

Performance

1. Benchmarks and Configurations

RADRunner can run on a configuration of any size from a laptop to an application server cluster, to support the user base and performance required. It scales *linearly*, in that adding more hardware resources will result in a corresponding performance improvement. This can be seen from the figures below, which are based on those obtained at *IBM Solution Partnership Centre, Hursley* during scalability testing, and assume the use of the application server WebSphere version 4.0.4 with the relational database DB2 version 7.2 for persistence.

Here are some recommendations as to the sort of configurations to use in different circumstances, with guidelines as to the performance that can be expected from RADRunner.

1.1. Single-user/Workgroup

With the database, the servlet container, and the EJB container all running on a single-CPU laptop (866MHz Intel Pentium II, 512MB RAM), RADRunner could be expected to serve 12-13 requests per second, equivalent to about 45,000 requests per hour. In this configuration the application is CPU-bound.

This performance would meet the needs of a single user to do modelling. If the laptop was replaced with a PC of equivalent specification, which would have a faster hard disk, RADRunner would run fast enough to serve a workgroup using the application constantly.

1.2. SMB (Small to Medium Business)/Department

With the database running on one four-processor Wintel server machine and the servlet/EJB container running on another (700MHz Intel Pentium III, 2GB RAM, fast SCSI hard disk), RADRunner could be expected to serve 25-30 requests per second, equivalent to about 100,000 requests per hour [*based on figures taken during stress and load testing at IBM Solution Partnership Centre, Hursley*]. In this configuration the application is CPU-bound on the servlet/EJB container.

This performance would meet the needs of a small organisation, or departmental usage in a large organisation .

1.3. Enterprise

With the database running on one four-processor server machine and the servlet/EJB containers clustered across 3 such machines (700MHz Intel Pentium III, 2GB RAM, all writing their logs to a shared network-attached fast SCSI hard disk array), RADRunner could be expected to serve in the region of 70 requests per second, equivalent to about 250,000 requests per hour *[based on figures taken during stress and load testing at IBM Solution Partnership Centre, Hursley]*. In this configuration the application is CPU-bound on the database machine.

An alternative (more expensive but probably more robust) configuration is a single eight-processor IBM AIX server machine for both the database and the servlet/EJB container (700MHz PowerPC_RS64-IV, 32GB RAM, fast SCSI hard disk), with which RADRunner could again be expected to serve about 70 requests/second, equivalent to about 250,000 requests per hour *[based on figures taken during stress and load testing at IBM Solution Partnership Centre, Hursley]*. In this configuration the application is again CPU-bound.

This performance would meet the needs of a large organisation. See [Clustering RADRunner](#) for recommendations on how to set up such a configuration.

2. JSP Compilation

The most common perceived performance problem is that some pages seem to take a long time to respond. This often arises the first time that a user requests a Java Server Page (JSP) since the application server was last started, and is because the application server must compile the JSP before processing the request, resulting in a delay to the user while this takes place. This problem can be eliminated by pre-compiling all the JSPs in RADRunner before starting the application server.

In general it is necessary to consult your application server documentation to find out how to precompile all the JSPs in a particular application. However, with WebSphere on Windows platforms this can be done using the supplied utility *RADRunnerJspBatchCompiler.bat* (this can be found in the *WebSphere* directory of the RADRunner install package). To use this utility:

1. Install RADRunner
2. Make sure the WebSphere service (*IBM WS AdminServer 4.0*) is started
3. Run *RADRunnerJspBatchCompiler.bat*.

This will compile all the JSPs in RADRunner one by one. If JSP compilation was causing the delays to users, they will no longer manifest themselves.

3. Servlet Container Initialization

After a servlet container is started up, RADRunner will carry out some initialization the first time it receives a user request, to start a *Role Interaction Machine* (RIM) in the container. This is an automatic procedure.

However, the user that makes the initial request to a servlet container will perceive a short delay while this takes place. Although the delay will be short - typically no more than a second or two, depending on your configuration - a system administrator may wish to avoid any end-user experiencing it. This can be done by manually requesting either the login screen or the *Start RIM* screen from each servlet container in turn (in a [clustered configuration](#), there may be more than one servlet container).

4. Stateful Session Bean Initialization

RADRunner uses several stateful session Enterprise Java Beans (EJBs) to manage a user's session. When these are created, the initialization method passes in an object (for example, a user or a Role).

By default, the J2EE specification mandates that the object passed in should be serialized (converted to a stream of bits) before passing it to the EJB, since the EJB may well run in a different JVM to the servlet which creates it. This serialization incurs a heavy performance overhead, particularly in CPU usage.

Hence, it is usually possible to instruct your application server to pass objects "by reference" into EJBs, rather than by serializing them. There is no problem with this as long as the same application server is running both your web application and your EJB application, which will usually be the case with RADRunner since both are supplied inside the same EAR file. Hence this approach is recommended to improve performance. Moreover, in general it is very easy to enable. For example:

- In JBoss, the application server will pass objects by reference by default, if both the web application and the EJB application are running inside the same JVM.
- In WebSphere, all that is required is to enable the property "Pass by reference" in the application server (or server group) used for RADRunner. This property can be found in the *Advanced* tab of the *Object Request Broker* service.

5. Tuning

In the general case, it is necessary to examine each machine involved in the configuration to see where the bottlenecks are, and how they can be removed:

1. Run a simple stress test of the application: create multiple Actors from within RADRunner, then use a tool to simulate simultaneous login and logoff to each of them. Freely available software for doing this includes [Microsoft Web Application Stress](#) (which only runs on Windows NT/2000/XP platforms - we use this tool ourselves) or [Apache JMeter](#) (which can be used on any platform).
2. While running the test, use hardware and operating system tools to monitor CPU, memory and hard disk access on each of the machines in the RADRunner configuration (servlet containers, EJB containers, HTTP proxies, HTTP servers, and data store - some of these may be the same machine). If CPU or memory peak consistently at more than 80% on one of these, or the hard disk is constantly busy, then that machine is a bottleneck.
3. Either duplicate the function of that machine across multiple machines (this can be done for everything except a Jini host which uses the JSK), tune it, or improve its specification - see below for more details.
4. Rerun the stress test - if RADRunner performance is still not acceptable, return to step 2.

The most likely bottlenecks are:

- CPU usage on the servlet and EJB container(s), as they respond to many concurrent requests, or perform an intensive operation such as world export
- CPU usage on the database host, as it reads from and writes to the object and transaction stores.

Memory and I/O do not seem to be bottlenecks in normal usage of RADRunner.

Some general advice about performance tuning is given below.

5.1. CPU Bottlenecks

CPU bottlenecks on the servlet and EJB containers can best be increased by switching to a [cluster configuration](#), or if you already have a cluster, increasing the number of nodes.

5.2. Memory Bottlenecks

Memory bottlenecks can often be dealt with by increasing the size of the heap assigned to the JVMs in question. In the case of the JSK, this can be done using in StartJini.bat (parameter `-Xms` specifies minimum heap size and `-Xmx` specifies maximum heap size).

In the case of a database, JSpaces, and servlet/ EJB containers, consult the product documentation - in WebSphere and DB2, for example, the relevant heap sizes can be set from the console.

5.3. I/O Bottlenecks

Performance

In general, I/O bottlenecks can be removed either by tuning the disk(s) in question, or by switching to a faster storage solution.

If RADRunner is set to log at DEBUG level, consider changing this to INFO - and if you already use INFO, consider tuning the logging to only show messages for certain classes (for details, see [RADRunner Log Output](#)).

If you use a RAID array, consider changing the level (e.g., from RAID 5 to RAID 0+1) to gain increased performance, although this may be at the expense of resilience.

It is generally possible to tune a database management system very closely for your hardware, and general database tuning is always valuable - for example, increase the amount of shared memory available to the instances in question. Remember that the databases concerned may include not only the database specified for RADRunner but, with an [application server cluster](#), also the database used for session EJB persistence.

In particular, DB2 can be set to manage a RAID array directly, using parameters such as NUM_IOCLEANERS (set this to 2 more than the number of spindles in the array) and NUM_IOSERVERS (set this to the number of CPUs used for the database instance). Here is an example setup procedure for the RADRunner table in a DB2 database on AIX:

1. From root, use *smit* or *smitty* to:
 - Create 3 logical volumes in a volume group (with *jfs2* as file system), one for each of the following 3 tablespaces:
 - *regular* data (standard columns)
 - *long* data (in particular, CLOB values)
 - *index* data
 - Create a mount point for each logical volume, via "Add An Enhanced Journal File System"
2. From root, create an empty file in each logical volume to use for the tablespace, setting its owner to be the DB2 administrator user (via *chown*) and its group to the corresponding group (via *chgrp*)
3. From the DB2 administrator user, run a script based on the following, but specific to your own system (you may wish to alter it in various ways, for instance to alter the numeric values or create more buffer pools):
 - catalog tcpip node loopnode remote localhost server
TCP/IP_service_name_for_DB2_instance
 - create database *database_name*
 - catalog db *database_name* as *remote_database_name* at node loopnode
 - connect to *database_name* user *user_name*
 - create regular tablespace *regular_tablespace* managed by database using (file '*regular_filename*' 40000) extentsize 64

- create long tablespace *long_tablespace* managed by database using (file '*long_filename*' 40000) extentsize 64
- create tablespace *index_tablespace* managed by database using (file '*index_filename*' 40000) extentsize 64
- create table RM_PLAYWRIGHT_OBJECT (
 - RMPO_WORLD VARCHAR(100) NOT NULL,
 - RMPO_ID VARCHAR(41) NOT NULL,
 - RMPO_NAME VARCHAR(100) NOT NULL,
 - RMPO_CLASS_ID VARCHAR(41),
 - RMPO_CLASS_NAME VARCHAR(100),
 - RMPO_SUPERCLASS_ID VARCHAR(41),
 - RMPO_SUPERCLASS_NAME VARCHAR(100),
 - RMPO_EMAIL VARCHAR(100),
 - RMPO_PASSWORD VARCHAR(100),
 - RMPO_TYPE VARCHAR(100) NOT NULL,
 - RMPO_PLAYWRIGHT CLOB(100000) NOT NULL,
 - RMPO_CREATED TIMESTAMP NOT NULL,
 - RMPO_LAST_UPDATED TIMESTAMP NOT NULL
)
 - in *regular_tablespace*
 - index in *index_tablespace*
 - long in *long_tablespace*
- update db cfg for *database_name* using NEWLOGPATH /db2log
- update db cfg for *database_name* using APPLHEAPSZ 512
- update db cfg for *database_name* using MAXAPPLS 256
- update db cfg for *database_name* using NUM_IOCLEANERS
number_of_disk_spindles_used_by_database_plus_2
- update db cfg for *database_name* using NUM_IOSERVERS
number_of_CPUs_used_by_database
- update db cfg for *database_name* using LOGFILSIZ 5000
- alter bufferpool IBMDEFAULTBP size 10000

You would then use *remote_database_name* when specifying to RADRunner which database to connect to.

6. Clustering RADRunner

RADRunner, like most J2EE applications, is suitable for running on a *cluster* of application servers - more than one servlet container and/or more than one EJB container. A particular machine can act as either or both of these container types.

Performance

Details of setting up such a configuration are specific to each application server, but here are some general points to take into consideration.

6.1. Session Affinity

RADRunner uses a type of EJB called *Stateful Session Beans*. As implied by the term *Stateful*, this means that the objects created during a particular user's session maintain persistent information, which may be accessed by each of the requests during that session.

In order to support these EJBs in a cluster, your application server group must be configured to support *session affinity* - in other words, if one request from a user goes to a particular container, then so will all subsequent requests made during that particular session. This allows the Stateful Session Beans created during a particular request also to be accessed by subsequent requests in the same session.

Session affinity is supported by all major application servers. In WebSphere, for instance, session affinity is enabled by default (if you wish also to cater for failover and/or recovery after timeout, enable *Session Persistence* and set the *Workload Management Selection Policy* in the Server Group to either "Random Prefer Local" or "Round Robin Prefer Local").

It is also common, particularly in large web farms, to use a dedicated hardware solution - a box which itself receives HTTP requests, looks at the session id of each request, and if there is an existing session, routes the request to the HTTP server or application server where the session was last processed. In this case, there is no need to specify session affinity in the application server itself.

6.2. Sharing Workload

If you wish each application server in a cluster to take an equal share of the workload, it is necessary to ensure that they are configured with the same parameters. For instance, a parameter such as *maximum number of threads* can dramatically affect the amount of work allocated to a particular application server in a cluster.

6.3. Background Role Instance Automation

Background Role instance automation is controlled in RADRunner by setting the system property *com.rolemodellers.rim.automationInterval* in each servlet container to the required number of milliseconds. This will result in a periodic process being carried out to automate all role instances known to the Boss user, either directly or via intermediary role instances .

Each servlet container in which this property is set to an integer larger than zero will run its own automation process. Hence, when setting up a cluster, a strategy should be chosen

priate to your configuration for best making use of the system resources available.

For example, if you wish to turn off background automation altogether, set the system property to 0 in all servlet containers. Alternatively, the property could be made non-zero in one container only - that container could then be reserved entirely for automation by setting up your HTTP proxy so that end-user requests all go to the other servlet containers. Another option is to enable automation in more than one servlet container in case one goes out of action for some reason - in this case, the automation interval could either be the same or different in the containers concerned.

6.4. Jini Stress Limiting

As discussed in [Benchmarks and Configurations](#), the free JSK from Sun is optimized to run very fast on low-powered systems, and not designed to support a configuration which allows very high throughput. When RADRunner is installed on an application server cluster, or even a single multi-processor machine with a high (or infinite) limit on the number of concurrent application server threads, it is advisable to use either the [database persistence option](#) or [JSpaces](#) with RADRunner.

However, RADRunner does include a feature which allows the throughput to Jini to be dynamically limited, in order to prevent the JSK crashing in situations of exceptionally high load. If you wish to use a clustered configuration with the JSK, it is possible to do so via automatically [Monitoring Request Load Level](#).