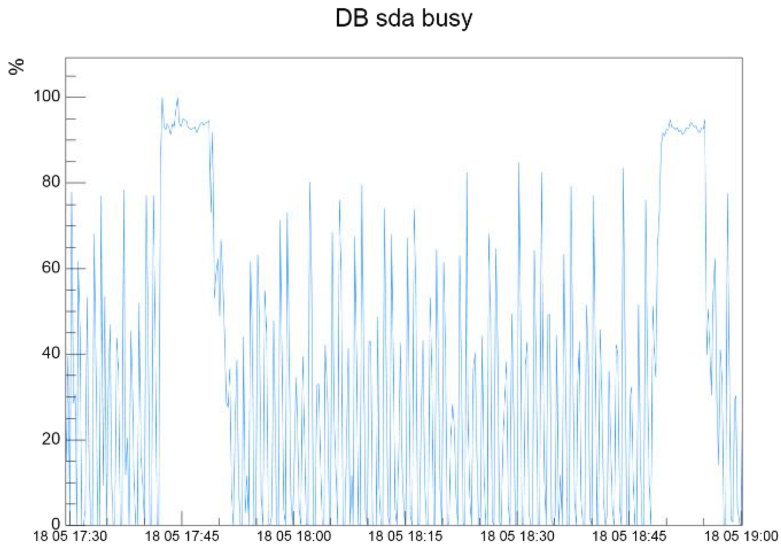

Figure 24. Database Server disk busy while dynamic jobs table is being cleaned up

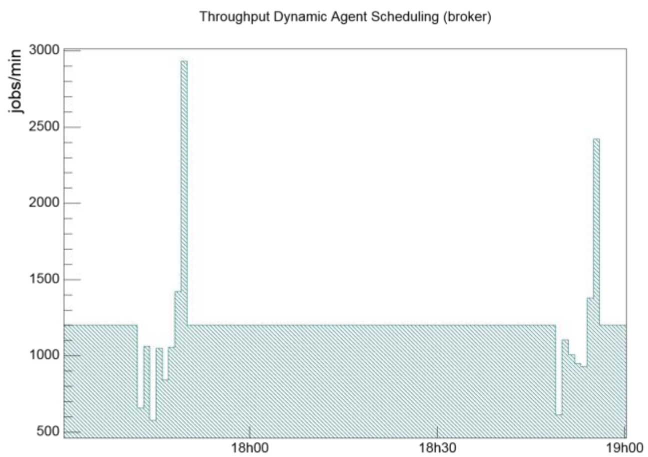

Figure 25. Impact Dynamic Domain Manager throughput capabilities while dynamic jobs table is being cleaned up

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:

<installation path>/TWA/TDWB/bin/movehistorydata.sh -successfulJobsMaxAge 240

The script could be executed in a job scheduled at the appropriate time windows. In this case the following configuration should be applied in the 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 26. 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 have been executed on virtual CPUs assigned exclusively to VMs (reserved resources). While planning the correct CPU sizing, the information provided in ***Table 8*** could be a reference point to start. It has been demonstrated 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.

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 has been underlined in previous performance reports in relation with the overall product performance.

The numbers presented in ***Figure 27*** could be used as 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

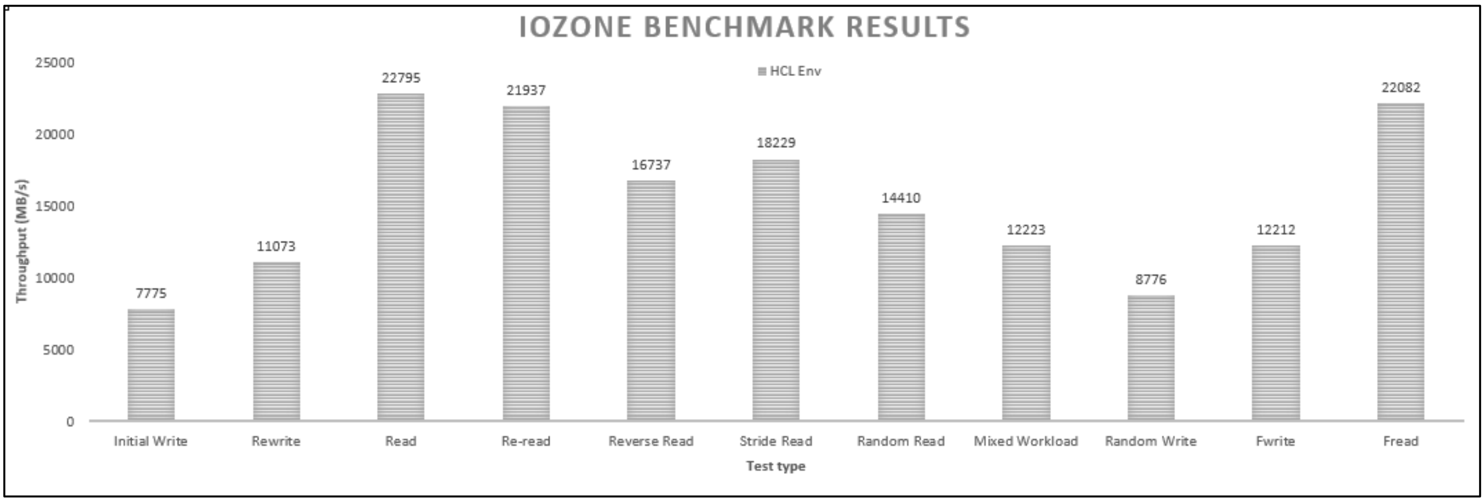

Figure 27. IOzone benchmark output run with “-R -l 5 -u 5 -r 4k -s 100m –F file1 ...file5” options

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 – 50	50 -100	100 - 150
DWC heap size	2 GB	4 GB	6 GB

Table 5. Dynamic Workload Console WebSphere Application Server 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 6. Engine WebSphere Application Server 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

On Linux x86 platform a specific analysis of JDBC data source configuration has been performed due to potential native limitation in socket activity within J2EE context. To achieve throughputs comparable with AIX, the JDBC data source configuration has been reworked. In the case of high schedule volume (for example greater than 1200 scheduled jobs/min) the following setting is strongly suggested:

- Validate existing pooled connections = false

This setting causes the Application Server to not perform a dummy query to the database to validate goodness of existing JDBC connections.

Having this setting equal true is used to catch the *StaleConnectionException* that could arise getting an held connection from the pool. StaleConnectionException extends SQL Exception that the application already catches. Handling this exception could make the application able to do recovery action on the pool (in Workload Scheduler the application server is in charge of this activity).

To limit the probability of encountering *StaleConnectionExceptions* when the validation of existing pooled connections is set to false, the following additional tunings are suggested:

- Data source MIN connections set to 0
- Data source Unused Timeout no greater than 1/2 the value configured for the firewall timeout, if a firewall is present
- Reap Timer value less than the Unused Timeout
- Data source Purge Policy set to entire pool

Dynamic Domain Manager jobs submission throughput benefits of this setting, the drawback effect is related to the possibility to have an exception while attempting to connect to DB, in case of unexpected network issues. In any case the other suggested settings allow to recover the event by recreating all the pool.

5.4.2. Plan Update (mirroring)

The plan update feature, also known as mirroring, has been improved release by release by mean of parallelism (multithreading) and caching. The former is defaulted to 6 thread 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 = 8

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 = 40000

Since 9.4.0.1 version of Workload Scheduler, to accomplish the stress of scenarios with thousands of file dependencies status update, a new caching mechanism has been added:

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

These settings are defined in the <WS_INST_PATH>/WAS/TWSProfile/properties/TWSConfig.properties file.

5.4.3. Comprehensive configuration and tuning

	Parameter	Value	Comment
Dynamic Workload Cosole	Dynamic Workload Console configuration settings repository (see https://www.ibm.com/support/knowledgecenter/SSGSPN_9.4.0/com.ibm.tivoli.itws.doc_9.4/distr/src_ad/awsaddwcanddb2.htm)	Use database as settings repository	It is strongly recommended to adopt this configuration to allow acceptable UI performance
	WebSphere Application Server WC Thread Pool Size	300	Should be adjusted with number of concurrent users accordingly
	WebSphere Application Server JVM max heap = min heap	Required: 4096 for [50, 100] users per node Suggested: 6144 for [100, 150] users per node	
	WebSphere Application Server JVM options	-Djava.awt.headless=true -Xdisableexplicitgc -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 JDBC max	300	

	Connections			
Master Domain Manager	WebSphere Application Server WC Thread Pool Size		300	
	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 Appllication Server 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	**For Oracle it must be changed after installation!**
			Connection Timeout = 180	
			Max Connections = 400	
			Min Connections = 0	
			Purge Policy = EntirePool	
			Reap Time = 180	
			Test Connection = false	
			Unused Timeout = 1800	
			Statement Cache Size= 400	
DB (db2)	LOGPRIMARY		200	MB total transaction log space
	LOGFILSIZ		3000	
	KEEPFENCED		NO	
	MAX_CONNECTIONS		AUTOMATIC	
	STMT_CONC		LITERALS	This setting optimizes query execution and reduces CPU usage
	SELF_TUNING_MEM		ON	
	APPL_MEMORY, APPLHEAPSZ, DATABASE_MEMORY, DBHEAP		AUTOMATIC	
	AUTO_RUNSTAT		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	JobDispatcherConfig.properties	Historical data management	MoveHistoryDataFrequencyInMins=720	
		Queue settings	Queue.actions.0 = cancel,cancelAllocation, cancelOrphanAllocation Queue.size.0 = 10 Queue.actions.1 = reallocateAllocation Queue.size.1 = 10 Queue.actions.2 = updateFailed Queue.size.2 = 10 Queue.actions.3 = completed Queue.size.3 = 30 Queue.actions.4 = execute Queue.size.4 = 30 Queue.actions.5 = submitted Queue.size.5 = 30 Queue.actions.6 = notification Queue.size.6 = 30	
	ResourceAdvisorConfig.properties		MaxAllocsPerTimeSlot = 1000	
			TimeSlotLength = 10	
			MaxAllocsInCache = 50000	
	TWSConfig.properties	Plan update configuratio n	com.ibm.tws.planner.monitor.subProcessors = 8	
			com.ibm.tws.planner.monitor.filecachesize =40000	
			com.ibm.tws.planner.monitor.cachesize = 40000	

Table 7. 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 as simple as possible:

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

With the above inputs, it is possible to forecast the resources needed to host the version 9.4.0.3 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.

<table>
<tr><th></th><th>NODE</th><th>Number of virtual cores</th><th>Disk Throughput Read-Write (MB/sec)</th><th>Network Throughput Read-Write (MB/sec)</th><th>RAM Capacity (GB)</th></tr>
<tr><td colspan="6">10K jobs (50% FTA +50% DYN) per day (8 jobs/min), 20 concurrent users</td></tr>
<tr><td>1 Node</td><td>WS Engine, RDBMS, DWC</td><td>4</td><td>0.5-0.1</td><td>0.5-0.7</td><td>16</td></tr>
<tr><td colspan="6">250K jobs (50% FTA +50% DYN) per day (175 jobs/min), 50 concurrent users</td></tr>
<tr><td rowspan="2">2 Nodes</td><td>WS Engine, DWC</td><td>4</td><td>0-1</td><td>0.9-2</td><td>16</td></tr>
<tr><td>RDBMS</td><td>4</td><td>2.3-0.9</td><td>0.5-1.5</td><td>16</td></tr>
<tr><td colspan="6">500K jobs (50% FTA +50% DYN) per day (350 jobs/min), 100+ concurrent users</td></tr>
<tr><td rowspan="3">3 Nodes</td><td>WS-Engine</td><td>8</td><td>0-1.3</td><td>1.6-1.3</td><td>32</td></tr>
<tr><td>RDBMS</td><td>8</td><td>2.3-1.2</td><td>1-2.2</td><td>32</td></tr>
<tr><td>DWC</td><td>5</td><td>0-0.1</td><td>1.2-1</td><td>20</td></tr>
</table>

Table 8. Capacity planning samples

7. Notices

This information was developed for products and services offered in the U.S.A.

HCL may not offer the products, services, or features discussed in this document in other countries. Consult your local HCL representative for information on the products and services currently available in your area. Any reference to an HCL product, program, or service is not intended to state or imply that only that HCL product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any HCL intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-HCL product, program, or service.

HCL may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to HCL TECHNOLOGIES LIMITED email: products-info@hcl.com

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: HCL TECHNOLOGIES LIMITED PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. HCL may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-HCL Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this HCL product and use of those Web sites is at your own risk.

HCL may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact HCL TECHNOLOGIES LIMITED email: products-info@hcl.com

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by HCL under terms of the HCL License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-HCL products was obtained from the suppliers of those products, their published announcements or other publicly available sources. HCL has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-HCL products. Questions on the capabilities of non-HCL products should be addressed to the suppliers of those products.

All statements regarding HCL's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All HCL prices shown are HCL's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

7.1. Trademarks

HCL, and other HCL graphics, logos, and service names including "hcltech.com" are trademarks of HCL. Except as specifically permitted herein, these Trademarks may not be used without the prior written permission from HCL. All other trademarks not owned by HCL that appear on this website are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by HCL.

IBM is a trademark or registered trademark of International Business Machines Corporation in the United States, other countries, or both. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.