Redpaper

Pierre-André Paumelle

Proven Practices for Enhancing Performance: A Q&A for IBM WebSphere ILOG BRMS 7.1

You must consider many aspects when addressing performance questions for a Business Rule Management System (BRMS). Historically, many of the questions have centered around rule execution, such as “How fast is the rule engine?”, “How much memory does it require?”, and “How many CPUs will I need to execute 1,000 rules per second?” As illustrated in this IBM® Redpaper™ document, the answers do not always come from the engine’s intrinsic capabilities, but they depend on the application and the way that the Rule Execution Server is used.

As rule projects grow, more and more performance topics relate to other modules, such as build time in Rule Studio, memory and CPU usage in Rule Team Server, and recommendations around the use of Decision Validation Services.

The BRMS Service is realized through a series of products that can be installed from the following websites:

- http://www.ibm.com/software/integration/business-rule-management/jrules/
- http://www.ibm.com/software/integration/business-rule-management/rule-team-server/
- http://www.ibm.com/software/integration/business-rule-management/decision-validation-services/

IBM® WebSphere® ILOG® JRules V7.1 comprises a set of modules that operate in various environments but also work together to provide a comprehensive BRMS. Figure 1 on page 2 shows the modules in the environments in which they are used, and how they work together through synchronization.

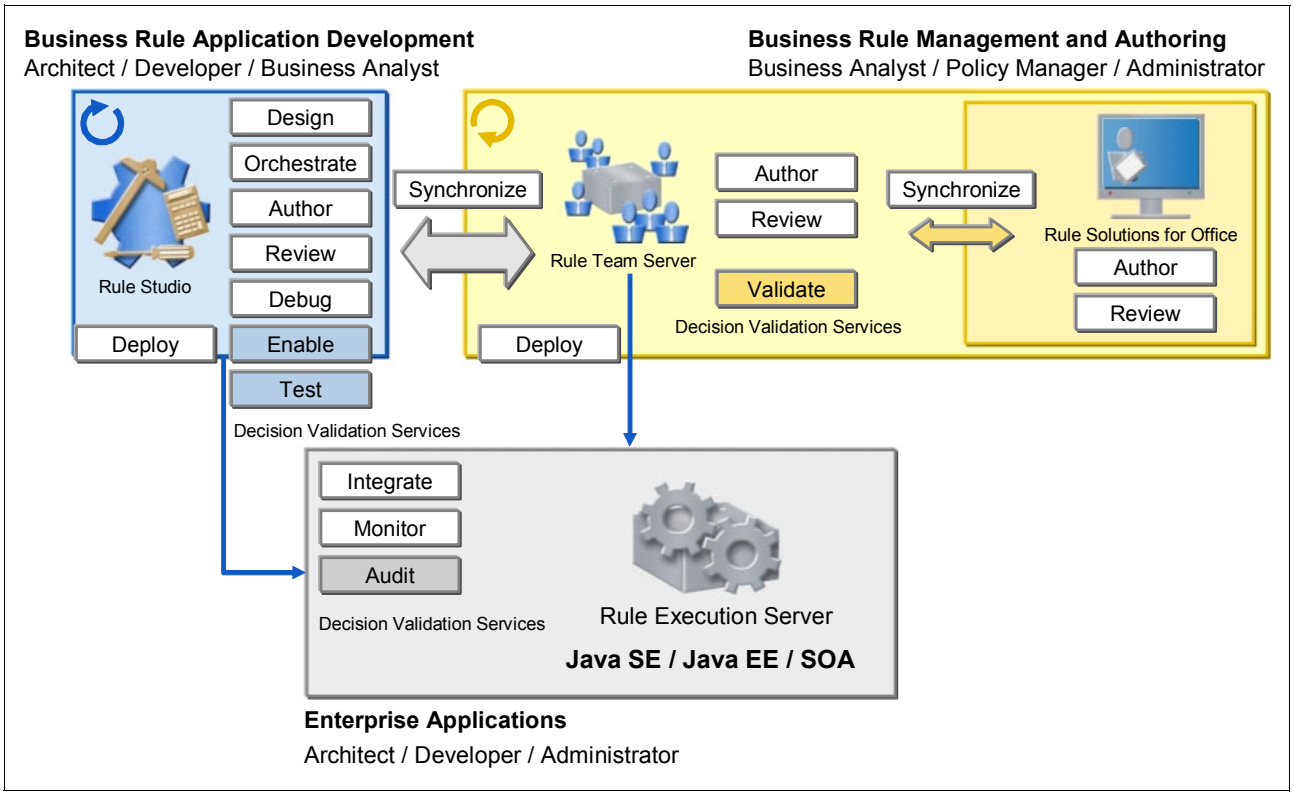


Figure 1 Product architecture

The purpose of this paper is to provide information about the performance aspects of the WebSphere ILOG BRMS V7.1.*x*. A question and answer (Q&A) format is used to cover as many dimensions as possible. Furthermore, performance is considered from various perspectives in this document, with an emphasis on production environments and execution.

Because Rule Execution Server plays a key role in performance, this document starts by answering questions about this module.

You can access the glossary of BRMS terms used in this document in the WebSphere ILOG JRules glossary at the following website:

http://publib.boulder.ibm.com/infocenter/brjrules/v7r1/topic/com.ibm.websphere.ilog.jrules.doc/Content/Business_Rules/Documentation/_pubskel/JRules/ps_JRules_Global2115.html

Enhancing the performance of Rule Execution Server

The Rule Execution Server (Figure 2) is a complete execution and management environment for business rule applications. It provides a number of features that are required for high-performance, scalable, and manageable applications:

- Pooling of rulesets and contexts
- Hot deployment of rulesets and asynchronous parsing.
- File and database persistence of deployed rulesets
- Web-based system administration console
- Runtime monitoring of the execution using the Java Management Extensions (JMX) application programming interface (API)
- Client APIs: Stateless Plain Old Java Object (POJO), stateful POJO, stateless Enterprise JavaBeans (EJB), stateful EJB, stateless Java Standard Edition (JSE), stateful JSE, asynchronous execution and Message Driven EJB for asynchronous invocation, with optional execution trace or with optional auditable execution trace recording (Decision Warehouse)
- Easy service-oriented architecture (SOA) deployment: One-click configuration and integrated monitoring of rules as Transparent Decision Services.
- Provision for simple testing and simulation of effects of rule changes (Decision Validation Services)

Figure 2 shows how the architecture incorporates the execution and management/monitoring stacks.

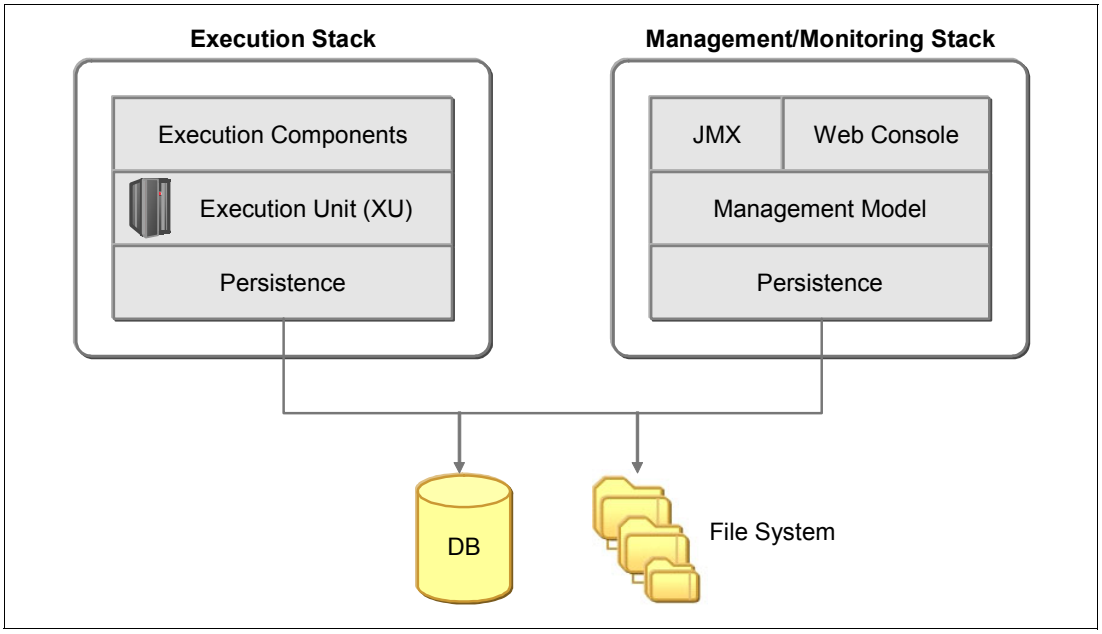


Figure 2 Product architecture

By deploying the Rule Execution Server (for Java Platform, Enterprise Edition (JEE) or JSE applications), your business rule applications automatically take advantage of the execution and management services that are offered by JRules, and you avoid having to implement ruleset and context pooling within your application code. Your overall application build and deployment processes can also integrate the Ant tasks that IBM provides for the Rule Execution Server.

What is the basic configuration for good performance

Use the following guidelines for the basic configuration settings:

- The log level in the Rule Execution Server must be set to the Severe or Warning level in the production environment to increase performance. This property is accessible in the resource adaptor of the Rule Execution Server or in the `ra.xml`.
- The Execution Unit (XU) pool size must be tuned to have a sufficient connection but with a reasonable memory consumption.
- The ruleSession factory must be unique and shared between HTTP sessions; otherwise, a lookup Java Naming and Directory Interface (JNDI) is performed for each factory creation.
- If you do not use stateful rule sessions and the transactions at engine level, you need to deactivate the transaction support in the Rule Execution Server and set NoTransaction instead of LocalTransaction in the properties of the Rule Execution Server (XU) or in the `ra.xml`. If you have any code in your Executable Object Model (XOM) or rules that use a data source, the call to the data source can participate in the transaction (TX) even if the XU transaction support is disabled. The XU transaction support that is set in the `ra.xml` only concerns the engine run time itself and not other resources, even if they are called by the engine.
- Use up-to-date ruleSessions libraries:
  - Do not use old versions.
  - Integrate in the classpath of your application the correct version of the ruleSession.
- Tune the GC and memory size. On IBM Java virtual machine (JVM), the gencon GC policy has good results.
- Use the asynchronous parsing of the Rule Execution Server if you have ruleset update in the application's use cases. This option is available in the properties of Rule Execution Server (XU) or in the `ra.xml`.
- Use the execution trace and the Decision Warehouse only if their use is mandatory, and tune the usage if needed (filtering).
- Limit the size of your XOM to useful classes.
- When on, Java2security halves performance. Java2security needs to be off if possible, especially on JEE.

How do ruleset characteristics impact performance

The size of a ruleset has a big impact on these areas:

- Parsing time
- Execution with traces (number of rules, tasks, and ruleflow)
- Execution with Decision Warehouse (number of rules and tasks)

Performance is greatly impacted by the following issues:

- Ruleflow complexity (Note that the ruleflow is interpreted, not compiled, in WebSphere ILOG JRules V7.1.).
- The usage of dynamic filter in the rule tasks. When possible, use a fixed list of rules.
- Decision table (DT) complexity: Verify the size of the DT in the Rule Studio technical view. Dividing a DT several times can sometimes have a huge impact on the ruleset size by reducing it.
- Algorithm selection (see "How can we decide on Rule Engine execution modes " on page 33).

Hints

Consider these tips:

- Filtering what gets traced can help minimize the performance overhead that is associated with execution tracing and the decision warehouse.
- A ruleset with mixed algorithms (RETE + Sequential or RETE + Fastpath) consumes more memory than a ruleset with a single algorithm.
- You can use the ruleflow with the mutually exclusive rule tasks to reduce the number of rules evaluated during the execution. In this way, the ruleset is segmented and only rules from relevant ruleTasks are evaluated. For example, if a ruleset of 20,000 rules is divided on five ruletasks of 4,000 rules and only one ruletask is evaluated, depending on a simple condition, the performance of this ruleset is similar to the performance of a ruleset with 4,000 rules with a single ruletask on the same XOM.

How does a ruleset's Executable Object Model (XOM) impact performance

XOM impacts performance in the following ways:

- The execution of JRules calls to XOM methods, so that the performance of the XOM is crucial. The Java XOM must be thread-safe for the stability of the system. A synchronized method can have a real impact on performance in the case of concurrent executions. The synchronized section of your XOM must be minimized.
- The Java XOM must implement Serializable if you call the Rule Execution Server remotely.
- The size of the Java XOM deployed in your EAR has an impact at the parsing and execution level (Factor 3 or 5 in the pathologic cases). The class loader must be the smallest possible. For example, do not put the complete JEE jars in the `WEB_INF/lib` directory of your web application.
- A ruleset on an XML XOM must be configured to run in multiple simultaneous executions by configuring the pool of XML document drivers. The ruleset property `ruleset.xmlDocumentDriverPool.maxSize` configures it. The default value is one.
- The toString() method performance can have a huge impact if you need to get an execution trace with the trace filter setInfoBoundObjectByRule set to true.

What impact does the execution mode have on performance (trace or Decision Warehouse)

Consider the following aspects of execution mode:

- No trace:

  Execution without traces is the optimal mode in terms of memory usage and performance.

- Execution trace:

  When you choose the execution mode with trace, you will experience the following side effects:

  - Memory usage increases as the execution trace is generated, the list of rules and tasks is cached, and the rule events are generated.
  - On Sequential or Fastpath, the ruleset property `ruleset.sequential.trace.enabled` set at true enables the generation of events when a rule is fired or a task is executed. The side effect is a code generation that produces event and store mapping information in the ruleset (performance and memory impact).

- The response of the execution is bigger than the original one as the execution trace is integrated. The execution trace size depends on the ruleset characteristics (number of rules and tasks).
- The trace filtering can have a big impact on the size of the execution trace and, therefore, performance.

- Decision Warehouse execution is divided into the following steps:
  - This execution mode depends on the execution trace mode, so we pay for the cost of the execution trace.
  - At the end of execution, a serialization of input and output parameters is done (business object model (BOM) serialization).
  - A complete serialization to XML of the execution trace.
  - A database insertion.
  - You can choose through a list of filters (as ruleset properties) what to save in the Decision Warehouse. To get in the execution trace, input, output parameters, and execution events (tasks and rules executed), set the properties that are shown in Example 1.

    Example 1 Ruleset properties

    ```
    monitoring.filters=INFO_EXECUTION_EVENTS=true,INFO_INPUT_PARAMETERS=true,
    INFO_OUTPUT_PARAMETERS= true, INFO_RULESET_PROPERTIES=false,
    ,INFO_INET_ADDRESS=true,INFO_EXECUTION_DURATION=true,INFO_TOTAL_RULES_FIRED=false,IN
    FO_TOTAL_TASKS_EXECUTED=false, INFO_TOTAL_RULES_NOT_FIRED=false,
    INFO_TOTAL_TASKS_NOT_EXECUTED=false, INFO_RULES_NOT_FIRED=false,
    INFO_TASKS_NOT_EXECUTED=false, INFO_EXECUTION_DURATION=false, INFO_RULES=false,
    INFO_TASKS=false, INFO_SYSTEM_PROPERTIES=false, INFO_WORKING_MEMORY=false,
    INFO_BOUND_OBJECT_BY_RULE=false
    ```

  - The Decision Warehouse is customizable. Further customization can be done through the Decision Warehouse extension point for better execution performance. You can define how data can be persisted or queried through the extension point. An asynchronous version of decision warehouse for WebSphere Application Server V7 is available as a contribution at the following website:

    http://www-01.ibm.com/support/docview.wss?uid=swg21433167/

Important: In the current benchmarks for the Decision Warehouse, the key bottleneck is the database insertion. Therefore, database tuning is extremely important if you use the Decision Warehouse as it is shipped.

How can you choose the architecture of your application (engine, JSE, or JEE Rule Execution Server)

The choice of an architecture depends on your decision service characteristics:

- Smallest memory consumption:
  - The engine alone without the Rule Execution Server
- No concurrent execution:
  - Use the engine alone
  - Use the JSE Rule Execution Server

- Concurrent executions:
  - Use JSE Rule Execution Server
  - Use JEE Rule Execution Server
- Integration with JEE animals (EJB, message-driven bean (MDB), POJO, Service Component Architecture (SCA)):
  - Use JEE Rule Execution Server
- Call as a Web Service:
  - Hosted Transparent Decision Service (HTDS) on XML XOM
  - Monitoring Transparent Decision Service (MTDS) on Java XOM

What impact does the topology have on performance (JSE embedded, JEE shared, or JEE isolated)

The Rule Execution Server can be deployed on three separate topologies. Consider the following information when evaluating the impact of the topology on performance:

- JEE shared:
  - The XU is deployed to the application server and shared (and possibly accessed) by all of the applications that are deployed to the server.

    This mode is analogous to installing a device driver into an operating system, in that the application server becomes globally enhanced with ruleset execution capabilities.
  - From an administrative perspective, one version of the XU is deployed and can be easily upgraded, started, stopped, and monitored using the application server management console or other tools.
  - The XU must be deployed on all nodes of the cluster.
  - The JEE ruleSessions types are available.
- JEE isolated:
  - The XU is deployed within a single application.
  - This mode is a more advanced deployment option that enables multiple applications to be deployed to the same server, and for each application to use completely separate versions of ILOG JRules classes.
  - The XU `Jrules-res-xu-xxx.rar` is embedded in the customer application.
  - The resource is scoped to the customer application.
  - The clustering is linked to the customer application.
  - The JEE ruleSessions types are available.
- JSE embedded:
  - The XU is deployed as a simple Java Standard Edition JAR and Rules implements a lightweight J2EE Connector architecture (J2C) container, as well as the Resource Adapter.
  - This mode allows the XU (and hence, the Rule Execution Server) to be used outside a JEE application server.
  - The application server no longer manages the Resource Adapter in this mode and the ILOG JRules pooling infrastructure is used.
  - The resource is JSE and scoped to the customer application.
  - Only the JSE ruleSession is available.

- Performance impacts:
  - The performance of the JEE shared Rule Execution Server and the performance of the JEE isolated Rule Execution Server are equivalent.
  - The performance of JSE embedded is similar to the performance of JEE shared, but the pool cannot be configured by the application server administrator.

How can we optimize the XU pool size and minimize the parsing impact

The optimal XU pool size depends on the number of possible concurrent connections and the number of rulesets to execute. The limit is reached by the memory consumption. On a dual-core computer, a pool size of 50 is correct. A sizing methodology is available at the following website:

http://www-01.ibm.com/support/docview.wss?uid=swg21400803/

The following scenarios illustrate how you can minimize the performance overhead of ruleset parsing:

- Parsing at the Rule Execution Server launch or the first invocation of the ruleset. In this case, the ruleset must be parsed before the first execution. You can force it with a dummy execution or a call of the loadUptodateRuleset in IlrManagementSession API (Management Session API).
- Update of a ruleset that is already in use. This scenario can be fixed by using the asynchronous parsing.
- A ruleset is used infrequently once a day, for example, and removed from the pool running other rulesets. This scenario can be fixed by setting ruleset.maxIdleTime to 0 on this ruleset. In this case, the ruleset will stay in the pool even if it is not used by a session.

See the *Minimizing the effect of ruleset parsing on the performance of Rule Execution Server* white paper for details. You can access the white paper at the following website:

ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/wsw14089usen/WSW14089USEN.PDF

Rule Execution Server and clustering

The Rule Execution Server is ready for clustering. Each XU is independent at pool level but shares the same Rule Execution Server persistence. The XUs do not share a parsed ruleset, so each time that a ruleset is run for the first time or if the ruleset is not in the pool on a node, the XU will parse the ruleset.

Cluster-wide Java Management Extensions (JMX) servers (as provided by most application servers) allow the Rule Execution Server Management Console to discover and interact with multiple XUs no matter where or how they are deployed across the cluster. Providing a single administration facility for all the rule engines that are deployed across a cluster of machines allows system administrators to gain an understanding of the deployment topology and diagnose runtime or configuration errors.

What is the tooling for analyzing the status of the Rule Execution Server

The Rule Execution Server's status is visible through a tool that is accessible through the server's console.

Retrieving ruleset caching details from the Rule Execution Server Console

You can get detailed ruleset caching information (XU dump) from the Rule Execution Server Console. This information includes how many free connections there are in the pool, which rulesets have been parsed, how much free memory is left, and so on.

Perform these steps to access ruleset caching information:

1. Change the Log Level of the Rule Execution Server Console to **Debug** (Figure 3).

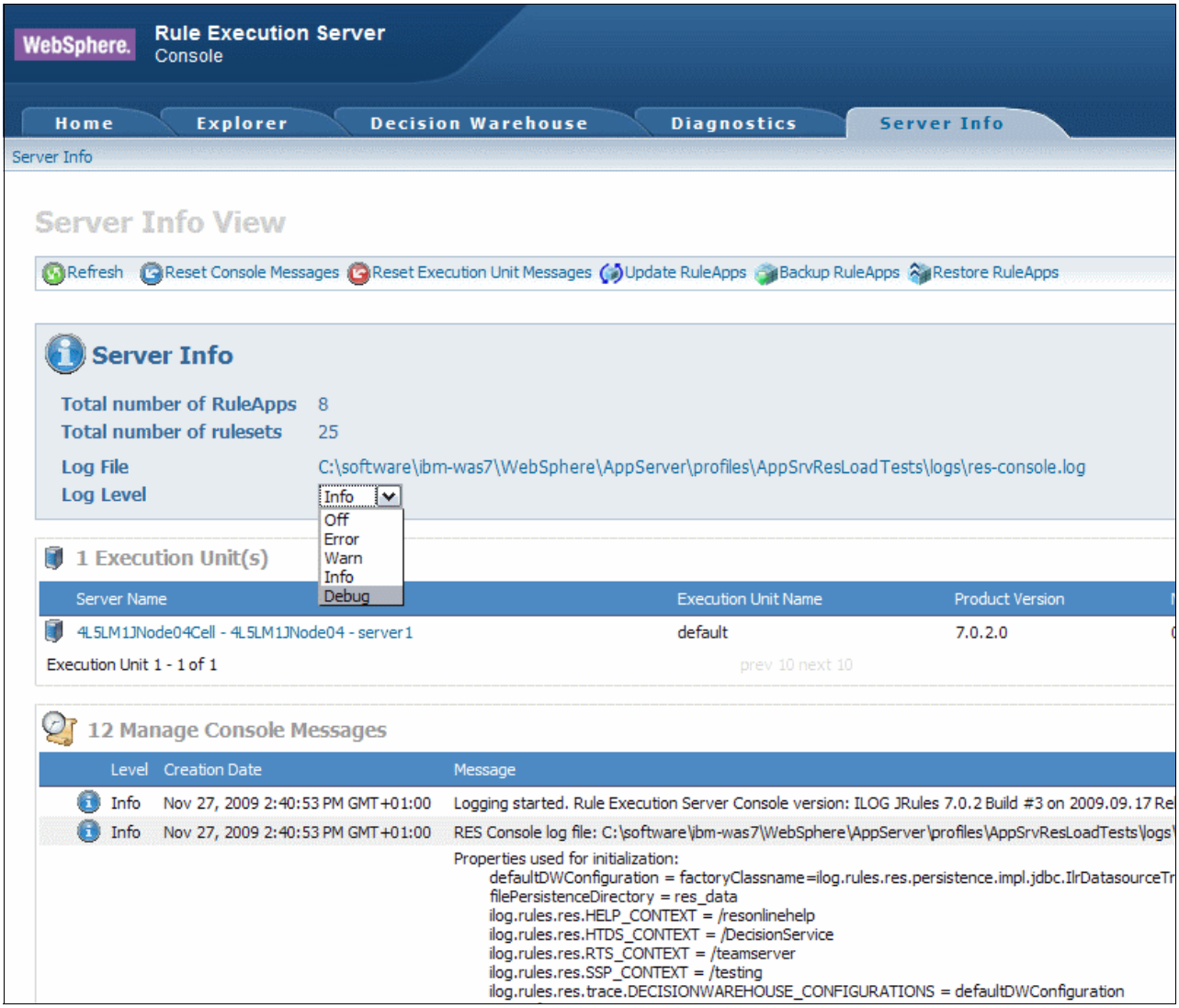


Figure 3 Status of the Rule Execution Server

2. Navigate to the XU that you want to investigate (Figure 4).

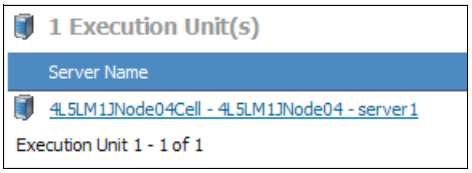


Figure 4 Status of the Rule Execution Server

3. Click **View** (Figure 5).

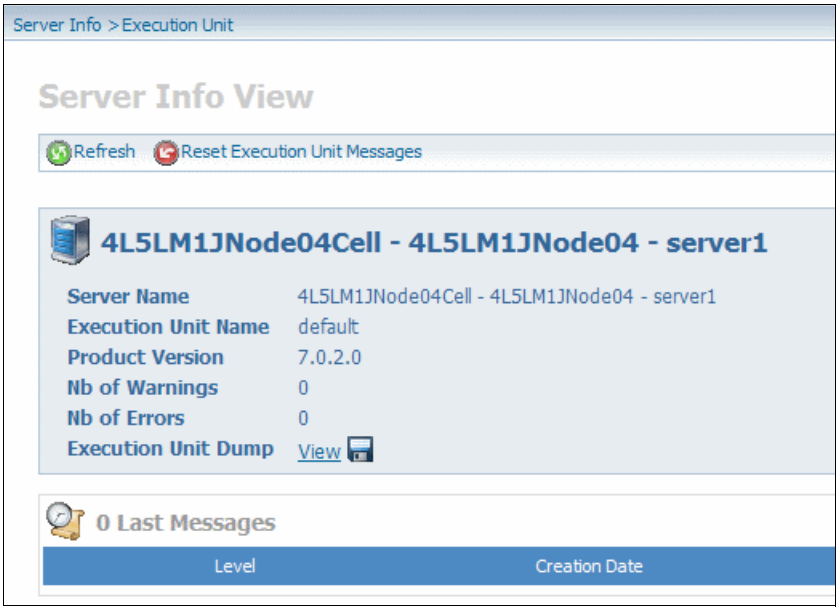


Figure 5 Status of the Rule Execution Server

You can see which rulesets have been parsed and are stored in the cache. Figure 6 shows that two rulesets are currently stored in the cache.

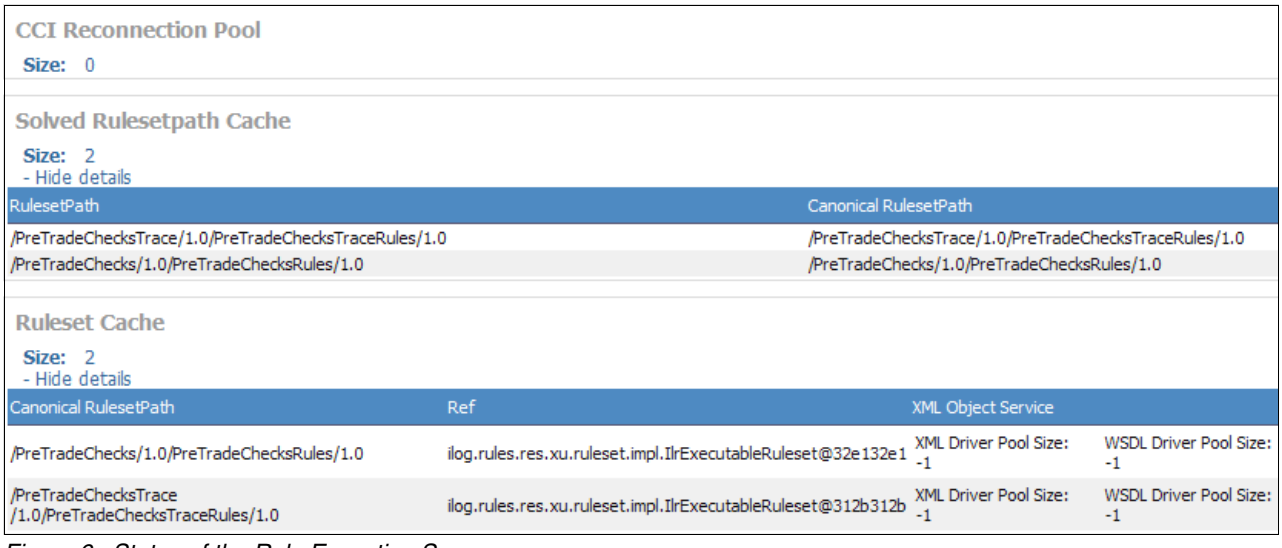


Figure 6 Status of the Rule Execution Server

Do you have benchmarks on Rule Execution Server with the Sequential Algorithm

We provide the following benchmarks to show the impact that the number of rules can have on performance. The number of rules fired is ~10% of the rules. The only difference is the number of rules. The benchmark is performed with an HTTP injector, which is created from 20 to 45 virtual users calling the same web application, to drive the ruleset through a POJO rule session or a Web Service call within HTDS. We ran all benchmarks on IBM WebSphere Application Server V7, with IBM WebSphere ILOG JRules V7.1.1 installed with a shared

instance of Rule Execution Server (for example, JEE shared topology). The duration of each benchmark was one half hour to reduce the impact of the first execution.

Figure 7 and Figure 8 illustrate the benchmark's configuration, spanning two machines on the same subnet:

- One machine with an HTTP injector agent
- One machine with IBM WebSphere Application Server V7, the Rule Execution Server, the benchmark application, and the database

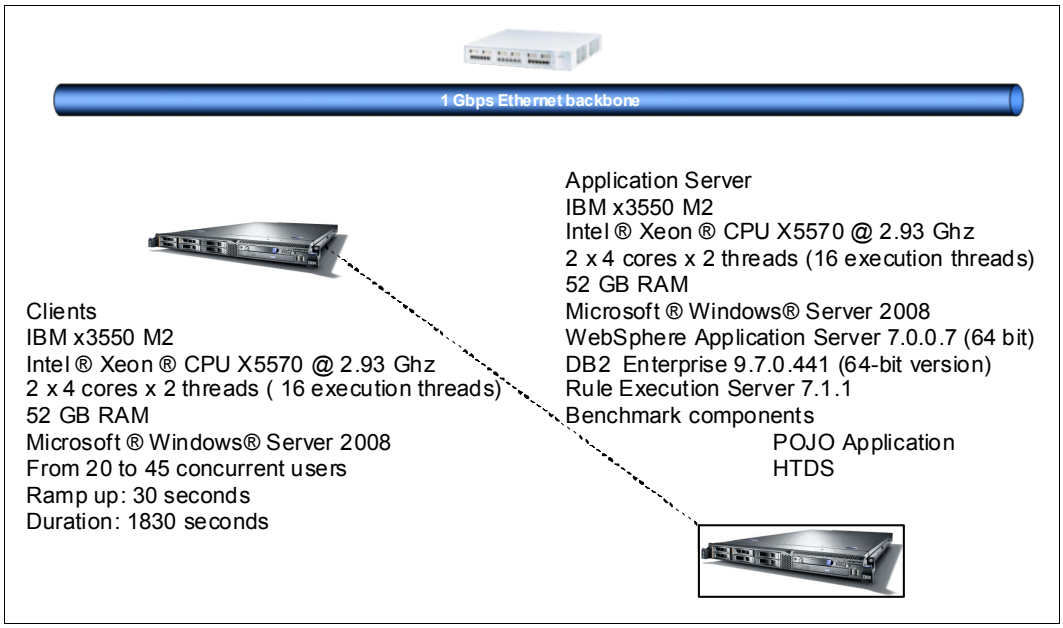


Figure 7 Benchmarks of Rule Execution Server (Sequential)

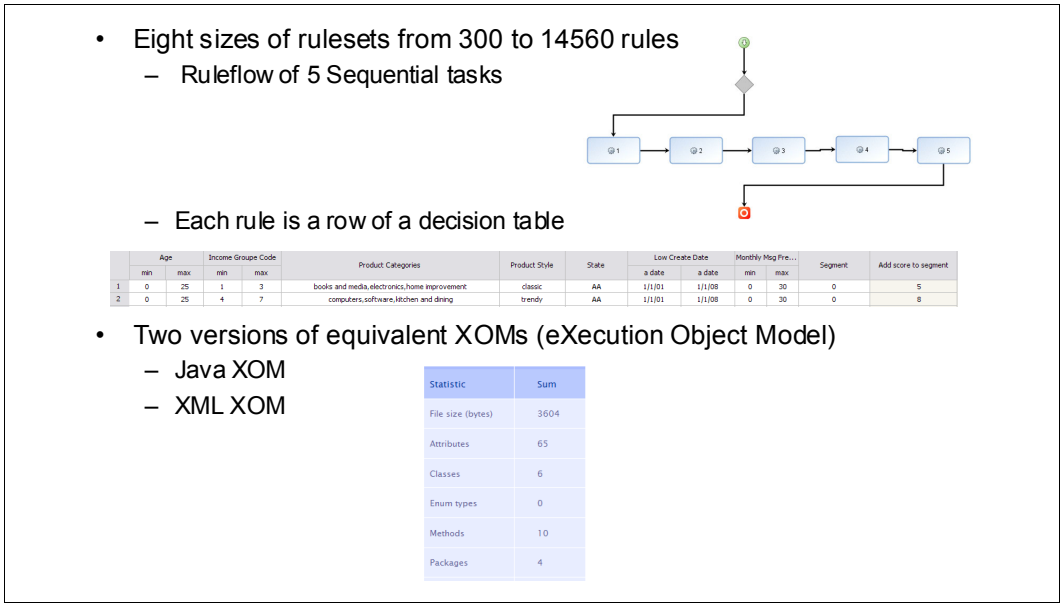
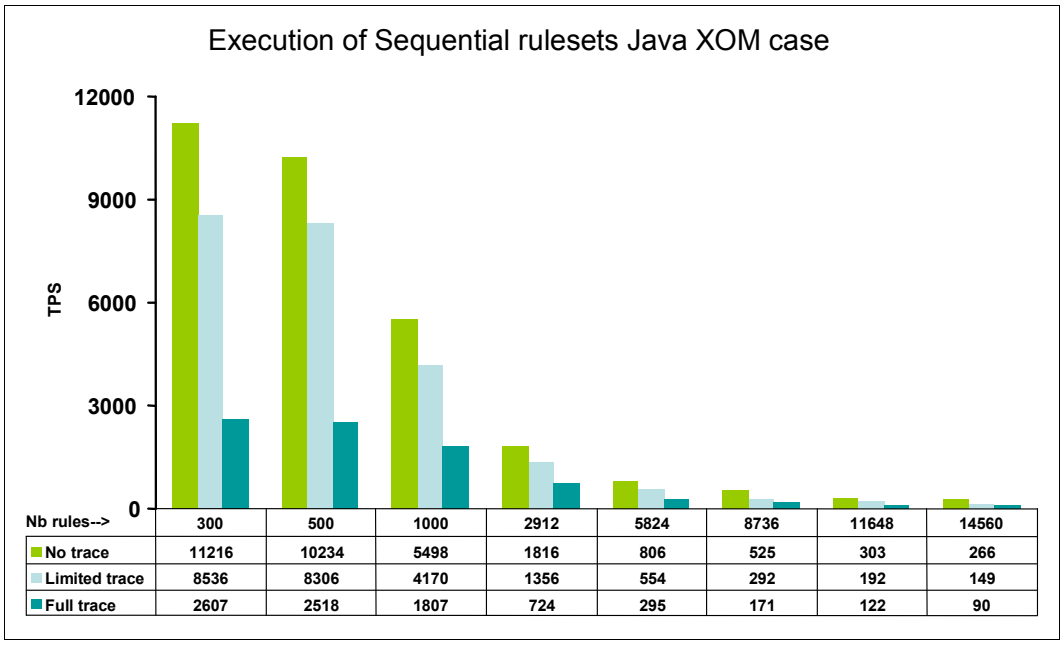


Figure 8 Benchmarks of Rule Execution Server (Sequential)

Executing a ruleset that is based on a Java XOM

Figure 9 shows the following information:

- On the Java XOM, the top performance is ~11,000 transactions per second.
- The number of rules has a direct impact on the execution of rulesets (measured as transactions per second (TPS)).
- Using full trace has a huge impact on the throughput.
- Using filtered trace has a limited impact on the throughput. The limited trace is composed of execution events and input/output parameters.



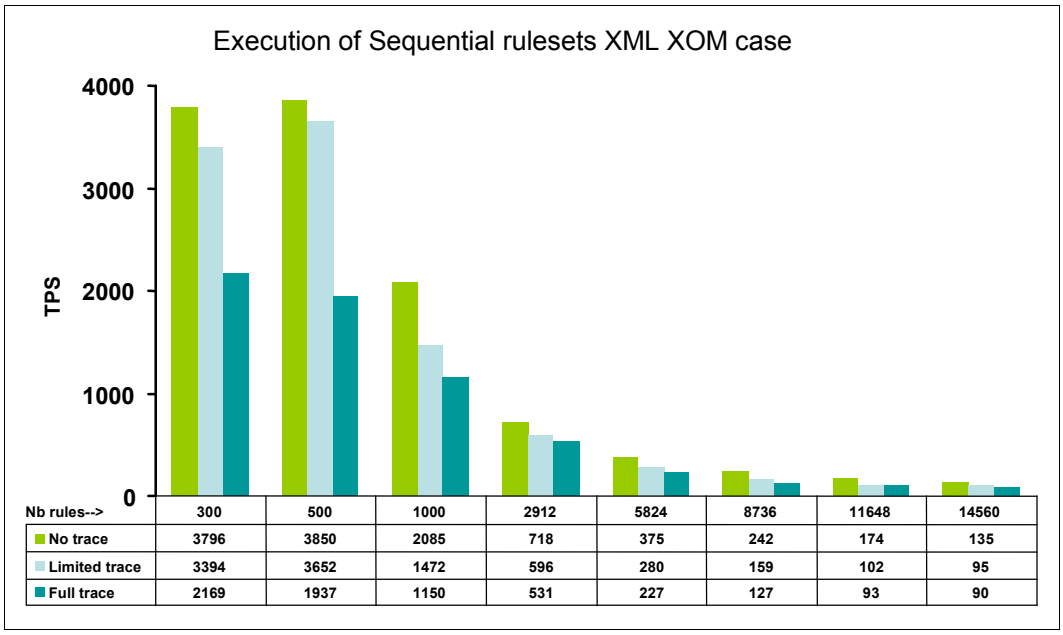
Nb rules-->	300	500	1000	2912	5824	8736	11648	14560
No trace	11216	10234	5498	1816	806	525	303	266
Limited trace	8536	8306	4170	1356	554	292	192	149
Full trace	2607	2518	1807	724	295	171	122	90

Figure 9 Benchmarks on Java XOM

Executing a ruleset that is based on XML XOM

Figure 10 on page 13 shows the following information:

- The top performance of a ruleset on XML XOM is 3,850 TPS.
- A ruleset on XML XOM is between three times to two times slower than an equivalent ruleset on JAVA XOM.
- The number of rules has a direct impact on the execution performance.
- With limited trace, the performance impact is almost transparent.
- The full execution trace has a real impact on throughput, especially on small rulesets.



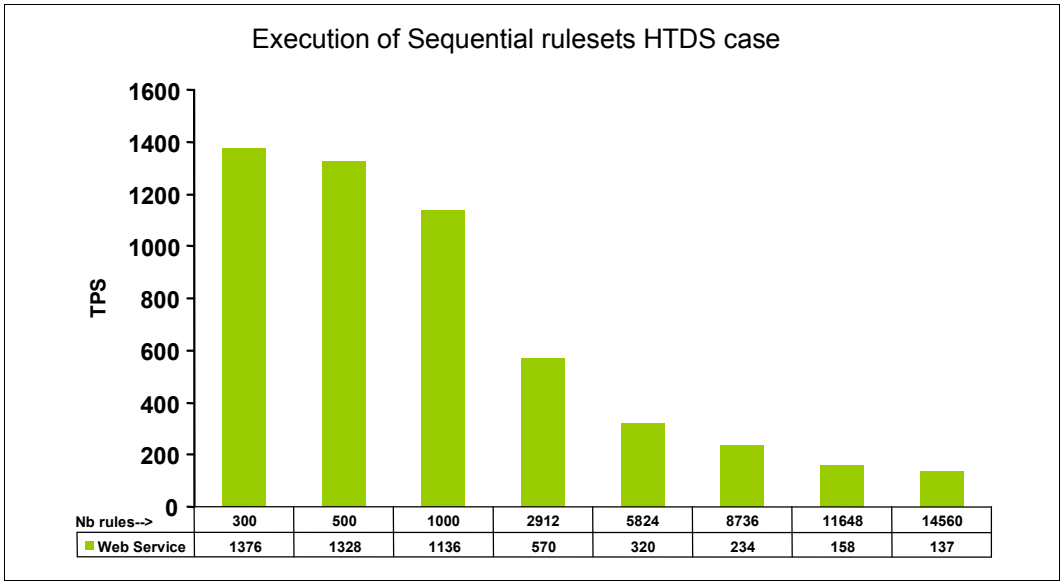
Nb rules-->	300	500	1000	2912	5824	8736	11648	14560
No trace	3796	3850	2085	718	375	242	174	135
Limited trace	3394	3652	1472	596	280	159	102	95
Full trace	2169	1937	1150	531	227	127	93	90

Figure 10 Benchmarks on XML XOM

Executing with HTDS (Web Service invocation)

As you can see in Figure 11, using HTDS has consequences on the performance:

- HTDS can run the XML XOM ruleset only, so we must compare HTDS performance with XML XOM ruleset performance.
- On a small ruleset, HTDS is slower by a factor of 2.5 times than a POJO execution.
- On a big ruleset, HTDS performance is similar to POJO execution on an XML XOM ruleset.

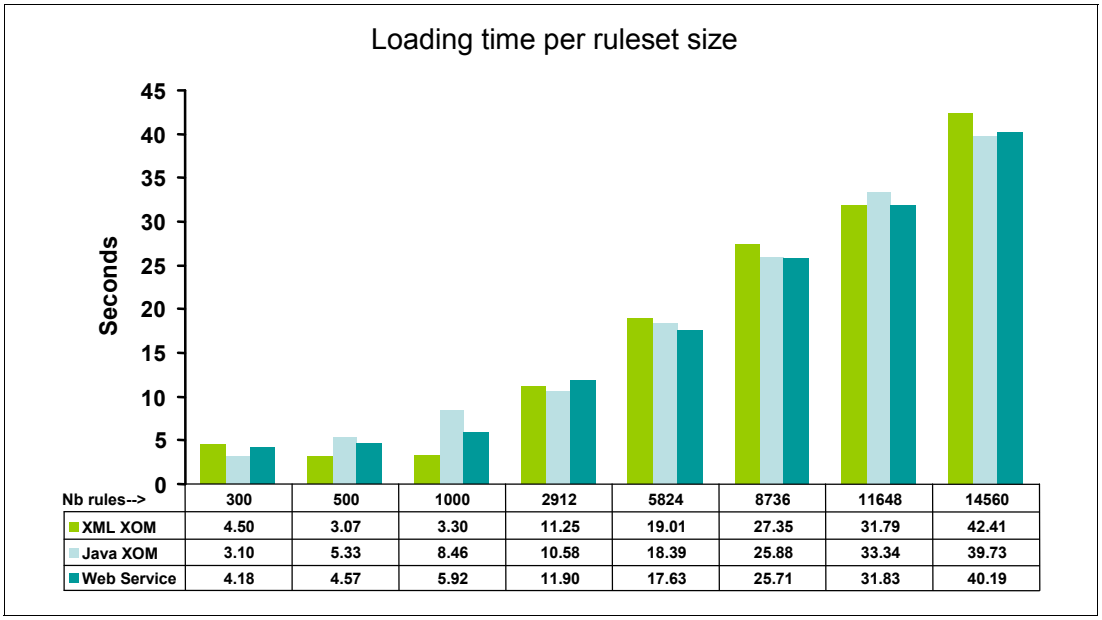


Nb rules-->	300	500	1000	2912	5824	8736	11648	14560
Web Service	1376	1328	1136	570	320	234	158	137

Figure 11 Benchmarks on Web Service (HTDS)

Loading time with a Sequential Algorithm

As you can see in Figure 12, the number of rules has a direct impact on the ruleset parsing duration (measured in milliseconds). This loading time is close to the response time for the first execution.



Nb rules-->	300	500	1000	2912	5824	8736	11648	14560
XML XOM	4.50	3.07	3.30	11.25	19.01	27.35	31.79	42.41
Java XOM	3.10	5.33	8.46	10.58	18.39	25.88	33.34	39.73
Web Service	4.18	4.57	5.92	11.90	17.63	25.71	31.83	40.19

Figure 12 Loading time for Sequential benchmarks on distributed

Performance impact of the number of concurrent users

In this benchmark, we run the ruleset with 1,000 rules on a Java XOM, with a variable number of concurrent users.

The benchmark shows the scalability of WebSphere ILOG JRules versus the number of concurrent users.

As you can see in Figure 13 on page 15, the number of concurrent users has a direct impact on the response time:

- The average response time stays relatively stable when the number of concurrent users is lower than the number of cores.
- When the number of concurrent users is higher than the number of cores, the response time increases quickly, because the CPU used is between 85% to 96%. So, we reach the limit of the system hardware.

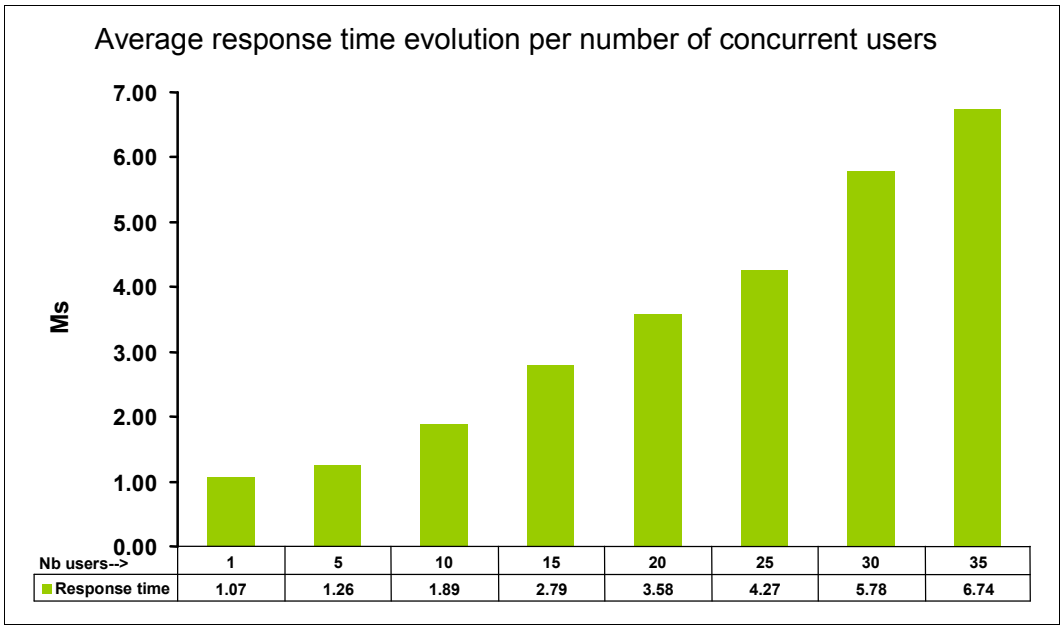


Figure 13 Response time evolution

As you can see in Figure 14, the number of concurrent users has a direct impact on the throughput:

- The throughput becomes stable to ~5,200 TPS when the number of concurrent users is higher than the number of cores.
- The performance of this ruleset peaks with 25 users. This peak depends on the configuration and the ruleset. For example, the peak throughput for the 300 rules ruleset is 45 users.

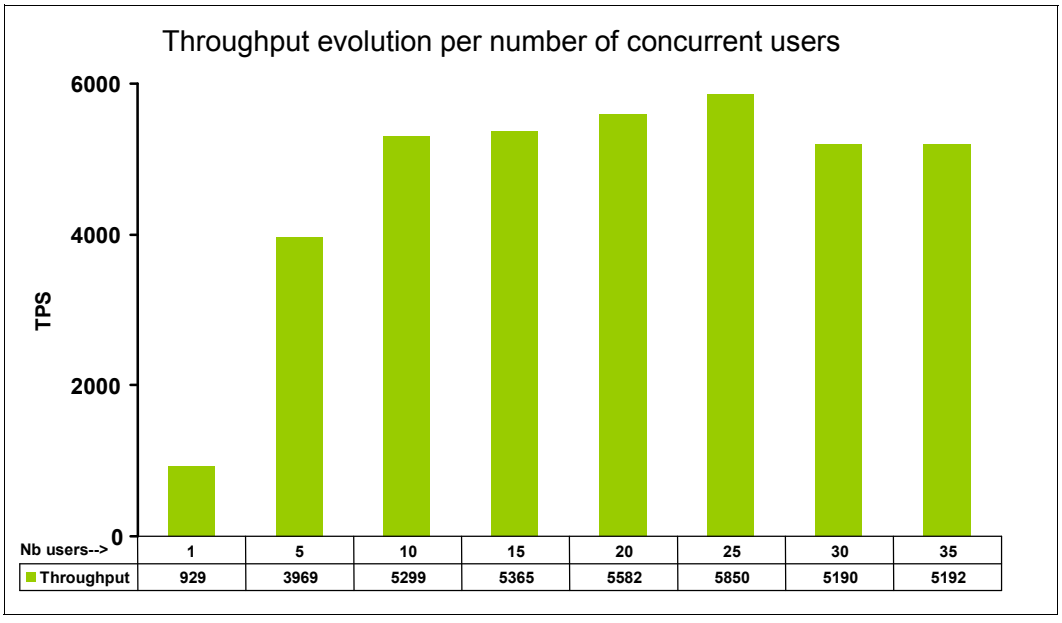


Figure 14 Throughput evolution

Conclusion

These benchmarks show the following conclusions:

- Rule Execution Server execution:
  - The best performance on Java XOM is more than 11,000 TPS on a 300 rules ruleset.
  - Execution on Java XOM is faster than on XML XOM by a factor of 1.75x to 3x.
  - The use of full execution trace has a huge performance impact (factor of 2.5x to 4.3x).
  - The use of limited execution trace has a reasonable performance impact (5% to 80%).
  - HTDS SOAP overload is sensible on small rulesets, not on big rulesets.
  - The number of increased concurrent users has the following impacts:
    - The throughput becomes stable around a limit depending on the complexity of the ruleset when the number of concurrent users is higher than the number of cores.
    - The average response time increases with the number of concurrent users, especially when the number of concurrent users is higher than the number of cores.
- Loading the ruleset during benchmark:
  - The loading time depends on the ruleset size, not on its type.

What is the performance improvement by switching to the Fastpath Algorithm

We provide the following benchmarks to show the impact that the algorithm choice and the number of rules can have on performance. The number of rules fired is ~10% of the rules. The only difference is the number of rules. The benchmark is performed with an HTTP injector, which was created from 20 to 45 virtual users calling the same web application, to drive the ruleset through a POJO rule session or a Web Service call within HTDS. We ran all benchmarks on IBM WebSphere Application Server V7, with IBM WebSphere ILOG JRules V7.1.1 installed with a shared instance of Rule Execution Server (for example, JEE shared topology). The duration of each benchmark was one half hour to reduce the impact of the first execution.

Figure 15 on page 17 and Figure 16 on page 17 show the benchmark's configuration, spanning two machines on the same subnet:

- One machine with an HTTP injector agent
- One machine with IBM WebSphere Application Server V7, the Rule Execution Server, the benchmark application, and the database

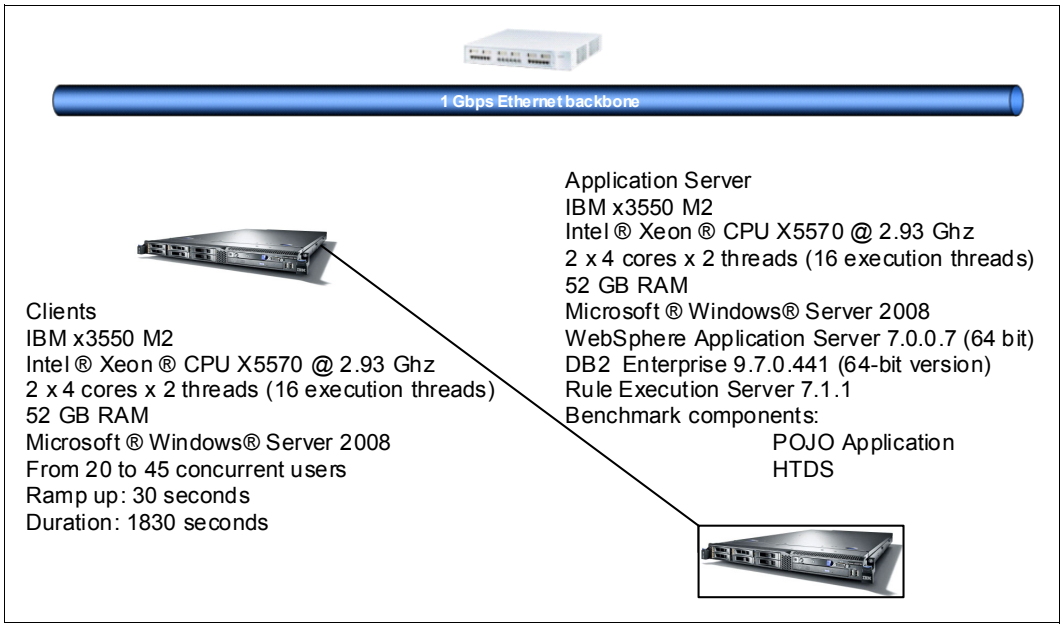


Figure 15 Benchmarks of Rule Execution Server (Fastpath)

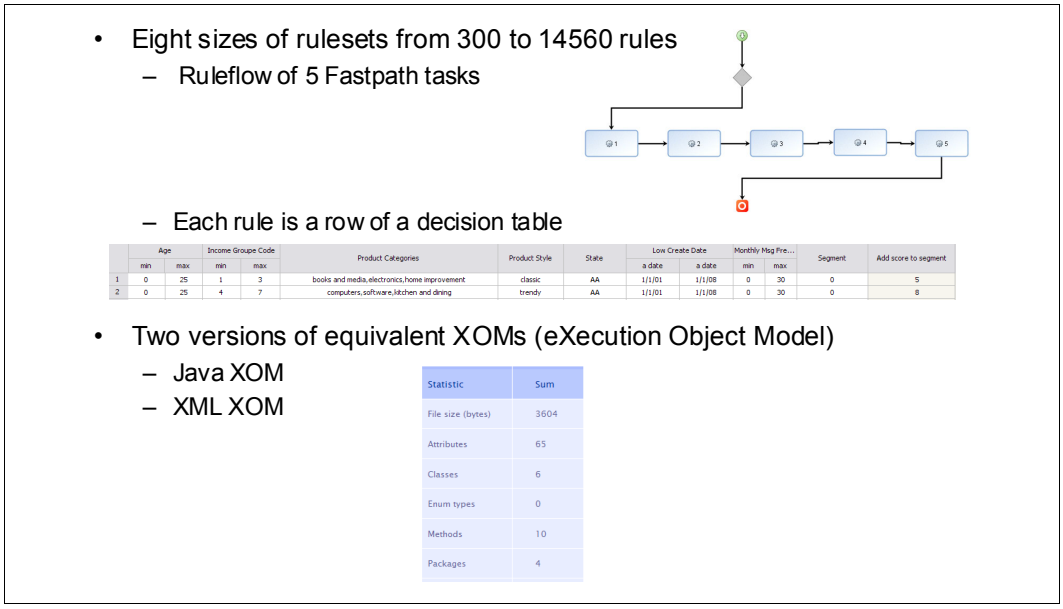
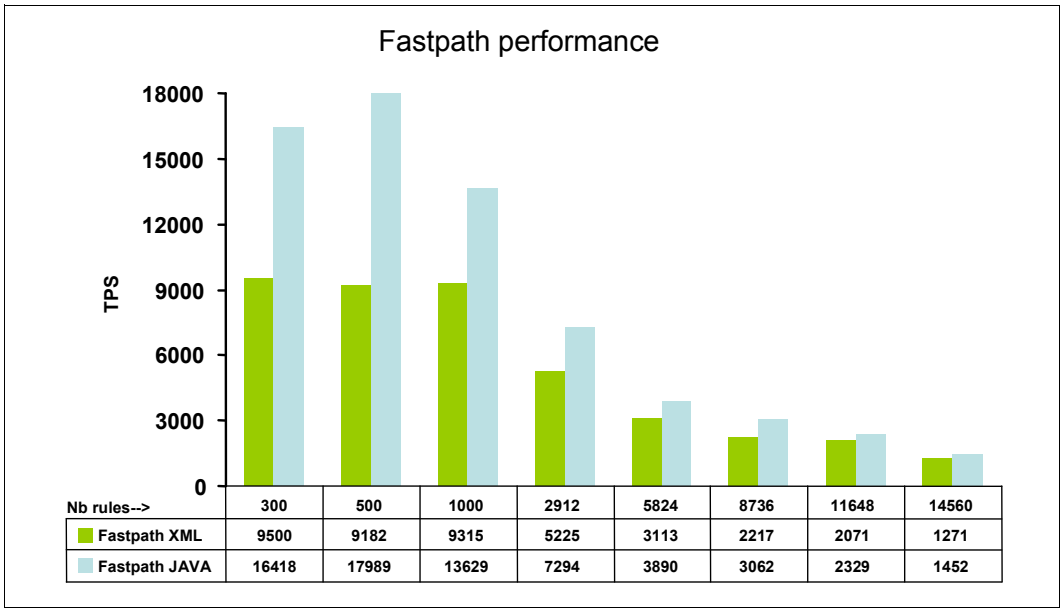


Figure 16 Benchmarks of Rule Execution Server (Fastpath)

Execution of rulesets using Fastpath

Figure 17 on page 18 shows these effects with Fastpath:

- The performance delta of JAVA XOM versus XML XOM is reduced.
- The usage of XML XOM becomes valuable.
- When the size of the rulesets increases the XML XOM, ruleset performance is closer to the JAVA XOM ruleset performance.



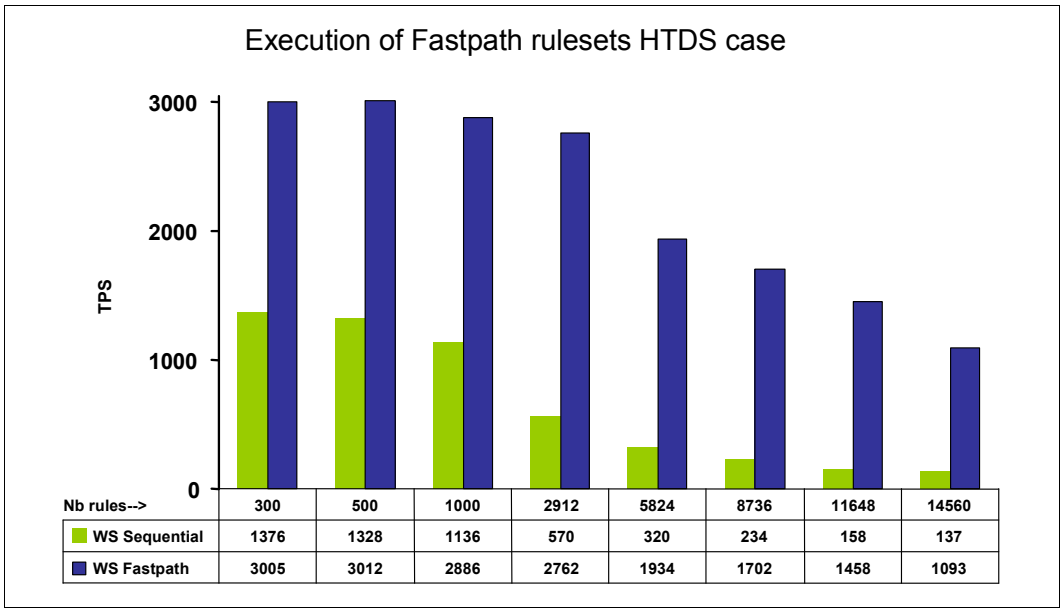
Nb rules-->	300	500	1000	2912	5824	8736	11648	14560
Fastpath XML	9500	9182	9315	5225	3113	2217	2071	1271
Fastpath JAVA	16418	17989	13629	7294	3890	3062	2329	1452

Figure 17 Benchmarks comparison of XML XOM versus JAVA XOM

Performance comparison of Sequential versus Fastpath with HTDS

Figure 18 shows these effects of the usage of a Web Service call with Fastpath:

- The application is more than twice as fast as the Sequential application on small rulesets.
- On a big ruleset, the performance delta between Fastpath and Sequential increases to almost a factor of 10x.



Nb rules-->	300	500	1000	2912	5824	8736	11648	14560
WS Sequential	1376	1328	1136	570	320	234	158	137
WS Fastpath	3005	3012	2886	2762	1934	1702	1458	1093

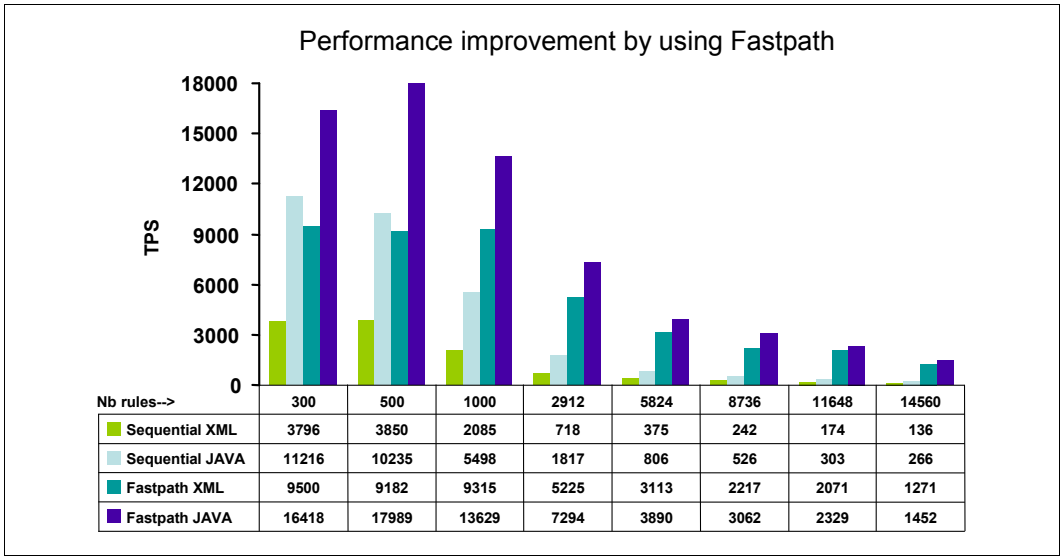
Figure 18 Benchmarks on Web Service call: Comparison of Fastpath versus Sequential algorithm

Performance comparison of Sequential versus Fastpath algorithm

Figure 19 on page 19 shows this performance comparison:

- Fastpath is much more scalable than Sequential on a ruleset with Decision tables.

- Fastpath on XML XOM has comparable performance with Sequential on JAVA XOM on small rulesets.
- Fastpath on XML XOM is faster than Sequential on Java XOM on big rulesets.



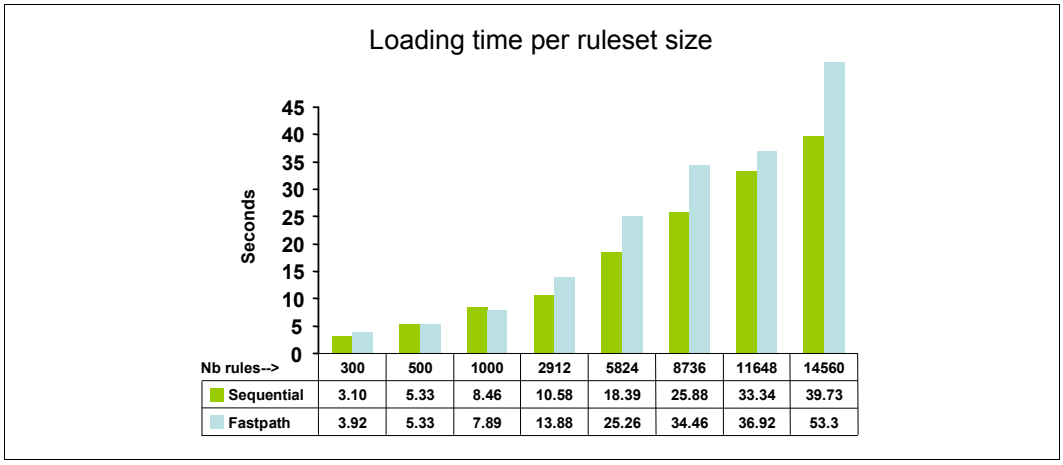
Nb rules-->	300	500	1000	2912	5824	8736	11648	14560
Sequential XML	3796	3850	2085	718	375	242	174	136
Sequential JAVA	11216	10235	5498	1817	806	526	303	266
Fastpath XML	9500	9182	9315	5225	3113	2217	2071	1271
Fastpath JAVA	16418	17989	13629	7294	3890	3062	2329	1452

Figure 19 Performance comparison Sequential versus Fastpath

Loading time

Figure 20 shows the following results:

- The loading time is longer with Fastpath than with Sequential.
- The delta is between 10% to 75%.



Nb rules-->	300	500	1000	2912	5824	8736	11648	14560
Sequential	3.10	5.33	8.46	10.58	18.39	25.88	33.34	39.73
Fastpath	3.92	5.33	7.89	13.88	25.26	34.46	36.92	53.3

Figure 20 Sequential versus Fastpath benchmarks on distributed

Conclusion

These benchmarks show the following conclusions:

- Rule Execution Server execution improvement (TPS):
  - Between 1.46x to 11.86x faster than Sequential.
  - Average improvement factor = 5x.
  - Best performance of Fastpath is 18,000 TPS on a 500 rules ruleset.

- Loading ruleset during benchmark:
  - Between 10% to 75% slower than Sequential loading time.
- Fastpath is much more scalable than Sequential.
- Using Decision table implies that you try Fastpath Algorithm.

Do we have a performance improvement since WebSphere ILOG JRules V6.7

We provide the following benchmarks to show the impact of updating an application from IBM WebSphere ILOG JRules V6.7 to WebSphere ILOG JRules V7.1. In this benchmark, the number of rules fired is ~10% of the rules. The only difference between the rulesets is the number of rules. The benchmark is performed with an HTTP injector, which created from 20 to 35 virtual users calling the same web application, to drive the ruleset through a POJO rule session or a Web Service call within HTDS. We ran all benchmarks on IBM WebSphere Application Server V6.1, with IBM WebSphere ILOG JRules V6.7.4 installed with a shared instance of Rule Execution Server (for example, JEE shared topology). The duration of each benchmark was one half hour to reduce the impact of the first execution.

We compare IBM WebSphere ILOG JRules V6.7 Sequential on WebSphere Application Server V6.1 versus IBM WebSphere ILOG JRules V7.1 Sequential on WebSphere Application Server V7.

Figure 21 and Figure 22 on page 21 show the benchmarks configuration spanning two machines on the same subnet:

- One machine with an HTTP injector agent
- One machine with IBM WebSphere Application Server V6.1, the Rule Execution Server, the benchmark application, and the database

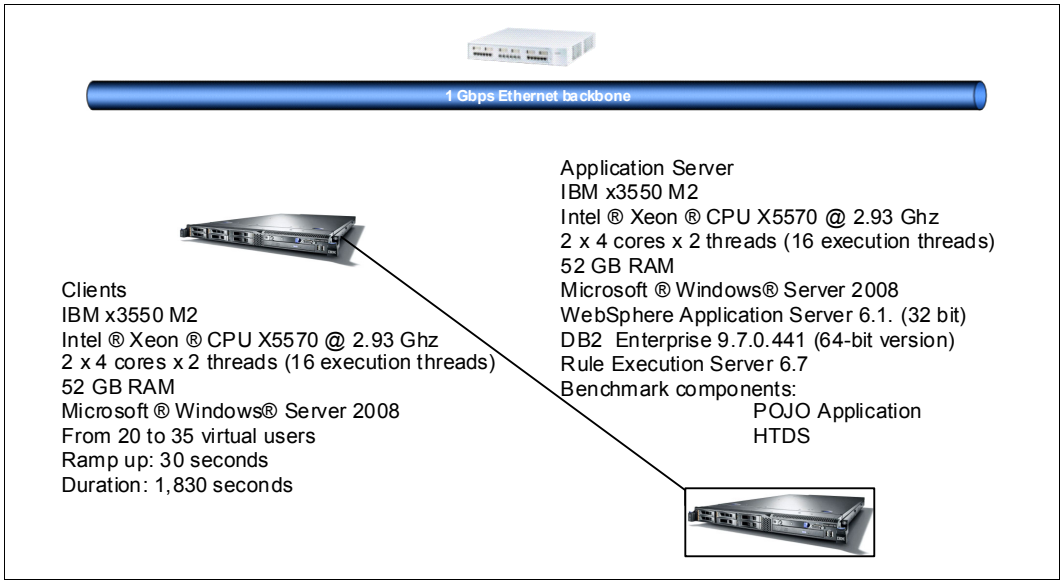


Figure 21 Benchmarks of Rule Execution Server (IBM WebSphere ILOG JRules V6.7)

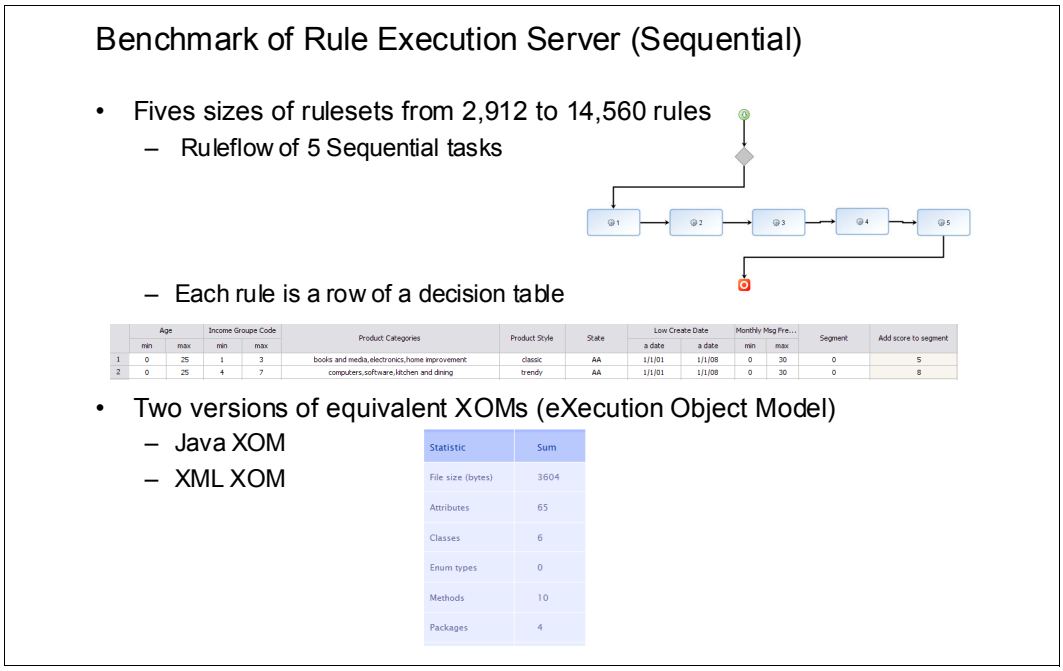
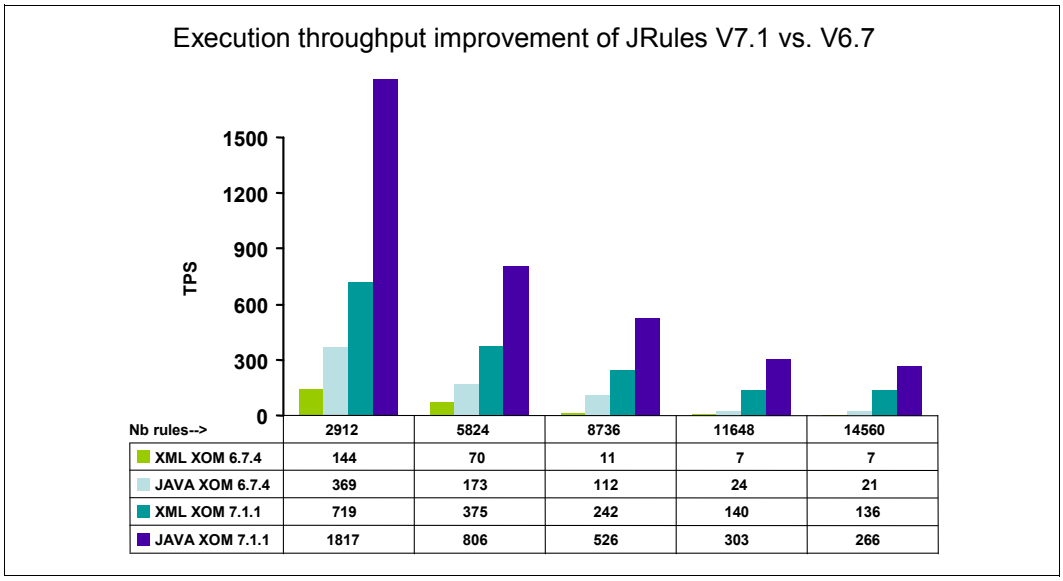


Figure 22 Benchmarks of Rule Execution Server (WebSphere ILOG JRules V6.7)

Performance comparison IBM WebSphere ILOG JRules V7.11 versus WebSphere ILOG JRules V6.7.4

Figure 23 shows the following results:

- IBM WebSphere ILOG JRules V7.1 is much more scalable than IBM WebSphere ILOG JRules V6.7.
- IBM WebSphere ILOG JRules V7.1 has better performance on every type of execution.



Nb rules-->	2912	5824	8736	11648	14560
XML XOM 6.7.4	144	70	11	7	7
JAVA XOM 6.7.4	369	173	112	24	21
XML XOM 7.1.1	719	375	242	140	136
JAVA XOM 7.1.1	1817	806	526	303	266

Figure 23 Benchmarks comparison Sequential Algorithm JRules V6.7.4 versus JRules V7.1.1

Performance comparison IBM WebSphere ILOG JRules V7.11 versus IBM WebSphere ILOG JRules V6.7.4 with HTDS

Figure 24 shows these results:

- HTDS V6.7 was not scalable.
- HTDS V7.1 is much more scalable than HTDS V6.7.
- HTDS V7.1 on a ruleset of ~15,000 rules has the same performance as HTDS V6.7 on a ruleset of ~3,000 rules.

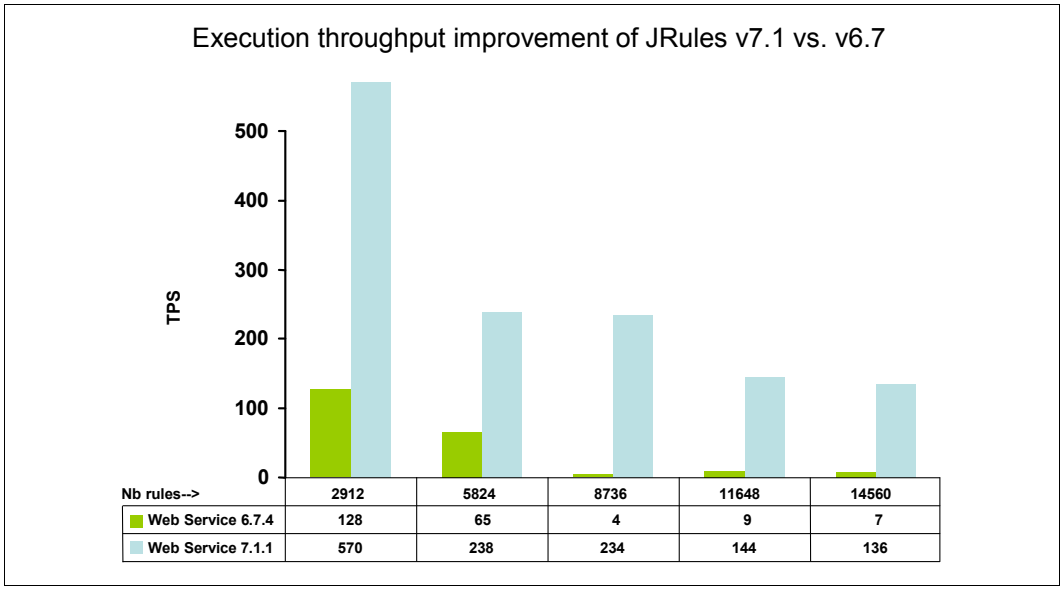
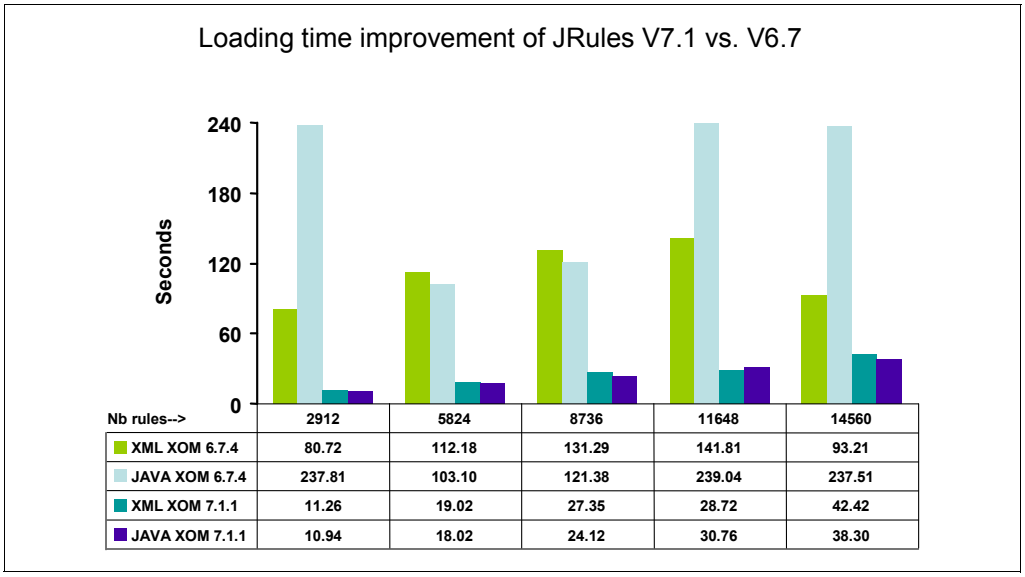


Figure 24 Benchmarks on Web Service call: comparison of JRules V6.7.4 versus JRules V7.1.1

Loading time comparison of IBM WebSphere ILOG JRules V7.11 versus IBM WebSphere ILOG V6.7.4

Figure 25 on page 23 shows the following results:

- The loading time with IBM WebSphere ILOG JRules V7.1 is shorter than with IBM WebSphere ILOG JRules V6.7.
- This improvement comes from various factors:
  - The new schema database is more scalable.
  - The improvement of ruleset parsing, especially in a multi-threaded environment, helps shorten the loading time.



Nb rules-->	2912	5824	8736	11648	14560
XML XOM 6.7.4	80.72	112.18	131.29	141.81	93.21
JAVA XOM 6.7.4	237.81	103.10	121.38	239.04	237.51
XML XOM 7.1.1	11.26	19.02	27.35	28.72	42.42
JAVA XOM 7.1.1	10.94	18.02	24.12	30.76	38.30

Figure 25 WebSphere ILOG JRules V6.7.4 versus JRules V7.1.1 benchmarks on distributed

Conclusion

These benchmarks show the following conclusions:

- Rule Execution Server execution improvement (TPS):
  - Between 3.66x to 59.97x faster.
  - Average improvement factor = 12x.
- Loading ruleset with a single execution:
  - Between 1.09x to 4.45x faster.
  - Average improvement factor = 2x.
- Loading ruleset during benchmark. (Stressed case. A great number of concurrent executions are waiting during the parsing.)
  - Between 1.5x to 21.75x faster.
  - Average improvement factor = 4x.
- WebSphere ILOG JRules V7.1 is much more scalable than WebSphere ILOG JRules V6.7:
  - The performance improvement with WebSphere ILOG JRules V7.1 on a multicore machine is linked to the number of cores.

Enhancing the performance of Decision Warehouse

The Decision Warehouse traces decisions in production applications. Figure 26 on page 24 shows the interface that is used to search and view decisions that are stored in the warehouse.

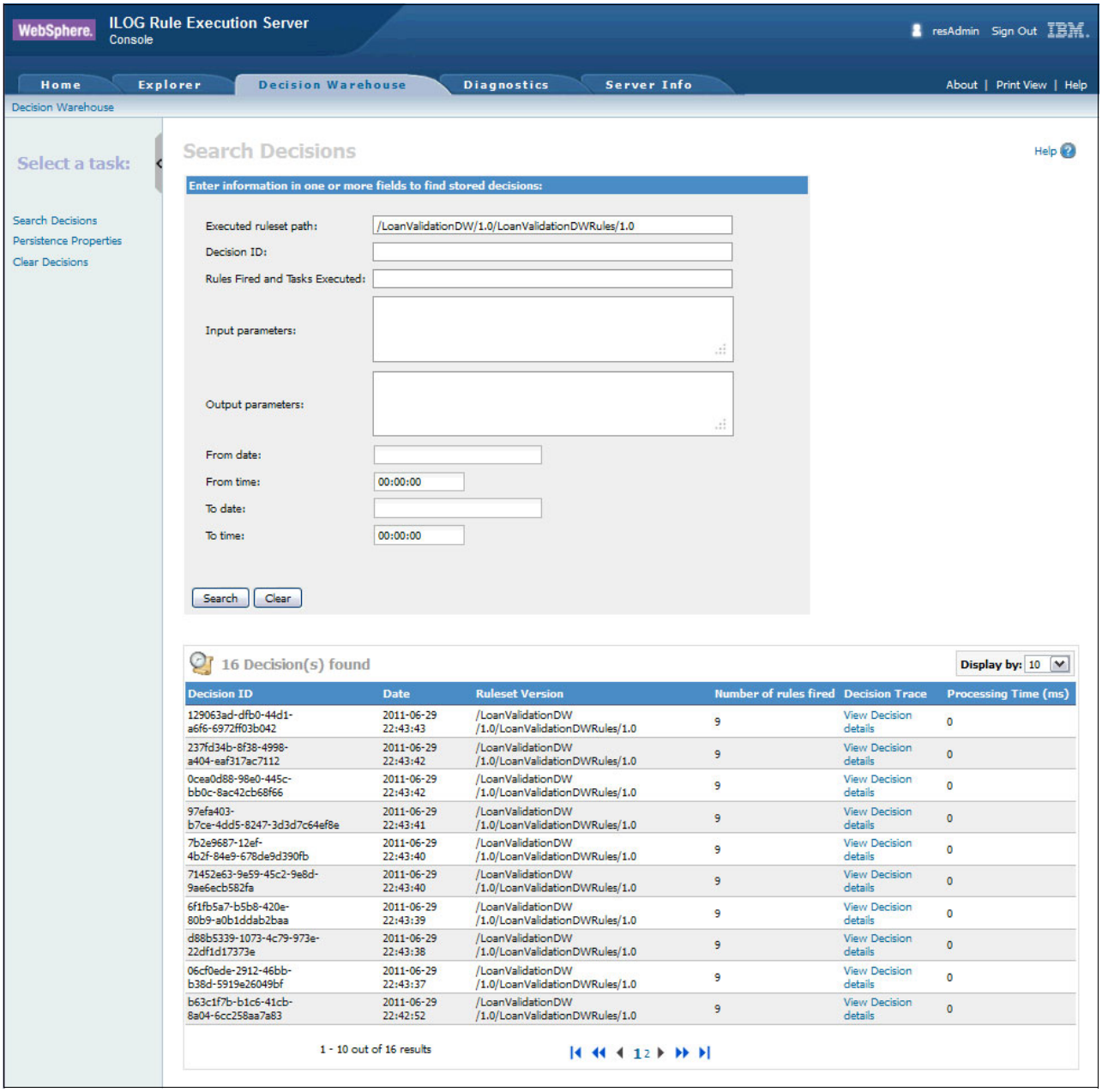


Figure 26 Decision Warehouse

Decision Warehouse offers the following features:

- Execution trace logging
- Input/output data
- Execution results
- Executed tasks
- Rules fired
- Queries
- Open API to connect third-party business intelligence (BI) tools

Figure 27 on page 25 shows an overview of the Decision Warehouse function.

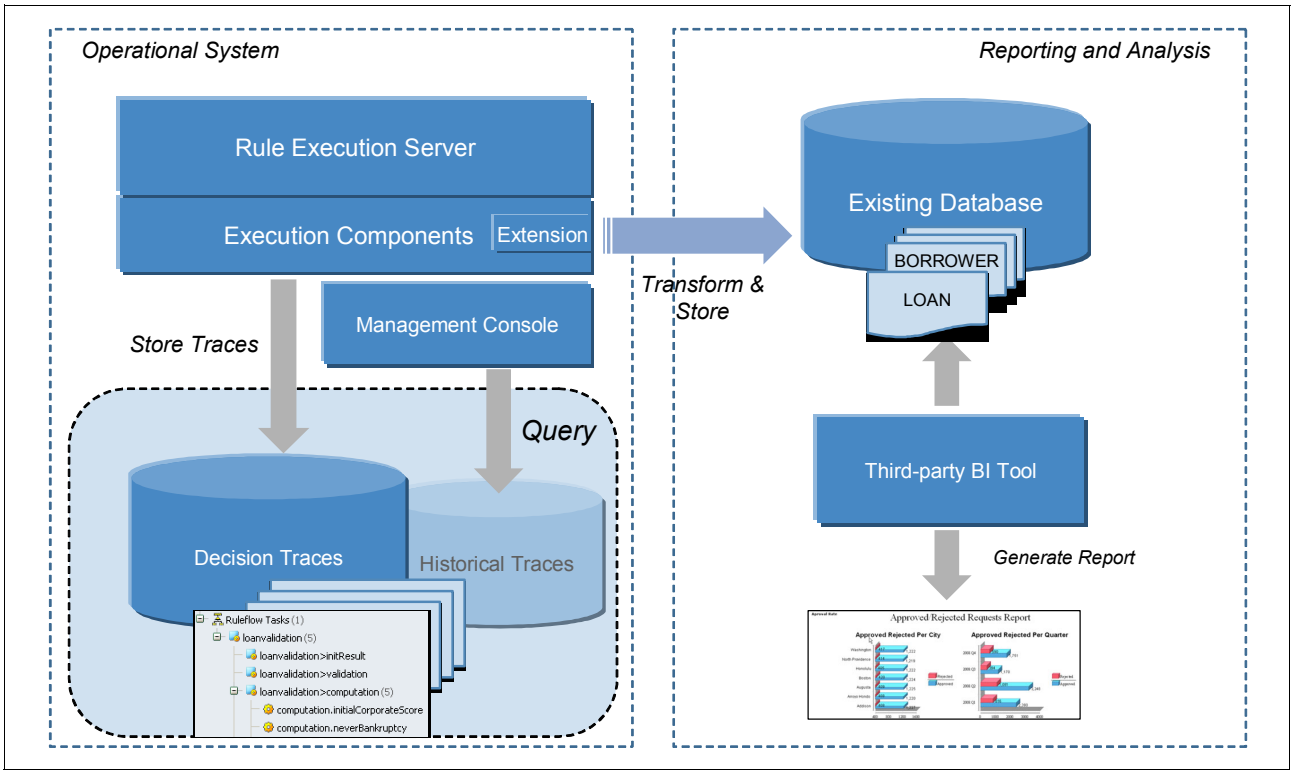


Figure 27 Decision Warehouse

What impact does the Decision Warehouse have on the Rule Execution

The performance of an application greatly depends on the following characteristics:

- The complexity of the object model:

  If the ruleset signature contains input/output parameters that have a large amount of data, it can take a long time to serialize and persist.

- The number of rules within the ruleset:

  Performance degrades as the number of rules fired increases.

- The level of detail that you want to capture:

  The more execution details that you want to capture, the longer it takes to complete the execution.

In general, the performance of the Decision Warehouse depends on these factors:

- The execution trace generation performance
- The serialization performance, which divides into two principal cases for the input/output parameter serialization to BOM (depending on the size of the ruleset parameters) and the serialization of the execution trace, which depends on the ruleset characteristics (the number of rules, number of rules fired, and so on)
- The performance of the database server and the network that is used to access it

How can we optimize the Decision Warehouse

Three major methods exist to optimize the Decision Warehouse:

- The Decision Warehouse is configurable at a ruleset base with filter properties. You can choose the information to save. For example, configurable parameters are available to specify the ruleset properties to optimize performance. You can choose which execution information you want to capture from this list:
  - Executed ruleset path
  - Decision ID
  - Rules fired
  - Tasks executed
  - Input/output parameters
  - Execution duration
  - Tasks not executed
  - Rules not fired
  - All the names of the rules/tasks
  - Number of rules/tasks executed
  - Custom execution output
- You can choose through a list of filters (as ruleset properties) what to save in the Decision Warehouse. To get in the execution trace, input and output parameters, and execution events (tasks and rules executed), set the properties that are shown in Example 2.

Example 2 Ruleset properties

```
monitoring.filters=INFO_EXECUTION_EVENTS=true,INFO_INPUT_PARAMETERS=true,
INFO_OUTPUT_PARAMETERS= true, INFO_RULESET_PROPERTIES=false,
,INFO_INET_ADDRESS=true,INFO_EXECUTION_DURATION=true,INFO_TOTAL_RULES_FIRED=fal
se,INFO_TOTAL_TASKS_EXECUTED=false, INFO_TOTAL_RULES_NOT_FIRED=false,
INFO_TOTAL_TASKS_NOT_EXECUTED=false, INFO_RULES_NOT_FIRED=false,
INFO_TASKS_NOT_EXECUTED=false, INFO_EXECUTION_DURATION=false, INFO_RULES=false,
INFO_TASKS=false, INFO_SYSTEM_PROPERTIES=false, INFO_WORKING_MEMORY=false,
INFO_BOUND_OBJECT_BY_RULE=false,
```

- The performance of the Decision Warehouse depends on the database. For example, you must optimize the database to handle character large object (CLOB) data, not in a generic way, but for the specific case of the ruleset.
  - The Decision Warehouse is customizable. You can customize it further through the Decision Warehouse extension point for better execution performance. You can define how data can be persisted or queried through the extension point. See the Decision Warehouse customization sample at the following website:

    http://publib.boulder.ibm.com/infocenter/brjrules/v7r1/topic/com.ibm.websphere.ilog.jrules.doc/Content/Business_Rules/Documentation/_pubskel/JRules/ps_JRules_Global1777.html
  - You can also implement your own asynchronous logging of the data that is captured with the Decision Warehouse feature. The product contains a sample, which shows the reference architecture for using the extension point. An asynchronous version of decision warehouse for WebSphere Application Server V7 is available at the following website:

    http://www-01.ibm.com/support/docview.wss?uid=swg21433167/

A section in the documentation describes the ways to optimize execution performance when using Decision Warehouse. See *Optimizing Decision Warehouse* at the following website:

http://publib.boulder.ibm.com/infocenter/brjrules/v7r1/topic/com.ibm.websphere.ilog.jrules.doc/Content/Business_Rules/Documentation/_pubskel/JRules/ps_JRules_Global733.html

Enhancing the performance of Rule Team Server

Designed to enhance productivity, WebSphere ILOG Rule Team Server (Figure 28) brings unsurpassed levels of usability and sophistication to business rule management. Distributed business teams collaborate with a powerful, easy-to-use web environment to create, manage, and deploy business rules.

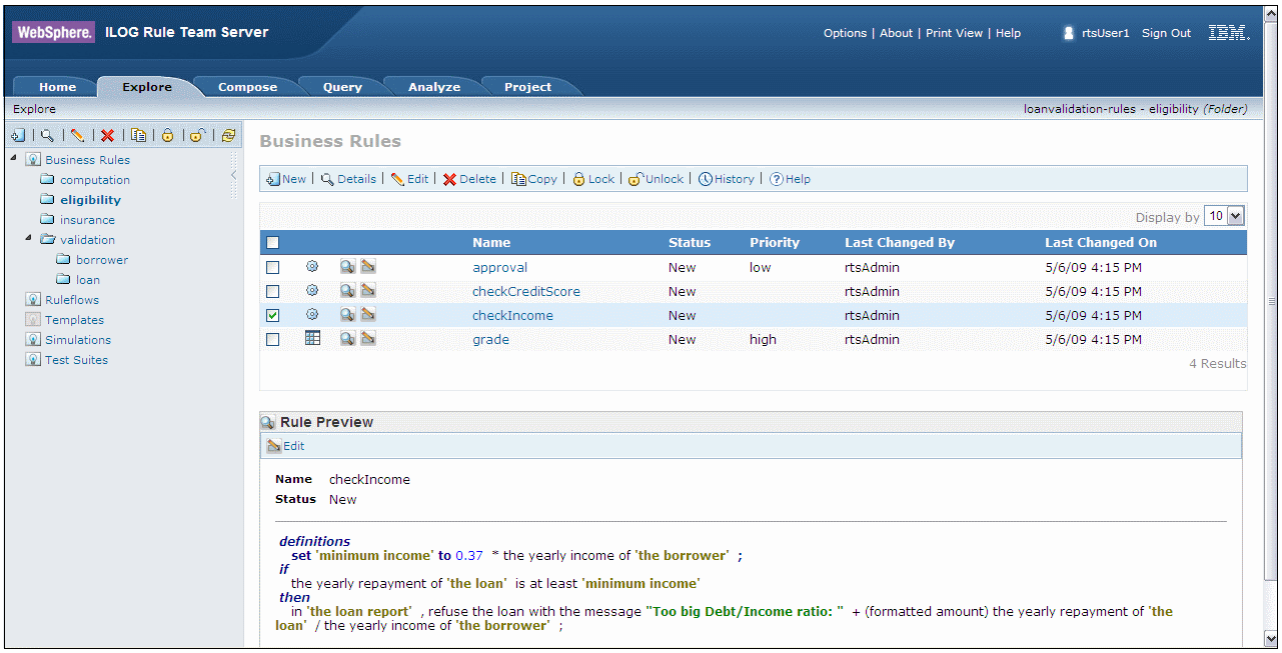


Figure 28 WebSphere ILOG Rule Team Server

Manage rules across the enterprise with confidence

Rule Team Server is easy to learn, use, and deploy. Business users can manage rules quickly and securely, saving time and money. Audit reporting and access to a central rule repository give users the peace of mind that they need to focus on other tasks, keeping them ahead in a competitive business environment.

With WebSphere ILOG Rule Team Server, you can perform these functions:

- Author and edit rules using a natural language and customizable business vocabulary. Rules can be expressed in graphical formats, such as decision tables and decision trees.
- Simplify rule projects through Smart Views (user-configurable navigation panes), filters, and reports.
- Manage collaboration with role-based permissions, access controls, and customizable rule metadata properties.

- Facilitate rule maintenance with templates, point-and-click editors, error checking, and versioning.
- Ensure rule quality with customizable queries, rule analysis, and visual comparisons of changes between versions.
- Add powerful testing and simulation capabilities using Rule Team Server's integration with WebSphere ILOG Decision Validation Services.
- Extend rule management across the enterprise through Rule Team Server's integration with WebSphere ILOG Rule Solutions for Office.

General guidelines and considerations on Rule Team Server

Rule Team Server makes heavy use of CPU, memory, and database resources. To ensure scalability, you must set up the hardware architecture to support usage requirements. Although it is difficult to give precise recommendations due to the variety of possible configurations, this section can help you to understand where possible bottlenecks can occur.

What is the memory footprint of Rule Team Server

Rule Team Server is designed to be accessed by several users at the same time. Each time that a user accesses the Rule Team Server URL, an HTTP session is created on the server and filled with several objects, including the current IlrSession to connect to Rule Team Server, several Java Beans to support the model of the JavaServer Faces (JSF) components that are used in the interface, and several web components (guided editor and decision table editor). The memory footprint that is required for a session is difficult to estimate, because it depends on many factors, including what actions are performed through the graphical user interface (GUI), the virtual machine (VM), and the structure of the repository.

However, we can provide a specific example. After browsing a repository with 10,000 rules, expanding and collapsing nodes in the rule explorer, previewing rules, editing one rule, and generating a ruleset (with the IRL already cached), the memory footprint of a session is around 10 MB.

During the ruleset generation, around 150 MB of short-lived objects are created, which can cause issues if many users are performing this operation at the same time.

Ruleset generation and semantic queries are the major tasks that consume memory.

A ruleset generation is performed by these tasks:

- Deployment to Rule Execution Server
- RuleApp generation
- Rule Analysis features, such as consistency checking and the completeness report
- TestSuite and Simulation execution

The semantic queries are queries, which contain at least one of the following sentences:

- May apply when
- May become applicable when
- May lead to a state where
- Modifies the value of
- Uses the phrase
- Uses the value of

Hints

If you think that the number of users connected simultaneously to a single server will not fit into the memory that is allocated for a single VM, deploy Rule Team Server to a cluster of servers and use load-balancing so that HTTP sessions are dispatched according to the server's available memory.

Another parameter to consider when dealing with memory is the number and size of the vocabularies that are used in a Rule Team Server repository. The vocabulary of a given project is stored in memory the first time that it is loaded and is shared across sessions. By default, vocabularies are stored in a pool of 10 instances. After 10 instances, other vocabularies are soft-referenced so that the VM can free the memory, if needed. The same pooling algorithm applies to instances of BOMs and variable providers.

The semantic queries are slower than other queries that can be transformed as SQL queries, so you must not use a semantic query to create a smart view. Otherwise, Rule Team Server calls this semantic query each time that the "explore" tab is clicked.

What is the guidance on CPU usage for Rule Team Server

According to the number of users and which operations they frequently perform when using Rule Team Server, the number of CPUs can be critical. Parsing a Business Action Language (BAL) rule or a decision table is an atomic operation that takes a lock on the vocabulary instance to be used. In addition, as test results have revealed, most of the long-running operations (ruleset generation, reporting, and so forth) make heavy use of the CPU. Therefore, it means that if many users are accessing the same project at the same time, browsing the repository, or running long operations such as ruleset generation, the shared vocabulary might become a bottleneck. This potential reinforces the need to deploy Rule Team Server on a cluster.

Hints for maintaining good database performance for Rule Team Server

Most Rule Team Server operations perform SQL queries.

Hints

Because certain queries can be time-consuming, deploy the application server and the database on separate machines to force the distribution of the load between Java computing and database access. This approach improves the user experience when multiple users connect to the repository.

Set the Java Database Connectivity (JDBC) connection pool of the data source to a number that is equivalent to the maximum number of users that will simultaneously connect to Rule Team Server. If the pool is too small, users might get connection timeouts when their sessions try to access the database.

Set the statement cache size in the WebSphere Application Server data source properties section to 50 or 100 instead of 10. Rule Team Server uses a great number of separate SQL requests. The performance impact is around 10% when you generate a ruleset or when you have several users using Rule Team Server.

Enhancing the performance of Decision Validation Services

Figure 29 shows an overview of the Decision Validation Services architecture.

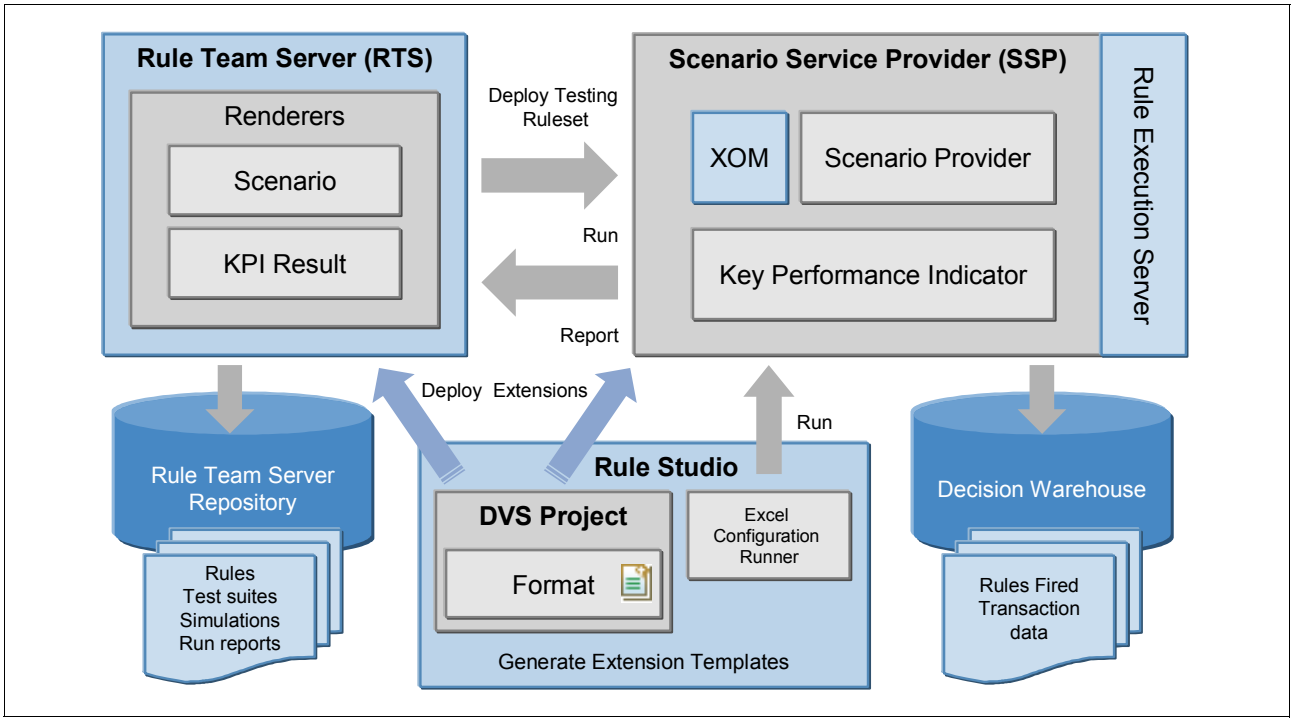


Figure 29 Decision Validation Services

Decision Validation Services offers the following major features:

- Predefined ruleset testing (Rule Team Server)
- Business impact simulation (Rule Team Server)
- Scenario configuration and customization (Rule Studio)
- Audit - Decision Warehouse (Rule Execution Server)

Suggestions for using Decision Validation Services

Executing a test suite or simulation can be a long operation, depending on the number of rules and the number of scenarios to execute. You perform these key steps during the operation:

1. Generate a ruleapp archive with the set of rules to test.
2. Deploy this ruleapp to a Rule Execution Server console.
3. Trigger the execution of the ruleapp on the Scenario Service Provider, which will pass the call to the XU.
4. Get the report.

Most of the time that is consumed on Rule Team Server takes place during ruleset generation. For a ruleset of 10,000 business rules, if this ruleset generation is a second ruleset generation (for example, if the IRL is already cached), it will take 10 seconds (six minutes for first ruleset generation). The rest of the time (almost two minutes, in our case) is taken in the Rule Execution Server (Scenario Service Provider/XU), parsing the generated archive and executing the scenarios.

Hints

Run Rule Execution Server and Rule Team Server on two separate machines, so that when Rule Execution Server consumes several resources executing a test suite or simulation, Rule Team Server resources (CPU and memory) are not affected.

Setting Rule Execution Server

Each time that a test suite execution is triggered from Rule Team Server, a new ruleapp is created, deployed, and parsed on the Rule Execution Server. Therefore, the parsing cost on Rule Execution Server cannot be avoided in the context of Decision Validation Services. From the tests performed, a 10,000-rule ruleset took almost all of the 768 MB of RAM allocated for the VM. Without any further configuration, a second execution launched just after the first execution will require a new item in the XU pool, provoking an out-of-memory situation for the VM hosting the Rule Execution Server.

Therefore, use these steps:

- Dedicate a Rule Execution Server instance to Decision Validation Services. Do not try to use a production Rule Execution Server for Decision Validation Services tests or simulations.
- Set the size of the XU pool[1] according to the total available memory/maximum amount of memory of a ruleset. For example, if you have one GB allocated for your VM and are executing a ruleset that takes 300 MB of memory, the size of the pool for the XU must be lower than or equal to four. The size of the pool must not be under two, because a test suite runs two rulesets, the ruleset to test and the ruleset that contains the tests.
- Set the size of the Scenario Service Provider pool[2] to the same size as the XU pool. Therefore, the XU pool (and thus the memory of the VM) will not be overloaded uselessly, nor will there be a bottleneck for test suite/simulation executions.
- If you have a huge volume of scenarios, the report options usage has a great impact on performance, because we use the Decision Warehouse to store the execution results.

Decision Warehouse optimization

To reduce the size of the Decision Warehouse trace in Decision Validation Services, see the following website:

`http://www-01.ibm.com/support/docview.wss?uid=swg21438208`

Enhancing the performance of the Rule Engine execution

The Rule Engine provides the following functions:

- Reads its rules dynamically at run time
- Works on objects that it knows using pattern matching
- Keeps track of changes to the objects that it knows
- Invokes the execution of the rules, which is also called *firing* the rules

Figure 30 on page 32 shows an overview of the Rule Engine.

[1] Set in `ra.xml` of the Scenario Service Provider, or configured in the Java EE Connector Architecture (JCA) connection factory of the application server

[2] JRULES_JOBS_POOL_SIZE in the `web.xml` of the Scenario Service Provider web archive (WAR) file

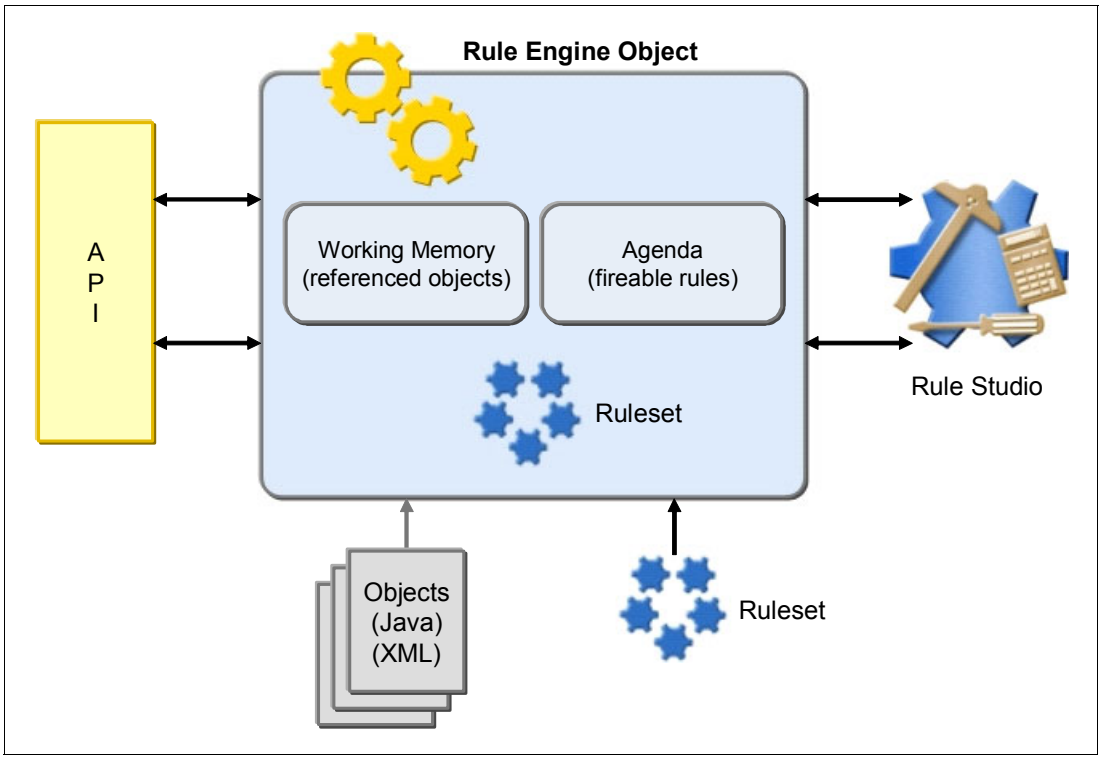


Figure 30 Rule Engine

The Rule Engine processes the rules that are provided to it. It evaluates the rules against the application objects and executes rule actions when appropriate. The Rule Engine can operate on ruleset parameters, local ruleset variables, and native objects that are inserted into the working memory (*references*). Rules are evaluated according to the ruleflows that are contained in the ruleset archive. If there is no ruleflow, all the rules in the ruleset are evaluated.

How relevant are academic benchmarks

The following issues exist with academic benchmarks:

- Small number of rules (fewer than 50).
- Test worst-case RETE inference and agenda usage, which is rarely encountered with well-designed applications.
- Implement solutions to combinatorial search problems, typically better implemented using a constraint programming (CP) engine rather than a RETE engine.
- No usage of common patterns, such as ruleflow and Sequential execution.
- The solutions are not verified, meaning that correctness might be sacrificed for speed, which is particularly the case for Waltz and Waltzdb.
- The benchmarks are easy to manipulate, because every vendor ports from the original OPS5 source code into their own rule language, where they take advantage of their product's features.

For reference, we describe the three most famous academic benchmarks:

- Manners

  Manners is based on a depth-first search solution to the problem of finding an acceptable seating arrangement for guests at a dinner party. The seating protocol ensures that each guest is seated next to someone of the opposite sex who shares at least one hobby.

- Waltz

  Waltz was developed at Columbia University. It is an expert system that was designed to aid in the three-dimensional interpretation of a two-dimensional line drawing. It does so by labeling all lines in the scene based on constraint propagation. Only scenes containing junctions composed of two and three lines are permitted. The knowledge that Waltz uses is embedded in the rules. The constraint propagation consists of 17 rules that irrevocably assign labels to lines based on the labels that already exist. Additionally, there are four rules that establish the initial labels.

- Waltzdb

  Waltzdb was developed at the University of Texas at Austin. It is a more general version of the Waltz program. Waltzdb is designed so that it can be easily adapted to support junctions of four, five, or six lines. The method that was used in solving the labeling problem is a version of the algorithm that was described by Winston [Winston, 1984]. The key difference between the problem solving technique used in Waltz and Waltzdb is that Waltzdb uses a database of legal line labels that are applied to the junctions in a constrained manner. In Waltz, the constraints are enforced by constant tests within the rules. The input data for Waltz is a set of lines defined by Cartesian coordinate pairs.

Conclusions

If performance is one of your major buying criteria, you are encouraged to build a proof-of-concept (POC) set of rules and data and to verify Rule Engine performance in your own environment. It is impossible to extrapolate from published academic benchmark results to your running application, with your rules and data, deployed to your OS and hardware. In addition, a POC also allows you to evaluate the BRMS from as many angles as possible, spanning ease of use, business user accessibility, support and professional services, performance, deployment platforms, scalability, and so forth.

How can we decide on Rule Engine execution modes

You can choose an execution mode for each rule task in your ruleflow. By default, a rule task is assigned the RetePlus execution mode. To achieve optimal performance, you might need to choose another execution mode that is better adapted for the rules in a particular rule task.

Table 1 on page 34 shows which mode to choose for each case.

Table 1 Quick view of execution mode selection

Choose this mode	In this case
RetePlus (the default mode)	All included semantically Stateful application Rule chaining Useful in the case of many objects and limited changes
Sequential	Covers most of the customer cases Many rules, few objects; has limitations Highly efficient in multi-threaded environment
Fastpath	Rules implementing a decision structure, many objects Might have longer compilation but faster at run time Highly efficient in multi-threaded environment

To determine which execution mode to use on a rule task, analyze the structure of the rules in the rule task, and what type of processing they perform. To make the best choice, you need to answer the following questions:

- What types of objects are used by your rules?
- What is the impact of rule actions?
- What sort of tests do you find in rule conditions?
- What priorities have you set on your rules?
- Who will modify the rules?
- What are your performance goals?

What types of objects are used by your rules

Depending on the types of objects that are acted on by your rules, the execution mode that you choose differs.

Working memory objects or ruleset parameters

If the objects that you use in your rules are passed with ruleset parameters, use the Sequential or Fastpath execution mode. If they are objects that you inserted into a working memory, use the RetePlus or Fastpath execution mode.

Bindings

Bindings are heterogeneous when rules do not operate on the same classes. When bindings are heterogeneous, the rules might have condition parts of various heights, as shown in Table 2.

Table 2 Heterogeneous bindings

Rule name	Rule condition
Rule1	... when{A();B()} ...
Rule2	... when{A()} ...
Rule3	... when{B()} ...

If your rules define heterogeneous bindings on the working memory, use the RetePlus or Fastpath execution mode.

If your rules define heterogeneous bindings on the ruleset parameters, you can use all execution modes, including the Sequential execution mode.

Bindings are homogeneous when all of the rules operate on the same class (the same kind and number of objects), but introduce separate tests. See Table 3.

Table 3 Homogeneous bindings

Rule	Separate tests
Rule1	... when{Person(age == 12);} ...
Rule2	... when{Person(age > 20);} ...

If your rules define homogeneous bindings, you can use all execution modes, including the Sequential execution mode.

What is the impact of rule actions

Depending on the types of effects that are generated by your rules on execution, the execution mode that you choose differs.

Modifications to the working memory

If the actions of your rules manipulate working memory objects, with the use of the IRL keywords `insert`, `retract`, or `update`, you must use the RetePlus execution mode. Because these keywords entail modifications to the working memory, the Rule Engine reevaluates subsequent rules.

If you use another execution mode, the Rule Engine will not reevaluate subsequent rules after the modifications.

Rule chaining

When your rule actions trigger modifications in the working memory or the parameters, and when your rule actions perform pattern matching on objects that have no relationship, such as people and hobbies, chaining exists between your rules, for example:

```
SILVER LEVEL CUSTOMER, GREATER THAN $5000 purchase
promote to GOLD LEVEL
GOLD LEVEL CUSTOMER, GREATER THAN $5000 purchase
apply 25% discount
```

You can see the chaining between these two rules, because the rules perform pattern matching on two objects, customer and purchase, and because the second rule will modify the level attribute of the customer object.

If you need for rules to consider modifications that are made by other rules, either use the RetePlus execution mode and its rule chaining capability, or, if it is possible to clearly split up the rules that have to execute first and the other rules, you can use a sequence of rule tasks.

What sort of tests do you find in rule conditions

The execution mode differs depending on the types of conditions that you use in your rules.

Tests that require a working memory

If you have rule conditions that test the existence of an object or that gather items in a collection directly in working memory, with the IRL keywords `exists` or `collect`, without `in` or `from` constructs, use the RetePlus or Fastpath execution mode.

Regular pattern for tests

If the tests in rule conditions have the same pattern and order, such as the tests that are generated from decision tables, use the Fastpath execution mode.

If the order of tests in rule conditions is not regular, use the RetePlus or Sequential execution mode.

What priorities have you set on your rules

If you have set static priorities on your rules, you can use any algorithm. However, if you set dynamic priorities, that is, priorities that are defined as an expression, you must use the RetePlus execution mode. The dynamic priorities are deprecated.

Who will modify the business rules

If business rules are modified and managed by business users, it is less risky to avoid using rule chaining. The performance and the semantics of rule chaining are extremely sensitive to rule modification or addition.

In that case, as described in the rule chaining section, you can simulate the rule chaining by using a sequence of rule tasks.

What are your performance goals

At run time, we can divide the performance goals into three categories:

- Ruleset loading. If the ruleset loading is a key performance goal, consider:
  - Using RetePlus or Sequential
  - Fastpath is in general slower to load
  - The complexity of the ruleflow must be limited
- Execution performance depends on the type of business rules:
  - If your rules are homogeneous (decision table case), try Fastpath.
  - If you need rule chaining:
    - Use RetePlus, but limit the usage of the update flag on your BOM to the accurate method or attribute. This flag forces a reevaluation of the Rete Network each time that this method is called.
    - Use several ruletasks in your ruleflow.
- Concurrent executions:
  - If you know that you will have a great number of concurrent executions, choose Sequential or Fastpath.
  - RetePlus does not have the scalability of Sequential or Fastpath when the number of concurrent executions increases.
  - The Ruleflow complexity has a performance impact when the number of concurrent executions increases. This complexity depends on the number of subflows and on the number of ruletasks evaluated per execution.

You can use Table 4 on page 37 as a reference in making your decision when you choose an execution mode for a rule task.

Table 4 Choosing an execution mode

In your rule task	RetePlus	Sequential	Fastpath
Ruleset loading	***Quick***	***Quick but slower than RETE***	Significant (rule analysis + bytecode generation)
Multi-thread scalability	**Poor**	***Good***	***Good***
Limitations	**No**	Collect in WM and not/exists are limited, no dynamic priority	**No dynamic priority**
Working memory objects	***Yes***	**Yes but only recommended with homogeneous rule conditions**	***Yes***
Ruleset parameters	***Yes***	***Yes***	***Yes***
Rule chaining	***Yes***	***No***	***No***
Tests on existence or collection items directly in working memory	***Yes***	***Limited***	***Yes***
Shared test patterns	**Syntactic Homogeneous conditions**	***No (but test sharing option)***	**Semantic Homogeneous conditions**
Heterogeneous bindings	***Yes***	***No***	***Yes***
Dynamic priorities	***Yes***	***No***	***No***
Runtime rule selection that selects a few rules among many	***Good***	***Good***	Low performance if selection of few rules among many (because of full evaluation cost)
Numerous rules	***No***	***Yes***	***Yes***

Do you have tips and tricks for business rule applications

For many people, business rule software is a new technology, and they do not yet have an innate sense of the performance profile of an application that uses a rule engine. For example, is invoking a set of 1,000 rules faster or slower than a database call? Of course, the answer depends on a variety of circumstances. We offer several guidelines when dealing with performance issues in business rule applications.

For applications with stringent performance targets or service-level agreements (SLAs), consider continuous performance monitoring. By investing daily in a small amount of performance monitoring, you can avoid expensive refactoring due to performance issues that are found at the end of your development cycle. Continuous performance monitoring is particularly important for business rule applications where the relationship between the rules being authored and the code being invoked at run time is not as obvious as with a traditional single-programming-language procedural software system.

Consider the sequence of the required operations to invoke a ruleset using WebSphere ILOG JRules:

1. Ruleset archive is parsed, creating an IlrRuleset.
2. Instantiation of an IlrContext for the IlrRuleset.
3. Creation of input data (IN,INOUT parameters) for the ruleset.
4. Optionally, add objects to working memory.
5. Execute the ruleset.
6. Retrieve INOUT and OUT parameters.
7. Optionally retrieve objects from working memory.
8. Reset IlrContext for subsequent execution.

Operation 1 is costly, and for a large ruleset containing more than 10,000 rules, it can take several minutes. However, it is a one-time cost, because the resulting IlrRuleset is pooled for subsequent reuse. Rule Execution Server provides predefined caching of IlrRuleset and IlrContext.

Operation 3 is often slow, because this operation is where your code affects various expensive back-end systems, such as databases, to retrieve the data that is required by the Rule Engine. Caching within your code might be a useful optimization here.

The time that is required for Operation 5 depends on the details of the rules, ruleflows, and rule tasks within your ruleset. The rules typically invoke methods in your Java Executable Object Model (XOM). It is important to determine the amount of time that is spent within your code versus the time that is spent within the Rule Engine or within the rules.

The documentation of the product contains useful information around optimizing execution.

Do you have a Rule Engine performance cost breakdown

The performance cost for a WebSphere ILOG JRules application might look similar to Figure 31.

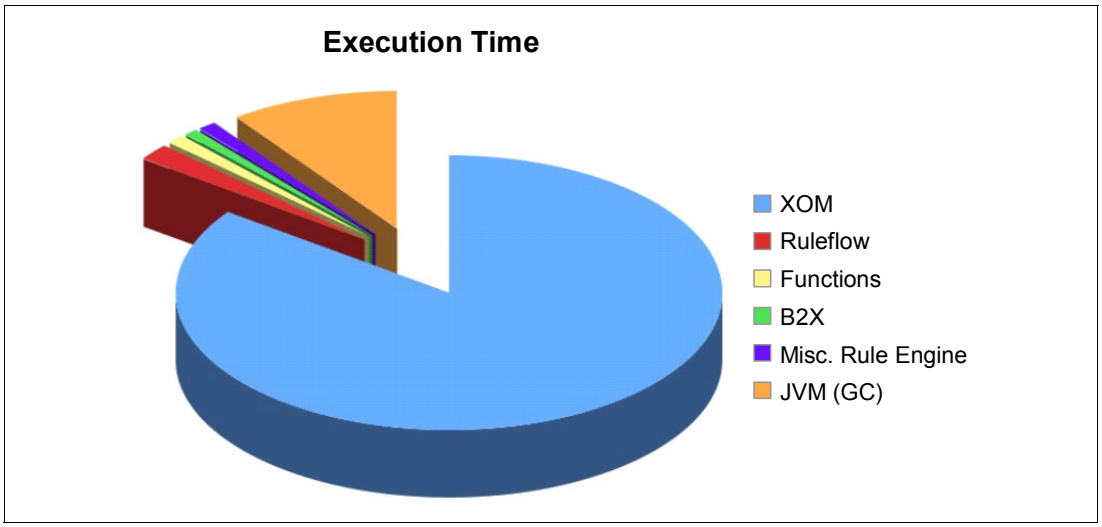


Figure 31 Execution time

The vast majority of application time is typically spent within the XOM code that is invoked by the rules. Getting a good understanding of which methods within your XOM code are going to be invoked by rules and roughly how frequently is critical. Also, consider the synchronization,

because if you have 10 rule engines concurrently calling a synchronized method within your XOM, your throughput is going to suffer.

Enhancing the performance of Rule Studio

Rule Studio is the development environment for business rule applications. Rule Studio is integrated into Eclipse. Developers can take advantage of this integration to develop their Java projects with rule projects. Developers and architects can therefore use Rule Studio to integrate and deploy the business rule application into their enterprise client application. Architects, developers, and business analysts use Rule Studio to design the business rule application and author, review, and debug business rules. Rule Studio also has tools for keeping the rules synchronized with the Rule Team Server repository.

Can Rule Studio build time be optimized

Use the following configuration to get better build performance:

1. Remove the build automatically on big rule projects.
2. Select **Rule Studio Menu** → **Windows** → **Preferences** → **Rule Studio** → **Build** (Figure 32). Clear the check mark from the following build characteristics:
   - Perform BOM to XOM checks during build
   - Perform IRL checks during build

Make these configuration changes provides better performance, but you lose the rule analysis of your ruleset.

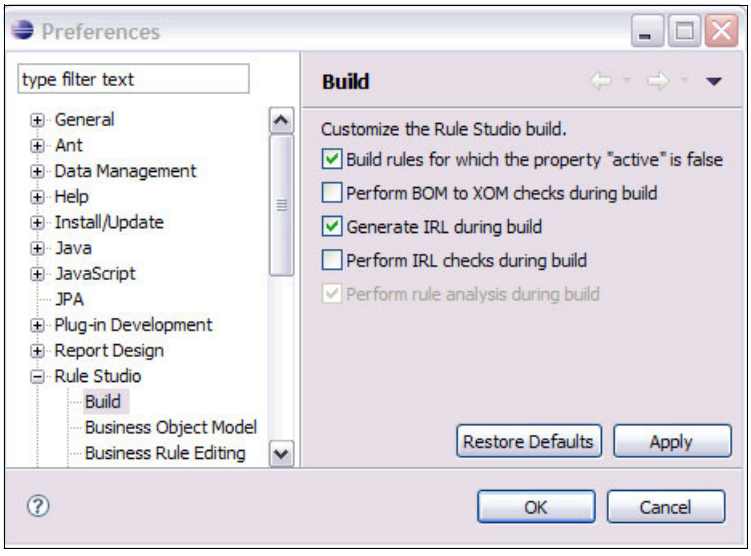


Figure 32 Rule Studio Preferences

Also, you can optimize the decision tables. Right-click a decision table to access its properties and remove the Table Checks on Overlap and Gap, as shown in Figure 33 on page 40.

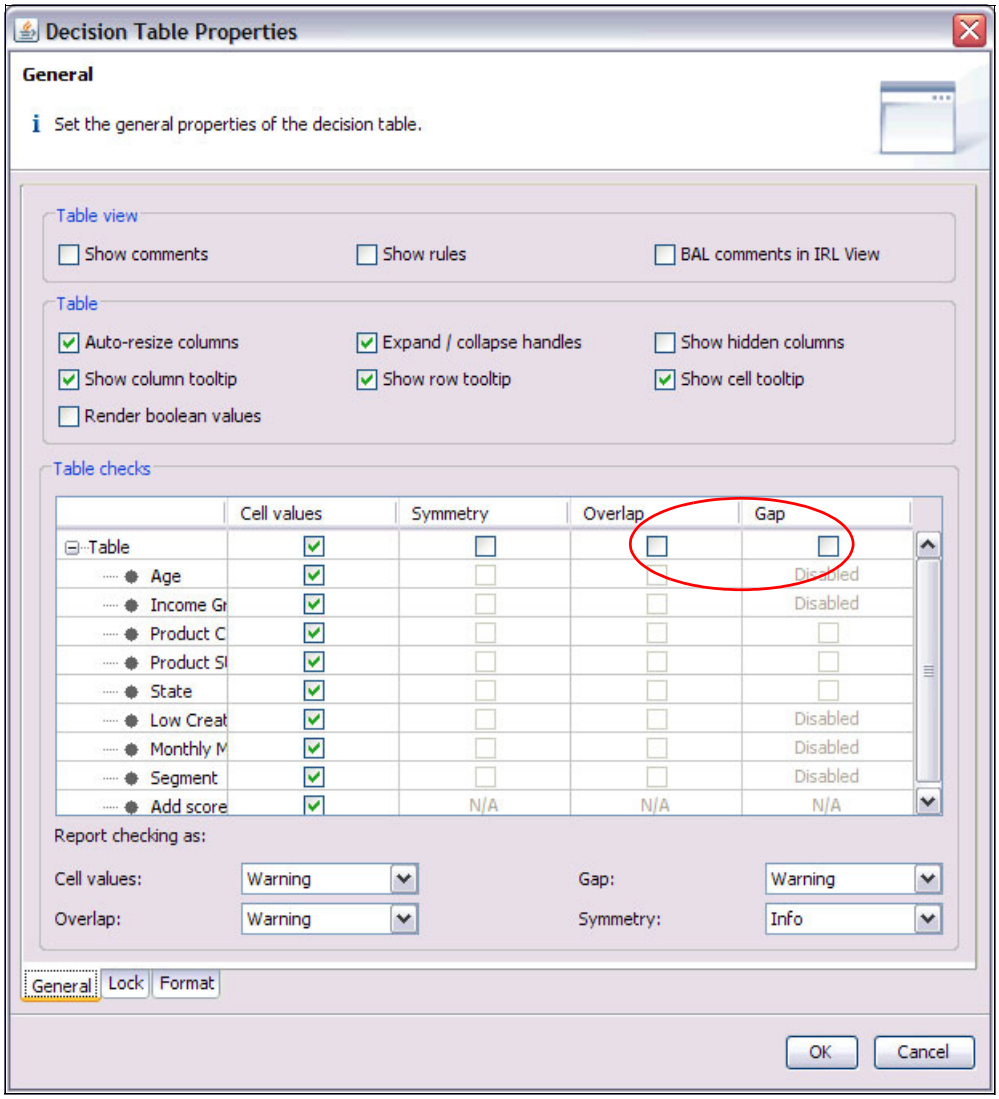


Figure 33 Decision Table Properties

Can Rule Studio build time have an impact on the productivity of the development

The time that it takes to build large rule projects can become a drain on productivity. We have worked hard in IBM WebSphere ILOG JRules V7.0 to improve the time that it takes to build large rule projects. The language team in particular is responsible for converting the business-level rule metaphors, such as Action Rules and Decision Tables, to executable IRL code. The language team has made great improvements in IBM WebSphere ILOG JRules V7, and everyone has been anxious to quantify them using actual client projects.

We performed detailed build performance analysis on six rule projects and one Java XOM project from a major international bank. All projects were rebuilt (“clean all” inside Rule Studio) and their build times were measured. Note that full builds, such as these builds, are not typically required during rule authoring, because we incrementally compile rule artifacts.

We aggregated the following artifact statistics for the six projects:

- Ruleflows: 300

- Variable sets: 5
- Decision tables: 363
- BOM classes: 1,860
- Vocabulary elements: 1,876
- Action rules: 4,206
- Functions: 2
- IRL rules: 2

As you can see in Figure 34, we achieved a two-fold performance increase across almost all the options.

Product differences: Note that Rule Analysis in WebSphere ILOG JRules V7.1 performs more semantic checking than Rule Analysis in WebSphere ILOG JRules V6.7, so it is not an exact comparison.

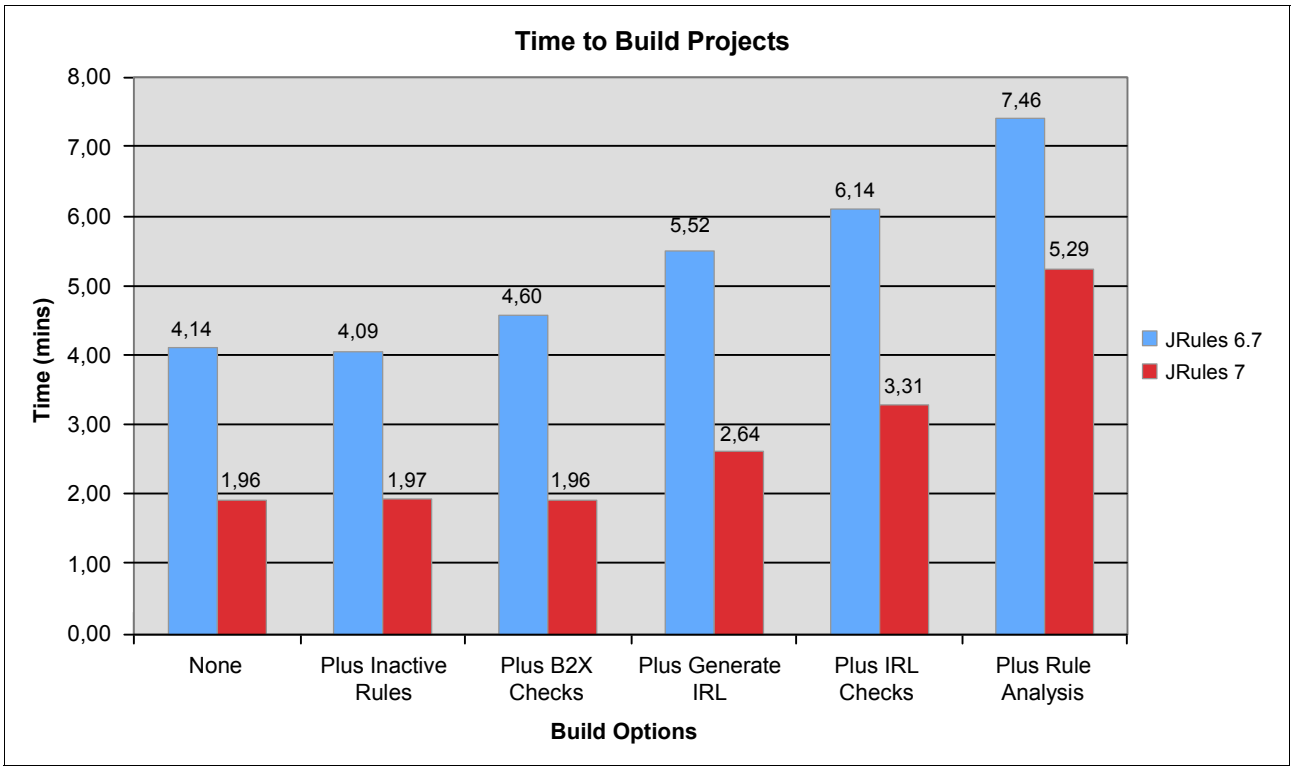


Figure 34 Project build times

We ran all the tests on a development machine running Java 5 with a 1 GB heap size. The results are good and can help make you more productive, but work continues to build performance, because it impacts your workflow and experience with the product.

Suggestions to optimize the synchronization

Our goal is the synchronization of Rule Studio and Rule Team Server and also to publish a project from Rule Studio to Rule Team Server.

Synchronization is a computation-intensive and memory-intensive process. Make sure that adequate heap size has been provided to both Rule Studio and Rule Team Server.

Synchronization is done at the top-package level. If the number of rules in a single top-level package is extremely large, this factor can be a significant bottleneck during synchronization. Restructure your rule project so that the rules are split among multiple top-level packages.

Suggestions about how to benchmark an application

Use the following suggestions to help you avoid performance issues in your rule projects:

- Understand, document, and communicate your performance objectives.
- Consider putting continuous performance measurement in place to provide you with a safety net as you modify your XOM and rules. You can see the impact of adding a synchronized method call to the XOM or adding a rule with a complex join more easily if you are receiving performance or throughput reports every morning.
- Performance test with realistic data. There is no point optimizing for a dummy data set that bears no resemblance to what you will see in production.
- Understand the frequency with which the methods in your XOM are invoked by rules, particularly if you perform lots of inferencing using the RETE algorithm.
- Spend time optimizing your XOM where profitable.
- Spend time getting to know the features of your profiler.
- The performance measurement needs to continue when the application is in production, because the number of rules might increase each year (from 10% to 50% increase each year).

Conclusion

This paper is intended to help you improve the performance of IBM WebSphere ILOG JRules V7.1. You can configure the BRMS modules in many ways to affect performance. In this paper, the focus for improving performance was to increase the speed of execution and to scale the application to meet the SLA. You can, more often than not, achieve a performance improvement by checking the default settings and assessing whether you can change parameters to get the best mix of speed and versatility for your application. This paper provides details about which parameters to check and the changes to make to them, depending on your type of application.

Every new release of the BRMS suite offers new features and new possibilities, which might affect performance. Ensure that you have an up-to-date version of this paper so that you make use of the performance improvements of these features.

The author who wrote this IBM Redpaper

Pierre-André Paumelle is the Performance Architect, IBM WebSphere ILOG Business Rule Management Systems, with over 15 years of experience designing and developing enterprise applications with C++ and Java (JEE). He worked side by side with clients for eight years as a consultant on ILOG products, especially those products in the BRMS product line. For the past five years, he has worked on the Enterprise Integration (BRMS) team as an architect and team lead. This team is in charge of Rules Execution Server, Decision Warehouse, and Decision Validation Services.

Contributors

Christophe Borde is a Product Marketing Manager for the IBM WebSphere ILOG Business Rule Management product line. Before joining the business rule team more than four years ago, Christophe worked as Product Manager for the ILOG visualization product line for over two years. Previously, he worked for ILOG UK as Technical Manager and Business Development Manager for Northern Europe. Before this position, Christophe held various positions within ILOG from Technical Support Engineer to Pre-sales Consultant and with Thomson CSF as a Software Engineer. Christophe earned a Master of Science in Computer Science and a Specialized Master in Marketing Management.

Albin Carpentier is a Software Architect on the IBM ILOG Business Rule Management Systems, with over six years of experience developing Java enterprise applications. He worked on the Enterprise Integration team as a developer in charge of management models and consoles. For the past two years, he has worked on the Rule Team Server team as a Software Architect, and participates actively in the integration of Decision Validation Services.

George Harley is a Senior IBM Developer with over a decade of experience designing and developing enterprise applications with C++, Java, XML, and model-driven technologies. George ran the Rule Execution Server benchmarks on zLinux.

Antoine Melki is a Software Architect with over a decade of experience developing and designing various parts of ILOG Business Rule Management System. He has been the major architect and team lead for ILOG BRMS Rule Team Server since its creation, four years ago. Rule Team Server is a highly scalable rule repository that is exposed through a web-based rule management environment for business users.

Daniel Selman is a Software Architect and a Team Lead for IBM WebSphere ILOG JRules Rule Studio, which is a rule development environment for technical users that is based on the Eclipse platform. Daniel has worked as an Architect and Product Manager on a variety of Java C, C++, and J2EE applications. Daniel blogs at:

https://www.ibm.com/developerworks/mydeveloperworks/blogs/brms/?lang=en

Antony Viaud is a Product Manager for WebSphere Business Rules Management System, responsible for rule development requirements, focusing on the Rule Studio and Rule Engine modules. Prior to this position, he was a Technical Account Manager with ILOG US, specialized in the Business Rules Management System for Financial Services.

Mark Wilkinson is ILOG Collateral Manager, IBM Software Group. His responsibilities include the management of print collateral relating to the IBM ILOG products. Before joining IBM in 2009, Mark worked as the Marketing Communications Writer and Newsletter Editor at ILOG S.A. for more than 11 years.

Yee Pin Yheng is a Product Manager for WebSphere Business Rules Management System for the last three years, responsible for enterprise integration requirements, focusing on Rule Execution Server functionalities, integration stacks with Java EE application servers, and Rule Team Server architecture. Prior to the position, he was a Staff Applications Developer in ILOG. He led the development effort of WebSphere ILOG JRules integration for various business process management (BPM) products, including WebSphere Process Server, MQ Workflow, and BEA WebLogic Integration.

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION 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. IBM 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-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

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

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

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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

This document REDP-4775-00 was created or updated on July 21, 2011.

Send us your comments in one of the following ways:
- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks