IBM® Software

IBM Workload Scheduler Version 9.3.0.1 Performance and Capacity Planning Guide

Document version 1.0

Pier Fortunato Bottan
Giorgio Corsetti

IBM Workload Automation Performance Team - IBM Rome Lab

This edition applies to version 9, release 3, fix pack 1 of IBM Workload Scheduler and to all subsequent releases and modifications until otherwise indicated in new editions.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

IBM Workload Scheduler, formerly known as IBM Tivoli® Workload Scheduler, is licensed on a managed environment. Workload Scheduler V9.3 includes all the functions by IBM Workload Scheduler for Applications. IBM Workload Automation V9.3 is the complete solution that includes both Workload Scheduler and IBM Tivoli Workload Scheduler for Applications. Workload Automation V9.3 is technically identical to Workload Scheduler and is licensed as per job.

The following enhancements for both Workload Scheduler and Workload Automation can:

- Help improve the day-to-day operational efficiency for the IT staff, with improved flexibility for modeling and monitoring enterprise workflows.
- Help empower schedulers to be able to predict future behavior and problems in support of decision making and optimization planning. Schedulers can:
  - Predict job durations and see historical plan information.
  - Perform impact analysis of maintenance and schedule related what-if actions in a GANTT view on the current plan.
- Integrate with other tools and business applications, and control overall lifecycle events while optimizing idle time of the business processes. New standard integration is available for:
  - Oracle E-Business Suite
  - Oracle PeopleSoft
  - Salesforce.com
  - IBM Sterling Connect:Direct® file transfer
  - IBM Netezza® Performance Server
  - IBM WebSphere® MQ
  - RESTful Web Services
  - Apache Hadoop connectors and IBM InfoSphere® BigInsights™ for Apache Hadoop
  - Workload converters for Crontab schedules and Microsoft™ Windows™ Task Scheduler

- Provide the IT staff with customizable dashboard capabilities, allowing the user to build a personal entry page to help monitor the key parameters of the scheduling environment.
- Help increase SAP automation with a new adaptor for SAP Business Objects reports and with the support and certification of the Solution Manager interface (SMSE) to enable Workload Scheduler as an external scheduler for SAP. With the SMSE integration, SAP users can define and monitor their SAP workloads without changing context from SAP, by exploiting Workload Scheduler's robust and scalable scheduling engine.

IT Administrators can:

- Deploy Workload Automation on Informix® or Microsoft SQL database, in addition to IBM DB2® and Oracle.
- Apply maintenance levels or fix packs on the agent's networks with the centralized agent updates.
- Install agents by using domain users.
- Run jobs on SAP, PeopleSoft, Oracle for eBusiness and Informatica without the need to install and maintain a separate software component. At the infrastructure level, all the functions provided by Tivoli Workload Scheduler for Applications are now embedded into Workload Scheduler.
- Easily deploy new or updated job type plug-ins, installing them just on the main servers without the need to update agents.

IT Administrators can now automate the maintenance of the agents' network with either:

- The new centralized agent updates capability that allows IT Administrators to manage software updates on Workload Scheduler agents (maintenance levels and fix packs) directly from the Dynamic Workload Console. It scales up to 20 agents simultaneously and supports Microsoft Windows and UNIX™ Platform agents.
- IBM Endpoint Manager to manage agents' deployment and maintenance from the Endpoint Manager console. It offers greater scalability (1,000 plus agents), leverages multiple depots for optimized network

bandwidth, and allows IT Administrators to manage software updates on Workload Scheduler agents also through APIs and command line. With Workload Automation V9.3, users are entitled to use Endpoint Manager V9.2 to manage Workload Automation agents' patching activities (limited to only Workload Automation agents).

Schedulers can:

- Gain flexibility in the definition of complex automated workflows with every option available on jobstreams to enable automatic rerun on a defined period of time as well as nonoperational jobs capability that will allow a scheduler to temporarily disable some jobs in a flow without the need to rework the flow.
- Improve job return code mapping with customized conditions for improved flexibility in the analysis of the workload automation job results.
- Take advantage of new reports about job history and statistics, using the Tivoli Common Reporting component. This solution, built on IBM Cognos® technology, is easily customizable to exploit database history information and refine reports in terms of data filters, and look and feel.

2. Scope

2.1. Executive summary

The objective of the tests described in this document, is to report the IBM Workload Scheduler V9.3.0.1 scalability and performance improvements with respect to V9.1:

- Database plan status update throughput (mirroring);
  - ✓ **An improvement factor of 17.5 times for plan status update rates that caused a tremendous reduction of conman and Dynamic Workload Console synch delay**
- Dynamic agent schedule throughput;
  - **An improvement factor of 6.5 times for dynamic scheduling capacity**
- User Interface scalability
  - ✓ **An improvement factor of 1.4 times for Dynamic Workload Console concurrent users for a configuration with 4 Dynamic Workload Console nodes in a Dashboard Application Services Hub (DASH) High Availability (HA) cluster**
- Event-Driven Workload Automation rules (IWS objects) deployed on dynamic domain manager
  - ✓ **Performance impact analysis and tremendous improvements with respect to the V9.1 product release**
- Performance data on new features
  - ✓ **Conditional dependencies performance impact analysis**
- Product reliability verification while running a constant workload
  - ✓ **One week test with a workload of 450,000 jobs per day**
- Specific analysis of storage impact on performance
  - ✓ **Quantitative analysis of storage throughput impact**

Most of the above items have been already analyzed and documented in the “IBM Tivoli Workload Scheduler V9.1 Capacity Planning Guide”. The objective of this document is to report any improvements with respect to what was documented in the V9.1 document.

The ultimate target is to deliver capacity planning instruments to forecast the hardware and software configuration required to support an IBM Workload Scheduler workload. In this document some guidelines will be provided by mean of examples of configuration.

It could be useful to explain the meaning of some typical performance terms that are used in this document.

Throughput could be simply thought of as a rate, commonly expressed by the number of objects that transit in a system per unit of time. In this context:

- **Input Throughput** is the number of jobs that are scheduled to be executed per unit of time (typically 1 minute) that could be defined in the daily plan or dynamically added; in case of users interface it is the number of page requested rate.
- **Output Throughput** is the number of actual scheduled jobs, job result updates or the number of pages displayed in the user interface per unit of time.

3. Capacity Test

3.1. Test Approach

To measure throughput improvements, the workload used for the V9.1 Capacity Plan test has been kept as a reference model for this test. References to “Medium” and “Large” refer to test conducted and documented in the IBM Tivoli Workload Scheduler Version 9.1 Capacity Planning Guide. See **Table *3*** for a brief summary.

In addition to the collection of throughput benchmark results, specific focus was dedicated to product reliability. In particular, adding a workload to the medium scenario to create a continuous product 24x7 stress. Driven by customer feedback, this addendum was a schedule against thousands of simulated dynamic agents. Specific effort in this work was dedicated to analyze the impact of a storage solution in the product’s performance and throughput. This was feasible for the availability of an equivalent second hardware and software configuration whose main difference was the storage architecture.

3.2. Test Benchmarks

3.2.1. User interface scenarios

A particular test scenario was chosen to provide backward comparison with the benchmark run in the previous release and to consider some new key features that have been added in the meantime from IWS 9.1 up to the 9.3.0.1 release. Four main areas were identified and among them a set of sub-scenarios were designed with a defined weight as follows:

Monitoring (45% of the users)

1. Performing a monitoring query by job to search for a specific job and eventually to retrieve the job log (20%).
2. Performing a monitoring query by job stream to search for a specific job stream (20%).
3. Workload Dashboard initialization and navigation through some portlets (available workstations, late jobs, and log messages) present in the dashboard (5%).

Graphical (20% of the users)

4. Performing a monitoring query by job stream to search for a specific job stream and to open the related job stream graphical view (10%).
5. Performing a monitoring query by job stream to search for a specific job stream and to open the related job stream impact view (10%).

Modeling (5% of the users)

6. Navigating through the "Workload Designer" application to create new jobs and job stream definitions (3%).
7. Navigating through the "Workload Designer" application to search for a job, edit it, and save it (2%).

Mobile (30% of the users)

8. Navigating through the Self-Service Dashboard for monitoring purposes (15%).

9. Navigating through the Self-Service Catalog to submit a service and monitor its completion status (15%).

An IBM Workload Scheduler master with around 65K jobs in plan was used. To keep constant the number of objects returned by monitoring queries, plan execution was kept blocked. Each sub-scenario consists of three steps:

- Log in
- Transaction (composed of a series of activities that start from the welcome page and finish with returning to the same page)
- Log out

Each user in the automation framework (Rational Performance Tester) logs in and completes three transactions before logging out and reentering again with different credentials. The delay between each transaction is controlled by the framework to have a frequency of:

20 transactions/hour per user

3.2.2. Scheduling scenarios for reliability test

The workload used for long run test was based on Medium scenario whose structure was defined as follows:

- 50% of jobs executed on fault-tolerant agents
- 50% of jobs executed on dynamic agents

All jobs were included in a job stream composed of 50 jobs and having the following structure: 25% of job streams with dependencies, and 10% of jobs having dependencies from external jobs (defined in different job streams). Event-driven workload automation and workload service assurance features were also used (Table 1). In addition to this, a constant workload of 1000 jobs scheduled every 4 minutes against 1000 simulated agents every 4 minutes have been applied all day long. A total of 420K jobs per day are executed continuously for 7 days.

Scenario Description		Schedule Time	(Jobs/Min)
50% FTA + 50 % Dynamic (8 agents)	**baseline**	**10:30 - 13:15**	**240**
	Peak (10 min)	**11:00 - 11:10**	**2660**
Workload service assurance: 24 complex pattern with multiple dependent JS (4) including 10 jobs with 4 critical ones		**10:30 - 13:15**	**6**
Event driven workload automation		**11:30 - 12:30**	**4 (generated events)**
Dynamic workload @1000 agents		**0:00 - 24:00**	**250**

Table 1. Schedule workload for reliability test

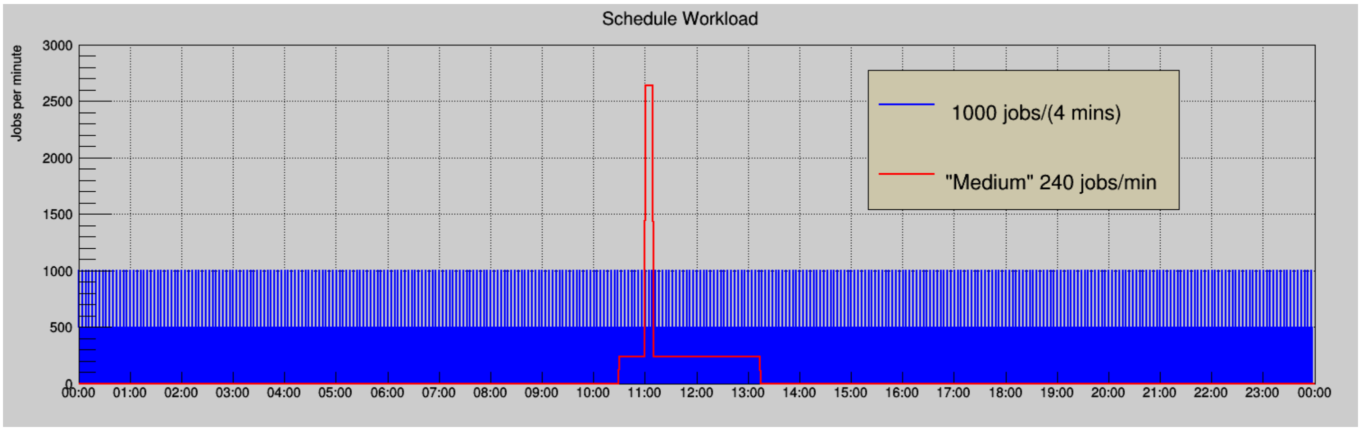


Figure 1. *Scheduling workload benchmark design. Baselines and peaks*.

3.2.3. Environment

The test environment was based on LPAR nodes hosted on a P7 IBM 8233-E8B (3GHz). All tests were performed in a 10 GB local area network. LPAR had dedicated cores whose numbers have been changed during benchmark executions.

The following table summarizes the software used and the version:

OS	AIX 7.1 TL 03
RDBMS	IBM DB2 v10.5.0.6
J2EE	IBM WebSphere® Application Server 8.5.5.4 with SDK 7.0.8.0
LDAP	IBM Directory Server 6.3
Jazz™ for Service Management	JazzSM 1.1.2.1 with DASH 3.1.2.1
IWS	9.3.0.1

Table 2. 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-01.ibm.com/support/knowledgecenter/SSGSPN_9.3.0/com.ibm.tivoli.itws.doc_9.3/distr/src_ad/ctip_config_ha_ovw.htm
was followed to set up a high availability configuration (also known here as cluster).

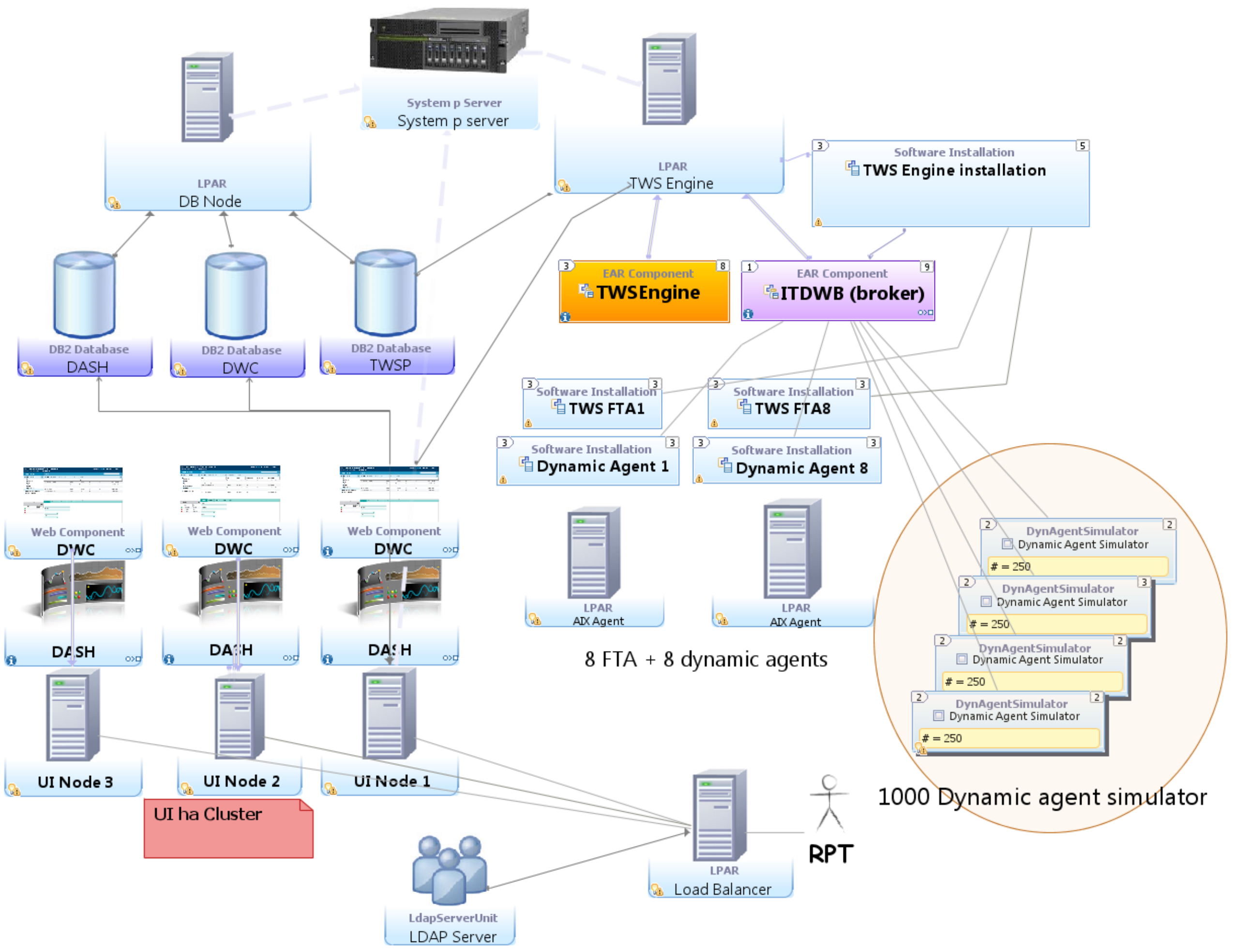


Figure 2. Overall deploy view of test environment

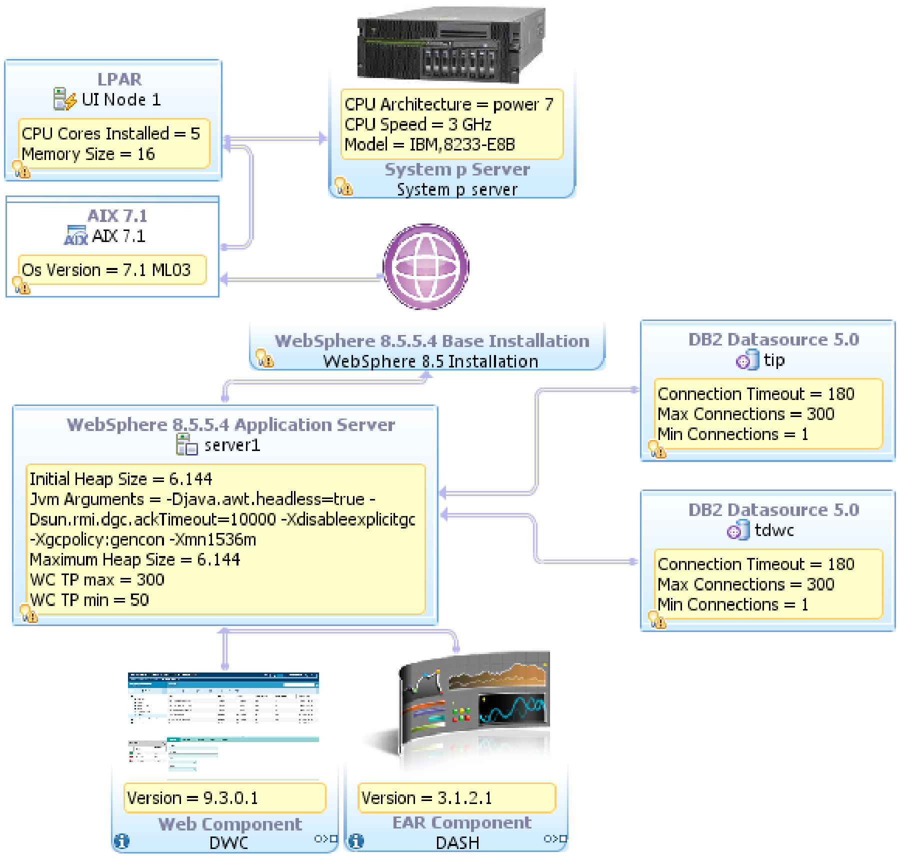
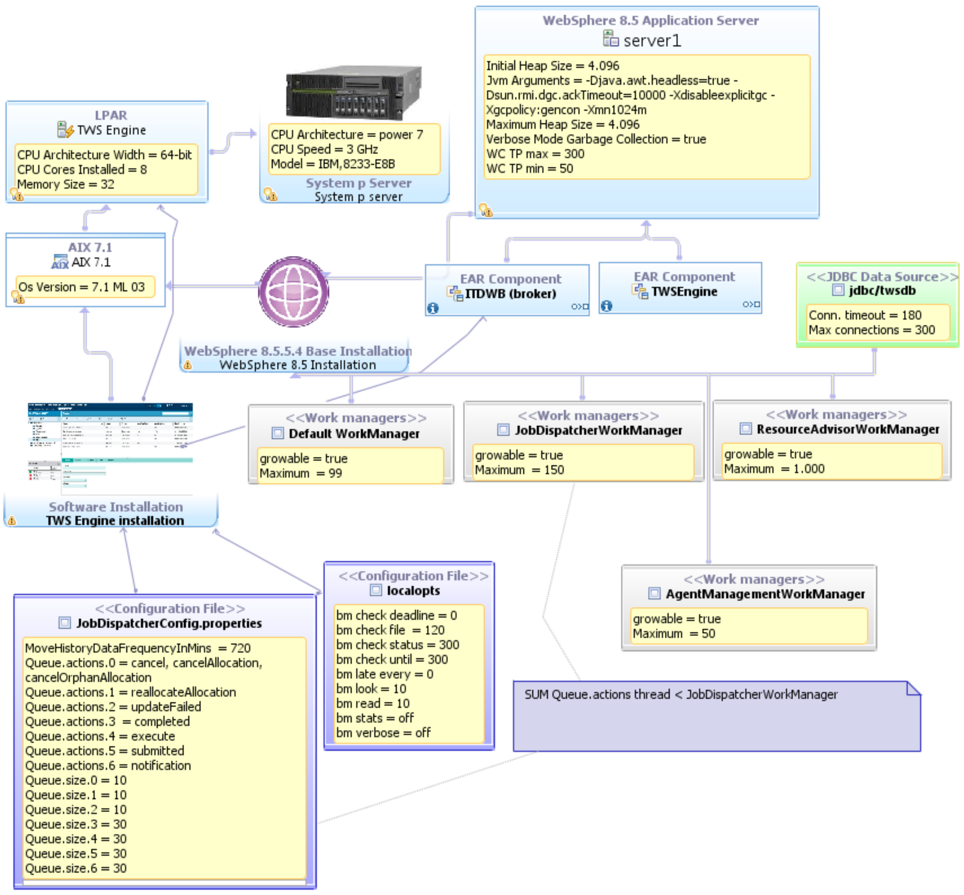


Figure 3. Dynamic Workload Console node configuration.



WebSphere 8.5 Application Server server1:
```
Initial Heap Size = 4.096
Jvm Arguments = -Djava.awt.headless=true -
Dsun.rmi.dgc.ackTimeout=10000 -Xdisableexplicitgc -
Xgcpolicy:gencon -Xmn1024m
Maximum Heap Size = 4.096
Verbose Mode Garbage Collection = true
WC TP max = 300
WC TP min = 50
```

LPAR TWS Engine:
```
CPU Architecture Width = 64-bit
CPU Cores Installed = 8
Memory Size = 32
```

System p Server:
```
CPU Architecture = power 7
CPU Speed = 3 GHz
Model = IBM,8233-E8B
```

AIX 7.1:
```
Os Version = 7.1 ML 03
```

jdbc/twsdb:
```
Conn. timeout = 180
Max connections = 300
```

Default WorkManager:
```
growable = true
Maximum = 99
```

JobDispatcherWorkManager:
```
growable = true
Maximum = 150
```

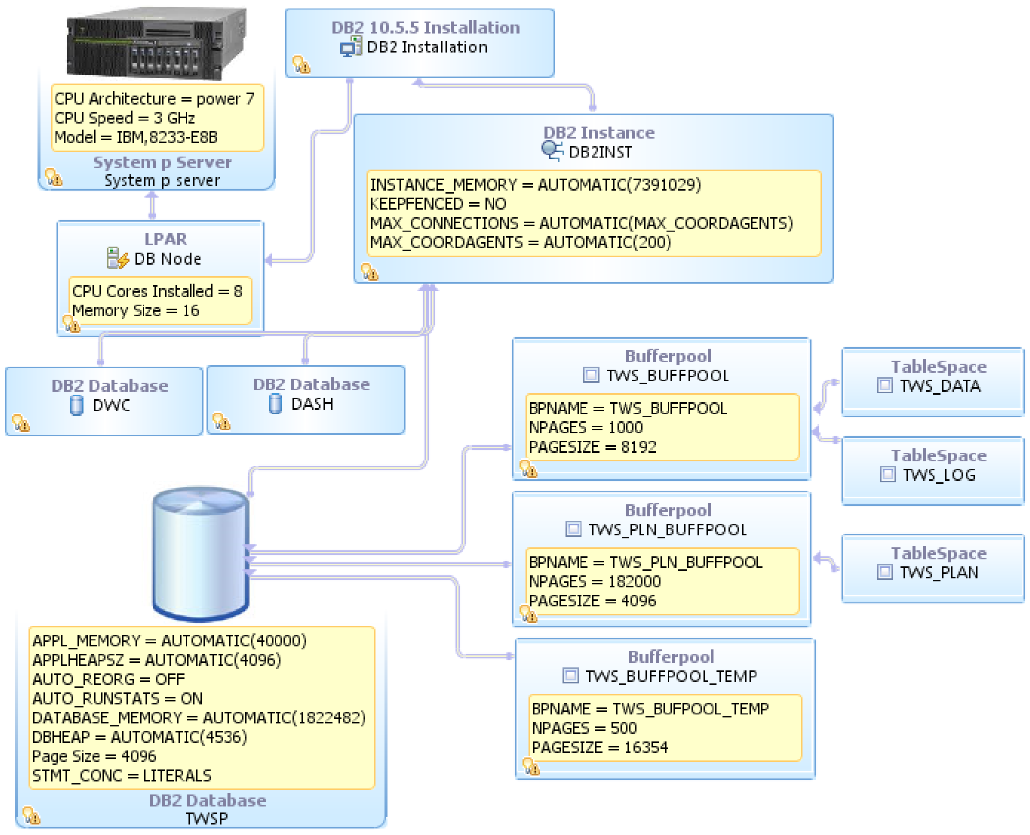
ResourceAdvisorWorkManager:
```
growable = true
Maximum = 1.000
```

AgentManagementWorkManager:
```
growable = true
Maximum = 50
```

localopts:
```
bm check deadline = 0
bm check file = 120
bm check status = 300
bm check until = 300
bm late every = 0
bm look = 10
bm read = 10
bm stats = off
bm verbose = off
```

JobDispatcherConfig.properties:
```
MoveHistoryDataFrequencyInMins = 720
Queue.actions.0 = cancel, cancelAllocation,
cancelOrphanAllocation
Queue.actions.1 = reallocateAllocation
Queue.actions.2 = updateFailed
Queue.actions.3 = completed
Queue.actions.4 = execute
Queue.actions.5 = submitted
Queue.actions.6 = notification
Queue.size.0 = 10
Queue.size.1 = 10
Queue.size.2 = 10
Queue.size.3 = 30
Queue.size.4 = 30
Queue.size.5 = 30
Queue.size.6 = 30
```

Figure 4. Engine node configuration



System p Server:
```
CPU Architecture = power 7
CPU Speed = 3 GHz
Model = IBM,8233-E8B
```

DB2 Instance DB2INST:
```
INSTANCE_MEMORY = AUTOMATIC(7391029)
KEEPFENCED = NO
MAX_CONNECTIONS = AUTOMATIC(MAX_COORDAGENTS)
MAX_COORDAGENTS = AUTOMATIC(200)
```

LPAR DB Node:
```
CPU Cores Installed = 8
Memory Size = 16
```

Bufferpool TWS_BUFFPOOL:
```
BPNAME = TWS_BUFFPOOL
NPAGES = 1000
PAGESIZE = 8192
```

Bufferpool TWS_PLN_BUFFPOOL:
```
BPNAME = TWS_PLN_BUFFPOOL
NPAGES = 182000
PAGESIZE = 4096
```

Bufferpool TWS_BUFFPOOL_TEMP:
```
BPNAME = TWS_BUFPOOL_TEMP
NPAGES = 500
PAGESIZE = 16354
```

DB2 Database TWSP:
```
APPL_MEMORY = AUTOMATIC(40000)
APPLHEAPSZ = AUTOMATIC(4096)
AUTO_REORG = OFF
AUTO_RUNSTATS = ON
DATABASE_MEMORY = AUTOMATIC(1822482)
DBHEAP = AUTOMATIC(4536)
Page Size = 4096
STMT_CONC = LITERALS
```

Figure 5. Database node configuration

3.3. Test tools

Rational Performance Tester (RPT) version 8.7.0.2 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 5.0 – Garbage Collection and Memory Visualizer. IOzone version 3.434 has been used to benchmark storage throughput.

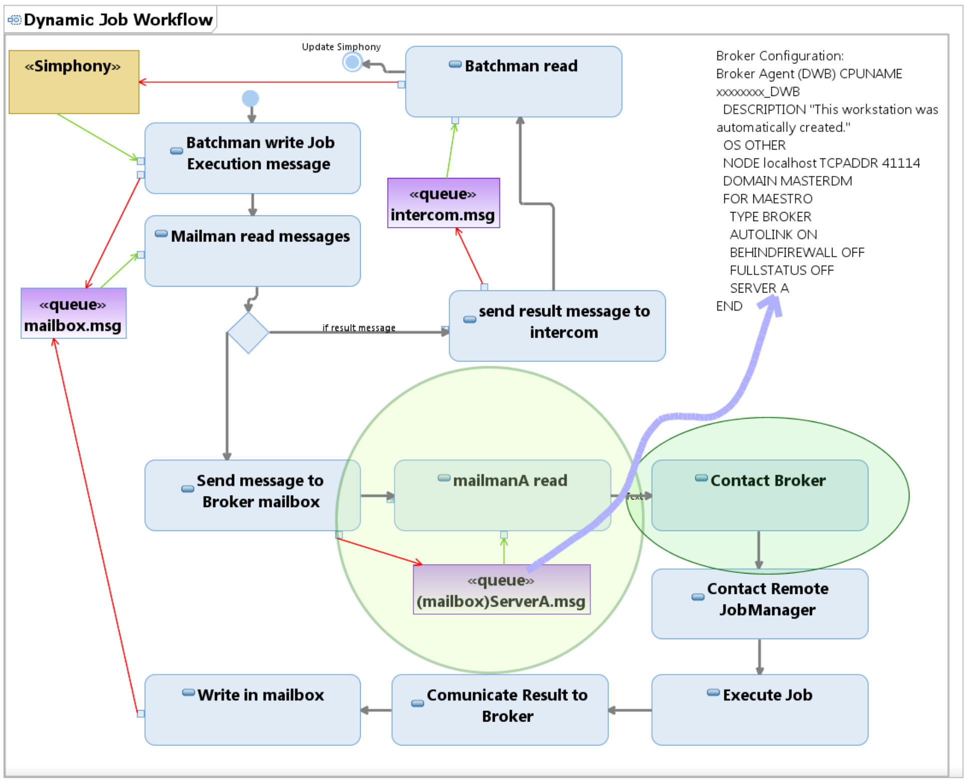
The Perfanalyst tool was used to control the middleware configuration and to analyse the DB2 snapshot.

4. Benchmark Results

4.1. Scheduling scenarios

In this section the main improvements of core scheduling capability of the version 9.3.0.1 product with respect to version 9.1 are exposed. In addition, not only the pure throughput benchmark results are reported but also the evaluation of a long run workload test to establish a touchstone on product reliability.

Differently from fault-tolerant agents, dynamic agents, managed by the dynamic domain master, job submission is handled centrally and causes additional processing at the Master node. The workflow of a dynamic job submission is described in Figure 6; in the version 9.1 of product, to improve performances an additional Mailman server was manually added to serve the Broker component. In this context, the same configuration has been maintained. The latest version not only includes improvements in queue serving (ServerA.msg in **Figure 6**) but also in the internal Broker algorithm.



```
Broker Configuration:
Broker Agent (DWB) CPUNAME
xxxxxxxx_DWB
  DESCRIPTION "This workstation was
automatically created."
  OS OTHER
  NODE localhost TCPADDR 41114
  DOMAIN MASTERDM
  FOR MAESTRO
    TYPE BROKER
    AUTOLINK ON
    BEHINDFIREWALL OFF
    FULLSTATUS OFF
    SERVER A
END
```

Figure 6. Dynamic agent job submission workflow. The dynamic domain manager (Broker) queue processing activity and Broker have been optimized to improve overall throughput. To separate this activity from the other `Mailman` server activities, a specific `Mailman` server process and queue has been defined using the dynamic agent broker configuration. In this example, the new server is called Server A

4.1.1. Long run test

The workload described in section 3.2.2, as planned, has been applied for 7 days without any server downtime handling almost 3 million scheduled jobs. Output throughputs were constant and no detectable leaks were found. Particular focus was given to the engine application server JVM heap during the test. ***Figure 7*** shows the similarity of garbage collection behavior with no increasing trend with the typical sawtooth shape of *gencon* policy.

Due to the high dynamic scheduling load (390K jobs per day) the default job archiving policy has been tuned as follows:

SuccessfulJobsMaxAge = 24

The above parameters in the JobDispatcherConfig.properties file, defines how long (in hours) the successful job results must persist in the Dynamic Workload Broker table before being archived. In this scope, the period is set to one day to avoid accumulating millions of entries causing possible transaction log issues whose total size is set to 1000 x 4K x 200 log archive files.

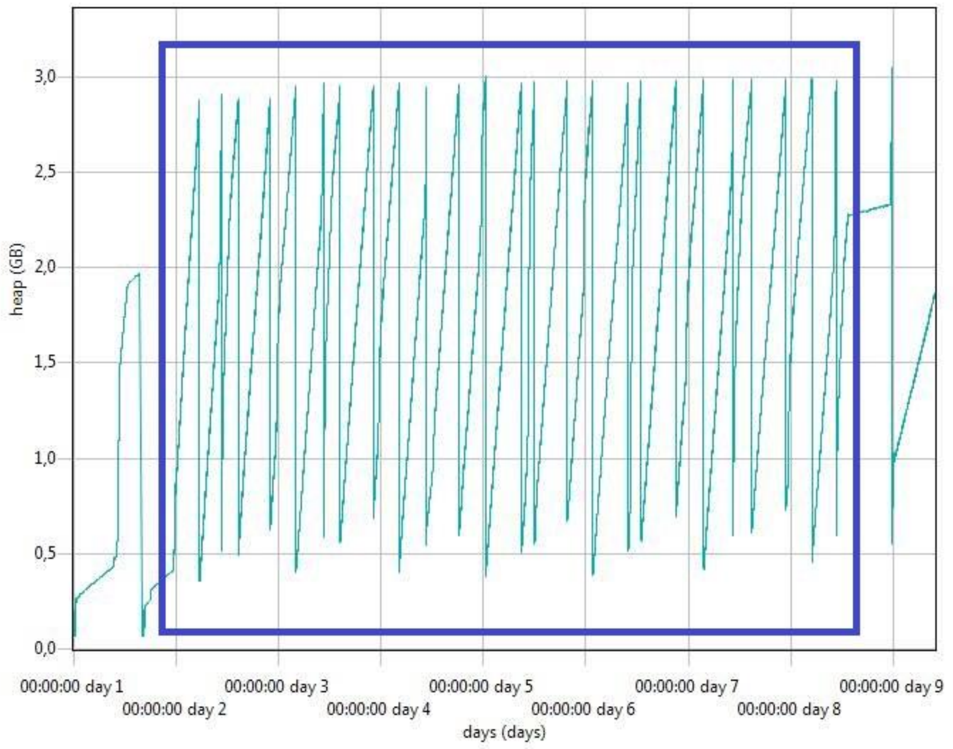


Figure 7. Used memory heap after collection in engine application server JVM during 7-day reliability test

4.1.2. Throughput improvements and scaling properties

Two different workloads were used as touchstones for the version 9.3.0.1 and 9.1 comparison test:

	"Medium" (jobs/min)	"Large" (jobs/min)
Baseline	240	480
Peak (10 min)	2660	5280

Table 3. "Medium" and "Large" refer to output throughput scenarios conducted and documented in the IBM Tivoli Workload Scheduler Version 9.1 Capacity Planning Guide

The output throughput of dynamic job submission reached up to 2.6K jobs submitted per minute (Figure 8). The effect of this tremendous improvement is that:

- Job submission delays are below 1 minute including during periods of peak activity.

Previous tests on the Version 9.1 product stated that this throughput was limited to almost 400 jobs/min, this implies a 6.5 magnitude increment.

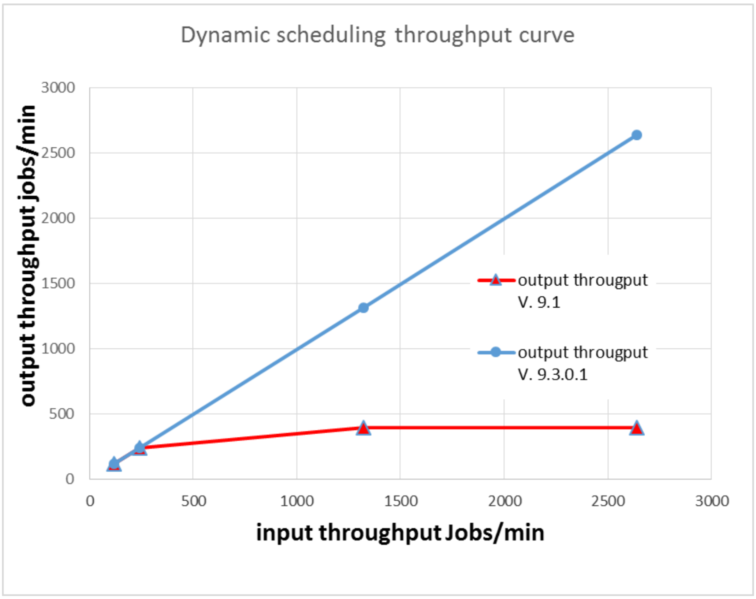


Figure 8. Incoming throughput, in terms of planned scheduled jobs on dynamic agents, vs output throughput, in terms of actual scheduled jobs

All scheduling reports are uploaded and enqueued into mirroring queues ready to be inserted in the database. With respect to Version 9.1, caching and multithreading mechanisms, based on multiple queues, have been implemented allowing to exceed the previous single thread bottleneck that limited the capacity to almost 300 job results updated per minute. The actual default of five queue threads sustains a global capacity of 5.3K status updates per minute. This causes two main positive effects:

- Removal of the Dynamic Workload Console status update delay
- Absence of a full plan resync on the database because mirroring queues never reach their maximum size

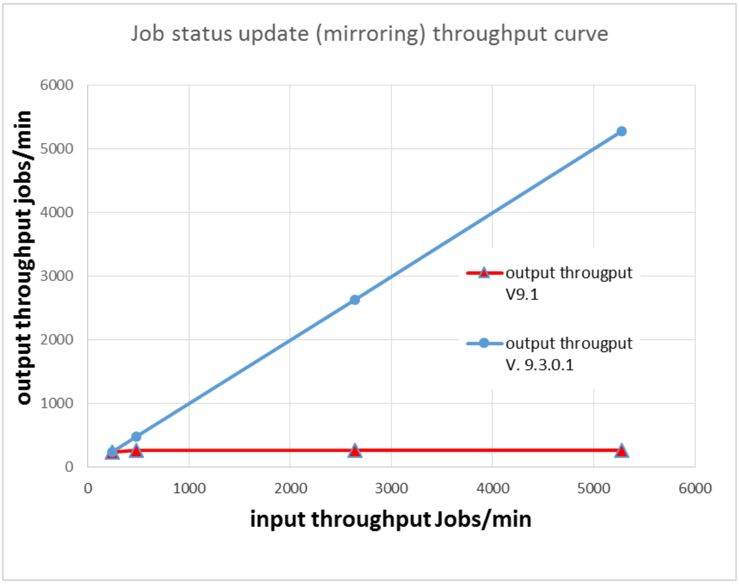


Figure 9. Incoming throughput, in terms of job results, vs output throughput, in terms of mirroring queue

events processed per unit of time.

This changed throughput curve behavior can be easily thought to cause a different scale property for resource usages. In the 9.1 version of the product, the dynamic scheduling bottleneck causes the scalability resource usage curve to have a slope reduction after the maximum capacity is reached (the resource increment was dedicated to handling the incoming queue and not necessarily to process the messages in the queue) resulting in an absolute non-linear behavior. The 9.3.0.1 version of the product showed a linear trend, at least within 0 - 2.6K jobs/min dynamic schedule range. As represented in **Figure 10** it could be noted the differences with 9.1 version trend as expected but also the positive impact of Broker optimization at lower workload in terms of resource usage (it is going faster consuming less power). This new behavior could increase the degree of freedom in choosing correct scaling approach. Previous capacity planning activity, that forecast an intensive high dynamic scheduling workload, couldn't do without considering an additional Dynamic Workload Broker component (horizontal scaling). Currently, with new release, the incremented throughputs could allow to keep only one Dynamic Workload Broker and, accordingly with resource utilization, adding additional resources such as CPU (vertical scaling).

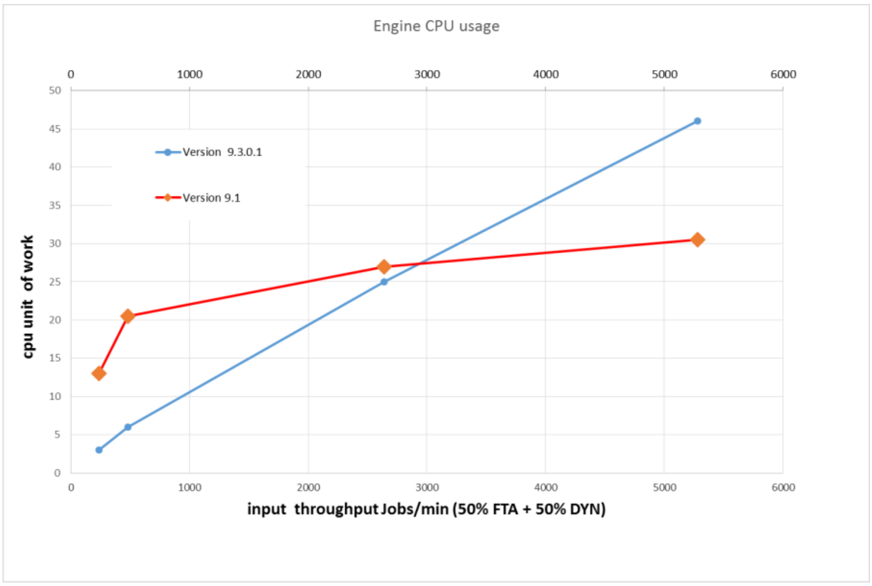


Figure 10. The distribution of the CPU usage is compared between version 9.1 and 9.3.0.1 as a whole

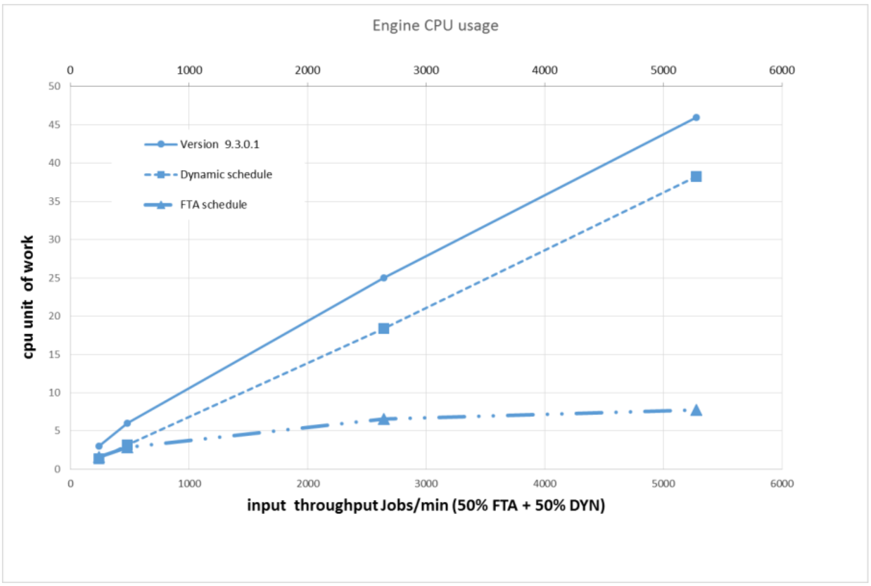


Figure 11. The distribution of the CPU usage for 9.3.0.1 dynamic (50%) agent job submissions, and fault-tolerant (50%) agent job submissions are depicted

As illustrated in section 4.1, dynamic agent job submission flow is different from the fault-tolerant agent flow. The executed scenarios have a balanced mixed workload of both types. An additional test demonstrated how different agent submission types contribute to the CPU usage.

Performance data for new features: conditional dependencies performance impact analysis

Tests with Version 9.3.0.1 were augmented with an additional 3200 jobs defined across 800 job streams and scheduled with over 4 dynamic agents and 4 fault-tolerant agents. This means that there are 100 job streams for each agent, half of which have internal dependencies, and the other half have external dependencies. These 100 job streams per agent are scheduled uniformly over time between 11:00 and 11:50. In relation to conditional dependencies, there are also over 800 join conditions.

The result was that there was no performance impact on the product KPIs (dynamic scheduling throughput, mirroring throughput, jobs submission delay on dynamic agents and FTAs) and on engine and database server resource consumption KPIs (average CPU usage, average disk busy).

4.1.3. Event-driven workload automation (EDWA) rules (IBM Workload Scheduler objects) deployed on the dynamic domain manager

The main objective of this additional performance evaluation was to carry out an assessment on the performance of the `monman` process on the AIX platform in a scenario with hundreds of “TWSObjectsMonitor" event rules deployed on a dynamic domain manager. This special need comes from complaints about the amount of time it takes to process the `monman` queue on the dynamic domain manager running on the AIX platform observed with previous product releases.

In particular, in addition to the Medium Scenarios, a new kind of workload was defined based on IBM Workload Scheduler TWSObjectsMonitor events, with special focus on sub-events such as Job Status Changed (Succ and Fail) and Job Late, that produced a log message each time a rule event was matched.

The following picture shows a comparison between the Version 9.1 and 9.3.0.1 releases on how the `monbox` queue is processed with 200 “IWS objects” event rules:

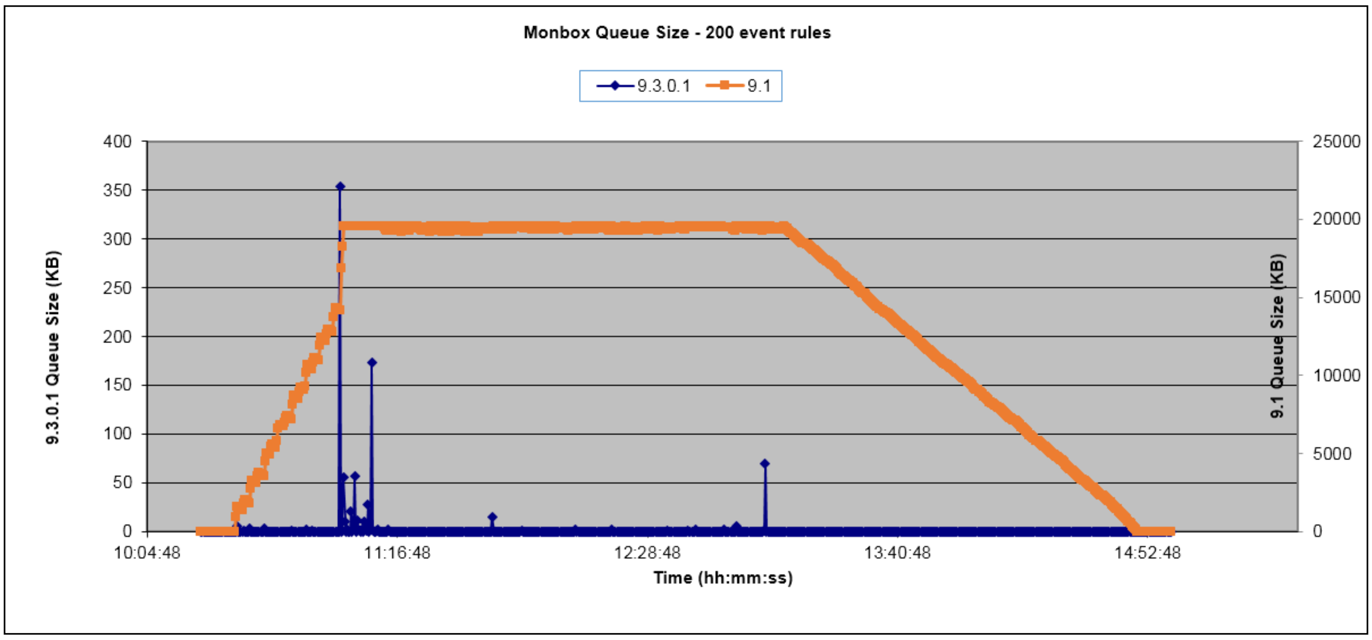


Figure 12. Monbox queue population with 200 defined rules

The main positive effects highlighted by the behavior of version 9.3.0.1 are:

1. There was no delay in processing the `monbox` queue. In the previous product releases, this delay was around 1 hour and 40 minutes.

2. 100% of expected events were matched and processed. In the previous releases, only 26% of expected events were processed because the maximum queue size was reached quickly during the Medium Plan 10-minute peak execution.

The next step was to collect basic capacity planning information scaling up with the number of event rules (1000, 1500 and 2000) in this type of scenario:

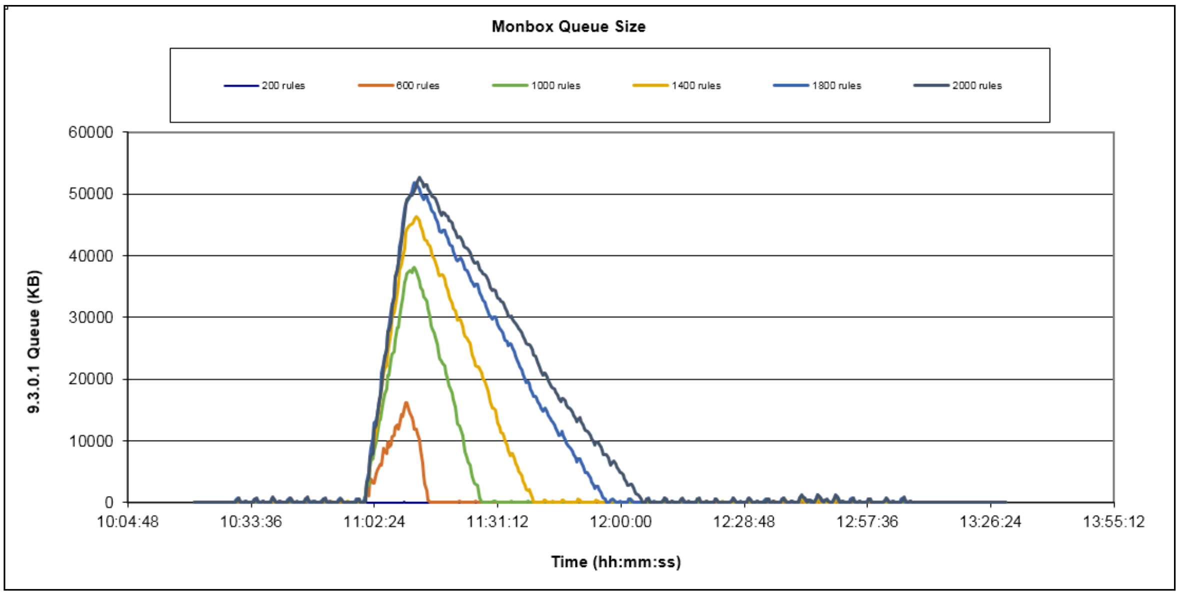


Figure 13 Monbox queue behavior changing the number of defined rules keeping constant the job schedule rates

Figure 13 shows how the `monbox` queue is processed with some delays starting from the case with 600 "IWS objects" event rules, reaching up to the maximum value of 65 minutes of delay in the case of 2000 event rules. This behavior is not optimal and is planned to be improved in the next major release of the product.

Figure 14 shows a linear interpolation that has been obtained using the data points collected during the different test runs:

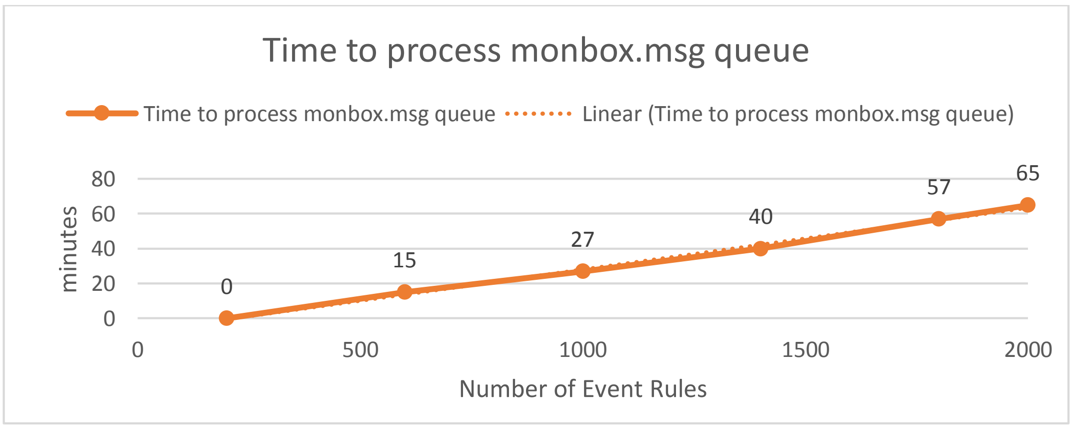


Figure 14. Time to process queue vs number of rules (constant job workload)

This data should only be used as a qualitative indicator because it depends on the workload and the hardware infrastructure used for the test simulations done in the lab.

4.1.4. Storage impact analysis

I/O activity is intensive on both engine and database nodes. In particular, a disk write throughput could reach an average of 11 and 14 MB per second in the peak time lapse of the “Large” scenario, for engine and database respectively. The I/O capacity of the system is not simply revealed by the speed of copying a file, but it depends on disk access concurrency, file size, and type of read and write. The typical marker for potential I/O bottlenecks is the disk busy metering.

It was possible, in this context, to test the performance in two equivalent software and hardware configurations that differ for the storage solution. The first environment called Environment A, described in **Figure 15**, consists of a midrange customer solution of a SAN Volume controller and an IBM DS5020. The second environment, called Environment B, described in **Figure 16**, consists of a STORWIZE 3700 solution.

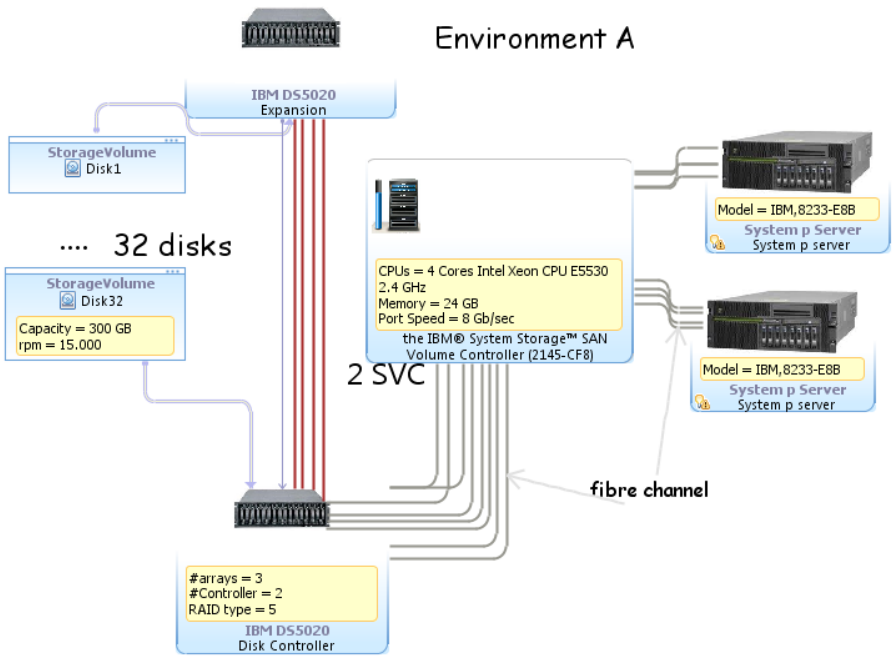


Figure 15 .Environment A based on SAN Volume Controller + DS5020.

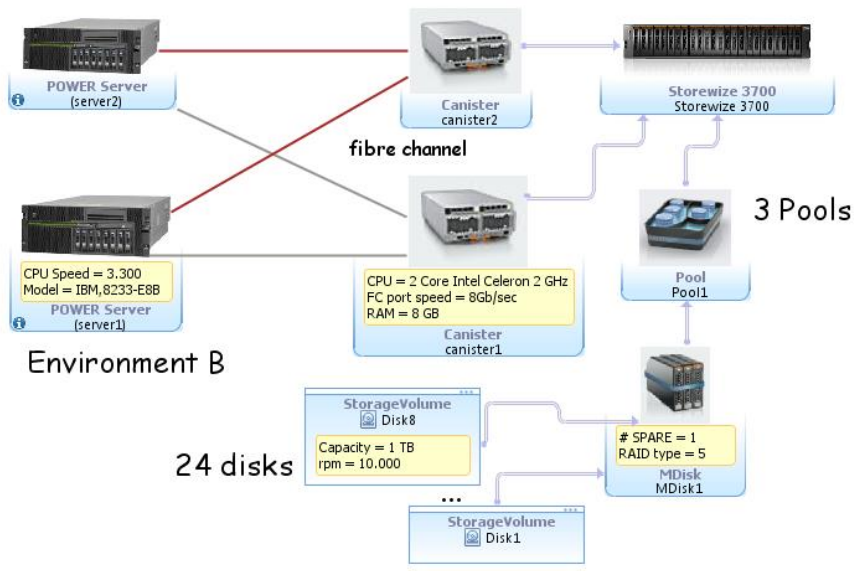


Figure 16. Environment B based on Storwize 3700 solution.

The same "Medium" workload was applied in the two environments. The comparison of two different dynamic scheduling output throughput highlights a decline during a peak phase for Environment B.

	Average Dynamic Schedule Throughput @ Peak	Decline
Environment A	1300	-36%
Environment B	830	

Table 4. "Medium" workload throughput for Environment A and Environment B

I/O metering was collected and compared in **Table 5**.

		xfer (activities/sec)	Disk Busy	Disk Write (MB/sec)	Disk Read (MB/sec)
Environment A	Engine	1627 (writes)	47%	6.6	-
	DB	2023 (662 read+1360 writes)	43%	7.3	5.2
Environment B	Engine	1070 (writes)	56%	4.3	-
	DB	1292(522 read+770 writes)	60%	4	6

Table 5."Medium" workload I/O metering for Environment A and Environment B

The presence of a storage bottleneck becomes clearly apparent on examining two different trends: reduction in I/O activity and an increment in the disk busy percentage.

The Iozone industry standard benchmark was used to analyze and to quantify storage throughputs. The command was executed with following parameters:

```
-R -l 5 -u 5 -r 4k -s 100m –F file1 …file 5
```

It performs a set of different read and write benchmark writing: 100 MB files with 4KB records with 5 thread concurrency. The results of the benchmark are displayed in **Figure 17** and they outline the almost equivalent capacity for the read activities while the write ones are declining by about 38%.

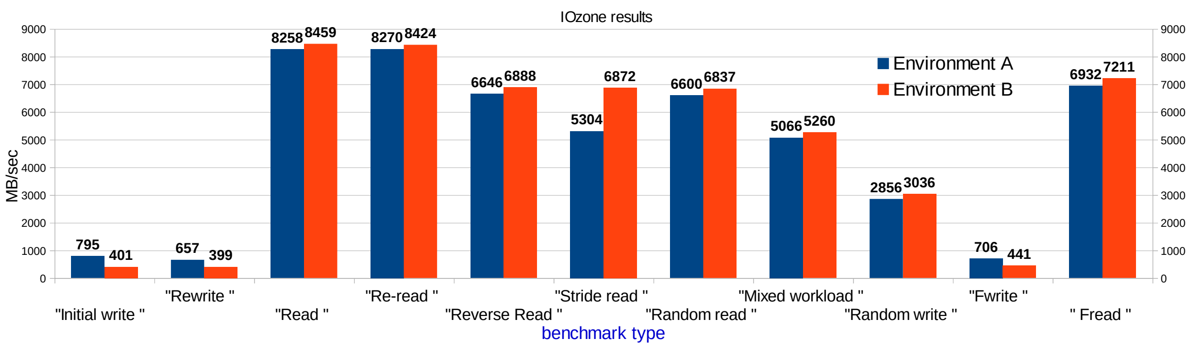


Figure 17. IOzone benchmark results.

For instance, a deeper investigation underlined that a large part of the gap between the two environments could be due to the presence of the SAN Volume Controller in the storage solution for Environment A which improved writing capability of this environment.

4.2. User interface scenarios

For the user interface scenarios, 4-node high availability configuration was used to support 700 concurrent users (175 users per node). Each user in the automation framework (Rational Performance Tester) logs in and completes three transactions before logging out and reentering again with different credentials. The delay between each transaction is controlled by the framework to have a frequency of:

20 transactions/hour per user

Concurrency test was composed of single stage of 700 users with an overall concurrency of around:

35 pages/second

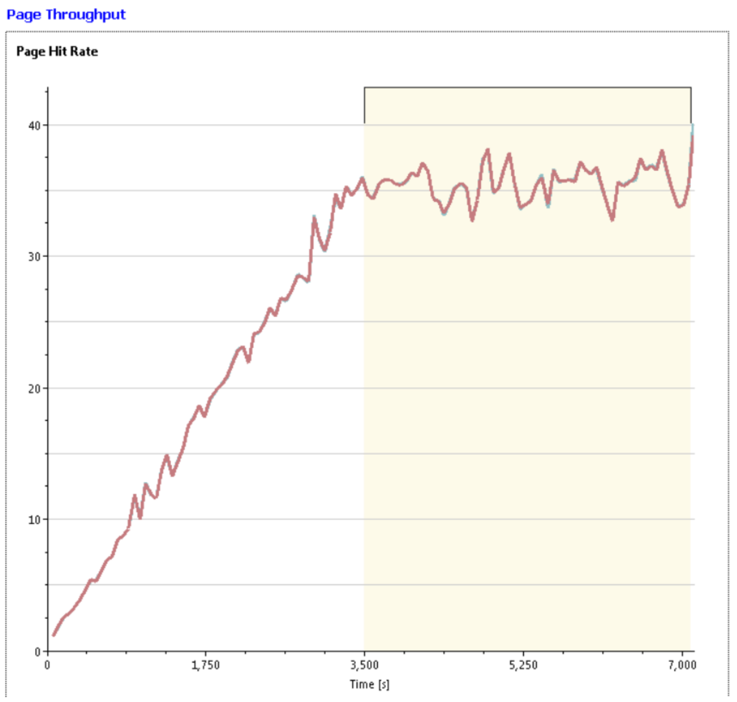


Figure 18. Page hit rate caused by a 700-user test workload

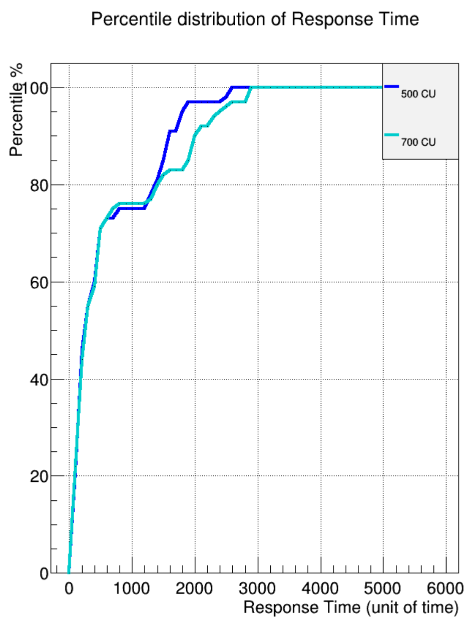


Figure 19. Percentile distribution of response time against 500 and 700 concurrent users

The test run with 500 users was performed in a configuration with 3 Dynamic Workload Console nodes and a heap size of 4 GB. Instead, the test run with 700 concurrent users was performed in a configuration with 4 Dynamic Workload Console nodes and a heap size of 6 GB.

Some recommendations to maximize the performance and reliability of the product when there are 300 or more concurrent users are:

1. Add a couple of database indexes on column NAME for the DASH database tables NODES and STORES to resolve a known issue that will be fixed in the next Jazz for Service Management release.
2. Ensure you do not preserve more than 1000 records of service requests in the Self-Service Catalog application history.
3. Archive job reports on the agents regularly to avoid impact on performance when retrieving job log transactions from the server side.

5. Recommendations

5.1. CPU capacity

All tests described in this document have been executed on **P7 IBM 8233-E8B 3GHz** processors assigned exclusively to LPAR (no shared pools or capping feature have been applied). While planning the correct CPU sizing, the information provided in Table 9 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 was outlined in section 4.1.4 in relation with the product performance. The numbers presented in *Table 5* could be used as reference while planning a solution and the output of I/O Industry standard benchmark, such as IOzone, as key performance indicators to compare with that reference.

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 -200
DWC heap size	1 GB	2 GB	4 - 6 GB

Table 6. Dynamic Workload Console WebSphere Application Server heap configuration

Schedule (jobs per min)	1 – 50	50 -100	100-200	>200
IWS Engine heap size	1 GB	1.5 GB	2 GB	4 GB

Table 7. Engine WebSphere Application Server heap configuration

In addition to the above memory requirements, the native memory for the Java™ process and IBM Workload Scheduler process should be taken into consideration.

5.4. Tunings and settings

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

	Parameter	Value	Comment
UI Node	Dynamic Workload Console configuration settings repository (see http://www-01.ibm.com/support/knowledgecenter/SSGSPN_9.3.0/com.ibm.tivoli.itws.doc_9.3/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 [100, 200] users per node Suggested: 6144 for [150, 200] users per node	
	WebSphere Application Server JVM options	-Djava.awt.headless=true -Dsun.rmi.dgc.ackTimeout=10000 -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 Connections	300	

IBM Workload Scheduler engine	WebSphere Application Server JDBC max Connections		300	
	WebSphere Application Server JVM max heap = min heap		2048 - 4096	
	WebSphere Application Server JVM options		-Djava.awt.headless=true -Dsun.rmi.dgc.ackTimeout=10000 -Xdisableexplicitgc -Xgcpolicy:gencon –Xmn 512m	- Xmn 1024m if heap size = 4096
DB	LOGPRIMARY		200	780 MB total transaction log space
	LOGFILSIZ		1000	
	KEEPFENCED		NO	
	dbMAX_CONNECTION		AUTOMATIC	
	STMT_CONC		LITERALS	This setting optimizes query executions and reduces CPU usage
	Db APPL_MEMORY, APPLHEAPSZ, DATABASE_MEMORY, DBHEAP		AUTOMATIC	
	Db AUTO_RUNSTAT		ON	
	AUTO_REORG		OFF	
	TWS_PLN_BUFFPOOL	NPAGES	182000	
		PAGESIZE	4096	
	TWS_BUFFPOOL_TEMP	NPAGES	500	
		PAGESIZE	16354	
	TWS_BUFFPOOL	NPAGES	8192	
		PAGESIZE	1000	

TDWB	JobDispatcherConfig.properties	Queue settings	`Queue.actions.0 = cancel, cancelAllocation, cancelOrphanAllocation`
			`Queue.size.0 = 10`
			`Queue.actions.1 = reallocateAllocation`
			`Queue.size.1 = 10`
			`Queue.actions.2 = updateFai`
			`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 = notificat`
			`Queue.size.6 = 30`	

	ResourceAdvisorConfig.properties	MaxAllocsPerTimeSlot	`1000`	

		TimeSlotLength	`10`	
		MaxAllocsInCache	`50000`	

Table* 8. 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 assuming a mixed scenario similar to the one described in section 3.2.1;
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 9.3.0.1 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 **PowerPC P7** platform; nevertheless, this information could be used as a reference point for different platform architectures.

	NODE	Core Capacity	Disk Throughput Read-Write (MB/sec)	Network Throughput Read-Write (MB/sec)	RAM Capacity (GB)
250K jobs (50% FTA +50% DYN) per day (175 jobs/min) 100 concurrent users					
3Nodes	**IWS-Engine**	2	0-0.5	1-1	3
	RDBMS	1	2-0.5	0.5-1.5	5
	DWC	2	0-0.1	1.2-1	6
500K jobs (50% FTA +50% DYN) per day (350 jobs/min) 100 concurrent users					
3Nodes	**IWS-Engine**	2	0-1	0.9-2	4
	RDBMS	2	2.3-0.9	0.5-1.5	5
	DWC	2	0-0.1	1.2-1	6
750K jobs (50% FTA +50% DYN) per day (485 jobs/min) 100 concurrent users					
3Nodes	**IWS-Engine**	3	0-1.3	1.6-1.3	4
	RDBMS	3	2.3-1.2	1-2.2	5
	DWC	2	0-0.1	1.2-1	6
10K jobs (50% FTA +50% DYN) per day (8 jobs/min) 20 concurrent users					
1Node	**IWS-Engine RDBMS DWC**	1	0.5-0.1	0.5-0.7	5

Table 9. Capacity planning samples

The above capacity planning examples refer to the workload described in section 3.2 . In particular, they are based on job scheduling performed on 50% dynamic agent workstations. If the ratio changes, the engine CPU capacity requirement changes. For example, assuming that all agents are dynamic (100%) the following configuration should be considered:

	NODE	Core Capacity	Disk Throughput Read-Write (MB/sec)	Network Throughput Read-Write (MB/sec)	RAM Capacity (GB)
500K jobs (100% DYN) per day (350 jobs/min) 100 concurrent users					
3Node	**IWS-Engine**	**3**	0-1	0.9-2	4
	RDBMS	2	2.3-0.9	0.5-1.5	5
	DWC	2	0-0.1	1.2-1	6

Table 10. Impact on workload with 100% dynamic agent job scheduling

7. Notices

7.1. Trademarks