

SoftRater for Windows Load & Performance Testing

An Oracle White Paper
November 2008



SoftRater for Windows Load and Performance Testing

Abstract.....	3
introduction.....	3
Rating Overview	4
Overview of the oracle insurance insbridge rating and underwriting system.....	5
RateManager conceptual view	6
SoftRater conceptual view.....	7
SoftRater configuration	8
Challenges of Performing SoftRater Performance Testing.....	8
SoftRater Performance Testing methodology.....	9
Step 1: Choosing Programs Based on Heuristics of the Program	9
NOTE.....	9
Step 2: Generate Test Cases	10
Step 3: Application to Generate Rate Requests	10
Step 4: Configuration of the SoftRater Database	11
Step 5: Configuration of the SoftRater Server	12
Step 6: Testing Machine	12
Use case scenarios	13
Use Case #1: Cache Enabled Test.....	13
Use Case #2: Policies Per Second Per Line of Business	14
Use Case #3: Commercial / Large Volume Item Policies	16
Use Case #4: Enterprise Services Object Pooling	17
Use Case #5: Average Response Times.....	18
Summary	18

SoftRater for Windows Load and Performance Testing

ABSTRACT

Companies and IT organizations are faced with adopting and incorporating new features and functionality while maintaining high levels of performance from their business applications. The Oracle Insurance Insbridge SoftRater (SoftRater) engine was design to provide the next generation rating, rules and underwriting functional flexibility while maintaining the high levels of system performance and scalability that IT has become accustomed to.

This paper discusses and outlines scenarios that demonstrate the performance and scalability of the SoftRater engine. This should be used as a guide for the methods of analyzing and targeting real world performance metrics and also provide possible configuration options which might satisfy integration efforts.

INTRODUCTION

The SoftRater engine is a Service Oriented Architecture (“SOA”) application, meaning that when deployed and activated it accepts requests in the form of XML documents, processes the requests and then returns responses also in the form of XML documents. A single XML request document is considered a single service request. Transactional response measurements encompass the total time recorded for servicing the request. While conducting the system tests, native Insbridge request XML documents were used and no document transformation process was incorporated in the analysis.

RATING OVERVIEW

Most insurance rating is performed by multiple systems based on the particular environment. The SoftRater system allows the same package to be utilized in every environment without the need to duplicate work as shown in the diagram below.

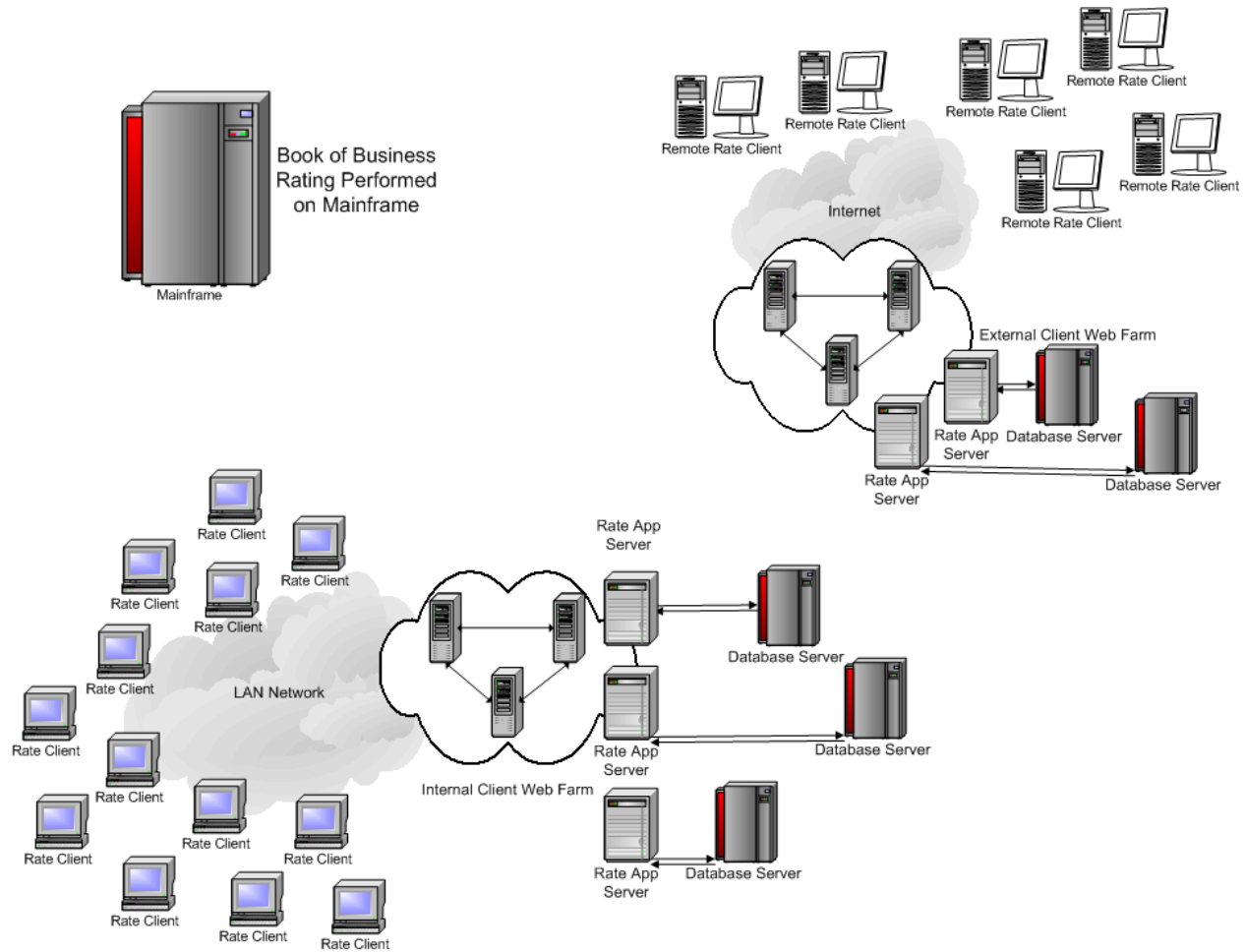


Figure 1. SoftRater Overview

OVERVIEW OF THE ORACLE INSURANCE INSBRIDGE RATING AND UNDERWRITING SYSTEM

The Oracle Insurance Insbridge Rating and Underwriting (OIIBRU) System is a suite of applications that provides companies unprecedented flexibility and control over the creation, management and execution of Pricing, Rating, Underwriting, Deployment and Real-time execution.

The three primary applications of the (IBRU) solution are RateManager the product definition and management system, PricingManager, the product analysis component and SoftRater the execution engine for servicing requests against a defined product.

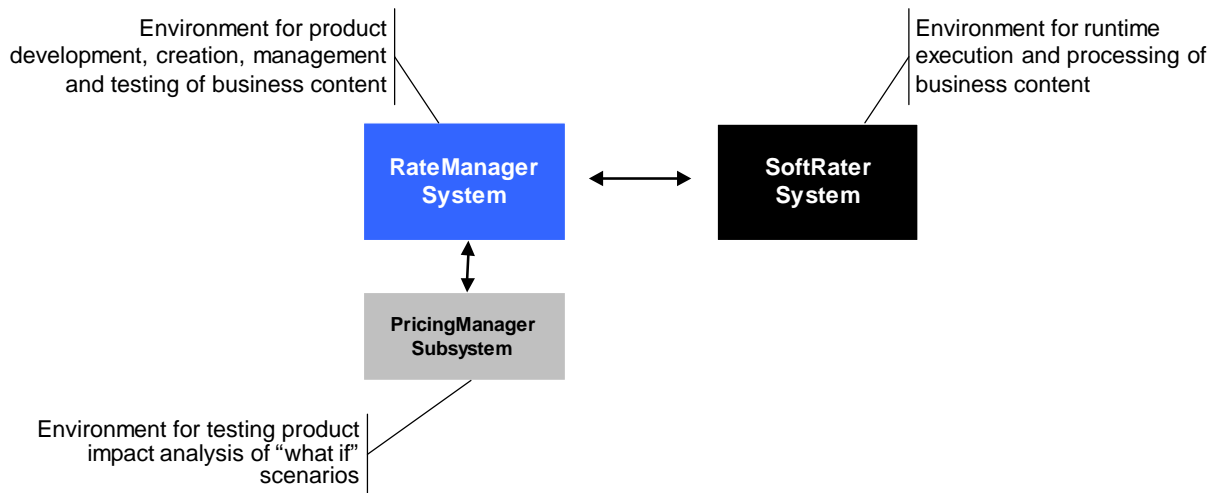


Figure 2. Oracle Insurance Insbridge Rating and Underwriting (OIIBRU) System Overview

RATEMANAGER CONCEPTUAL VIEW

RateManager is a component within the Oracle Insurance Insbridge Rating and Underwriting System that enables users to manage every aspect of the product definition and modification process, including rating and underwriting logic.

RateManager works in conjunction with the SoftRater engine as an integrated set of tools designed to flexibly manage rating and underwriting data and logic without programmer intervention.

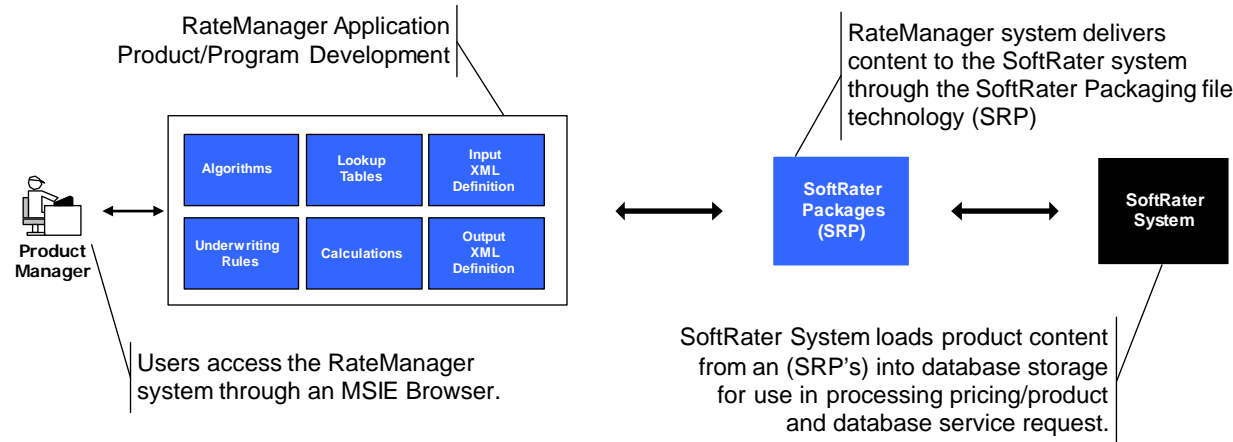


Figure 3. RateManager Conceptual View

SOFRATER CONCEPTUAL VIEW

SoftRater is the multi-platform rating engine component within the Oracle Insurance Insbridge Rating and Underwriting System that executes the user defined rating and underwriting instructions found in RateManager.

SoftRater works in conjunction with the RateManager as an integrated solution designed to manage rating and underwriting transactions.

With support for all major operating systems and databases on the market today, SoftRater enables organizations to leverage their investment in existing infrastructure while providing platform flexibility for the future.

The SoftRater supports request integration through:

- A Web Services (SOAP) interface
- Native Java
- .NET interface

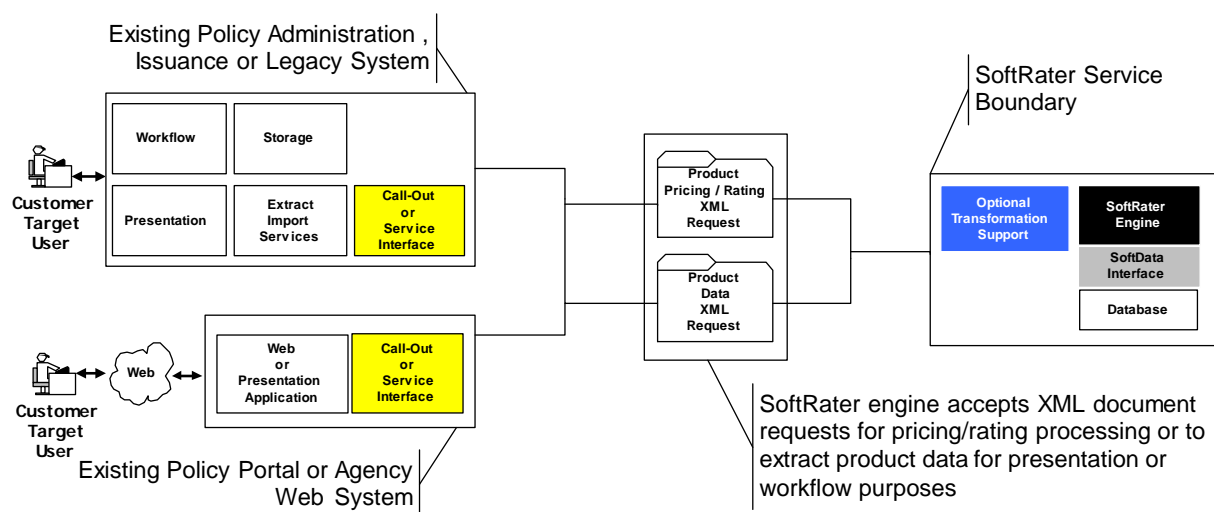


Figure 4. SoftRater Conceptual View

SOFRATER CONFIGURATION

SoftRater is designed to seamlessly handle multiple workloads for multiple environments.

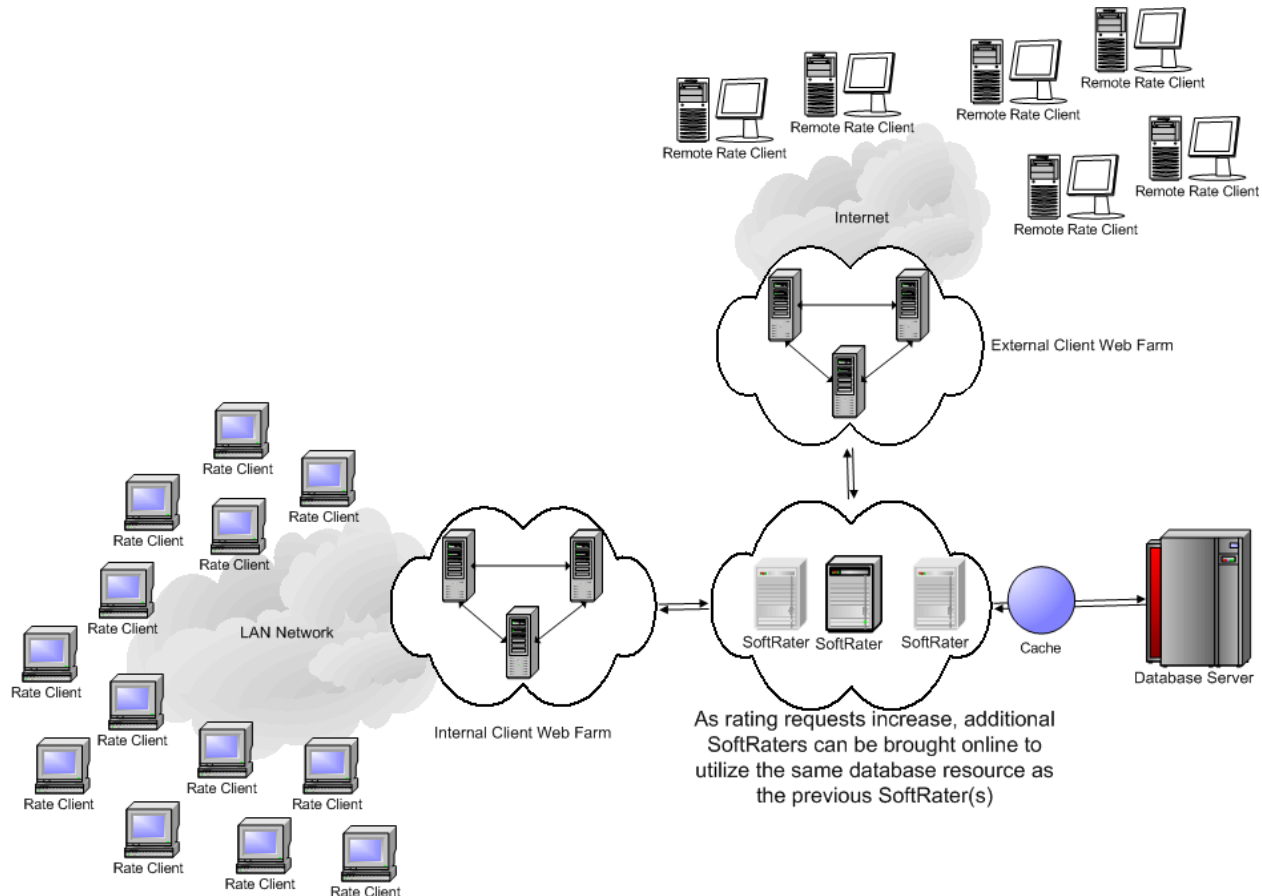


Figure 5. SoftRater Configuration

CHALLENGES OF PERFORMING SOFRATER PERFORMANCE TESTING

The primary challenge of testing the SoftRater engine was to test the engine and not the interconnecting systems like the database server, network, or testing machine. To isolate the performance of SoftRater, we utilized program caching on the SoftRater server to minimize database usage and offloaded the SoftRater database from the SoftRater server. On the testing machine, only necessary services were running.

Some of the other challenges included the actual efficiency of the testing application. The testing client, as described in the methodology section below, used a standard HTTP POST performed on the SoftRater Web

interface and recorded only the actual response time from the SoftRater server. The testing client then checked to see whether a valid response was returned from SoftRater.

The rating programs were chosen by heuristics that determined the complexity of each of the programs to simulate the real world business applications of SoftRater. Each specific line of business utilized was a real world production insurance program with the heuristics of each program playing a determining role in the decision.

SOFRATER PERFORMANCE TESTING METHODOLOGY

Based on customer makeup and to show the diverse nature of all SoftRater performance numbers, we choose to utilize six personal lines of business (“LOBs”) and three commercial LOBs. The personal lines of business were auto, home, fire, valuables, umbrella, and watercraft. The commercial lines of business used were disability, general liability, and commercial property.

Step 1: Choosing Programs Based on Heuristics of the Program

The decision on which programs per line to use was based on each program’s particular heuristics. The item heuristics were the number of inputs used, the number of results, the number of mapped variables, the number of calculated variables, and the number of algorithms as shown in the table below.

Table 1 Personal LOBs

Line of Business	# OF INPUTS	# OF RESULTS	# MAPPED VARS	# CALCULATED VARS	# ALGORITHMS
Auto	54	43	137	45	60
Home	61	48	163	23	69
Fire	68	68	93	57	95
Umbrella	17	21	56	4	20
Valuables	25	19	140	41	123
Watercraft	91	30	71	10	13

NOTE

The selected auto program contained a driver assignment scenario that consisted of 8 algorithms and 4 driver assignment macro steps.

Table 2 Commercial LOBs

Line of Business	# OF INPUTS	# OF RESULTS	# MAPPED VARS	# CALCULATED VARS	# ALGORITHMS
Disability	73	68	111	266	17
General Liability	47	71	50	11	25
Commercial Property	54	134	53	13	88

Step 2: Generate Test Cases

Utilizing an internal test case generator program, we were able to create 1,000 unique rate request XML files per line of business. Each rate request XML was considered a complete policy as dictated by the line of business for which it was generated. Some XML test files had multiple rates per policy, much like a real world environment. Some policy rate files had multiple items per request to simulate a real world environment.

Step 3: Application to Generate Rate Requests

We utilized an internally developed program to test the SoftRater engine. This application was designed to send a SOAP rating request to the designated SoftRater Web Services Interface; Verify the result; and calculate the amount of time taken to receive the response.

A test was not considered successful unless all results returned were passed. The application measured the system time before and after sending the request to the SoftRater server and then recorded the difference to retrieve the resulting per request time.

To maximize the testing machine resources, the decision was made not to utilize an individual client process per rate request because the resources needed by the testing machine would outweigh the accuracy of the testing. Each client process was configured with 50 input XML rate files that would be called sequentially as the client process could perform, taking in to account the time required to read each of the XML files from disk.

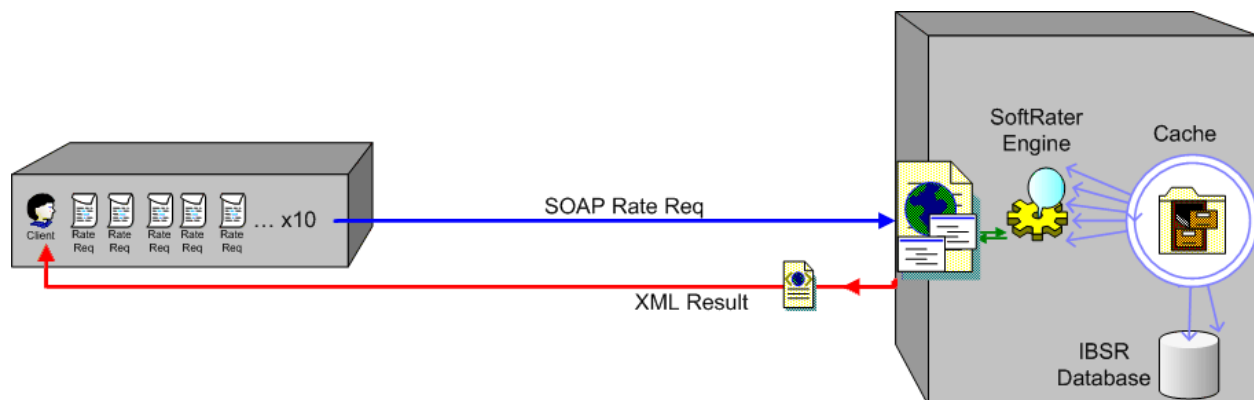


Figure 6. SoftRater Testing Routine

A single client would start by opening the first XML file and sending the XML in a SOAP request to the SoftRater Web Service on the SoftRater server. SoftRater would then check to see if the required data was loaded in the cache or whether a

database call was required. Once the SoftRater engine has the data, it would compute the result and send the resulting XML back to the client. The client would then parse the results to check whether the rate request had a PASSED result, record the time and then load the next rate file doing this until all the rate files had been processed.

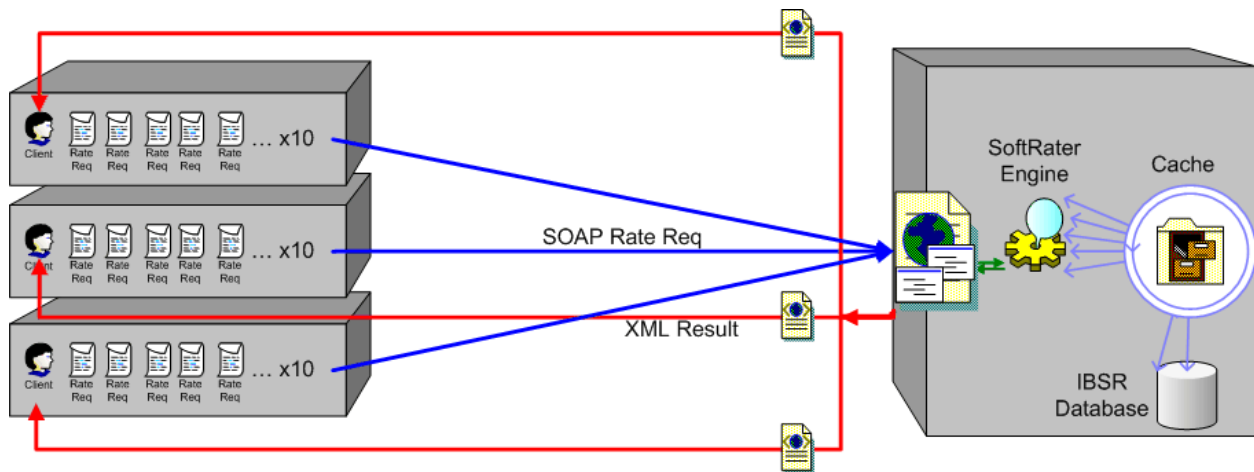


Figure 7. SoftRater Testing Routine

Additional tests were performed with more than one client process per test, usually with 10 or 20 client processes. A 40 client test was performed with the processes split between 2 separate testing machines in order to reduce client saturation caused by lack of resources on the testing servers.

During the testing period, the testing machine requirement was to never exceed 75% CPU utilization without labeling the test as invalid.

Also measured was the bandwidth of the network. The testing machine(s) and the SoftRater server was configured on the same network segment and total bandwidth utilized never exceeded 50%.

Step 4: Configuration of the SoftRater Database

Because of the SoftRater caching system, we chose to utilize a database on a separate machine from the engine. Although this might affect the initial performance, we were able to populate the SoftRater cache to minimize database usage unless where specifically required.

Step 5: Configuration of the SoftRater Server

The server was configured with dual Pentium III Xeon processors running at 1 GHz each, 1 GB of RAM, and an 18 GB SCSI U160 hard drive, using Windows 2000 Server Service Pack 4. No optimization was performed on the SoftRater server. The configuration utilized a default install of IBSS Version 3.5.06 as our SoftRater version with caching enabled and a 0% miss purge buffer. Unless otherwise noted in the tests below, no other optimizations were performed.

Step 6: Testing Machine

The testing machine was configured with a Pentium IV 1.8 GHz, 512 MB RAM, and a 20 GB IDE hard drive running Windows 2000 Server using Service Pack 4. The tests were started from the testing machine using a batch file to start the client processes.

USE CASE SCENARIOS

The use case testing scenarios listed below were each created with a specific goal of showing certain performance characteristics of SoftRater in a real world rating configuration.

Use Case #1: Cache Enabled Test

The first test performed was designed to show the improvement of rate requests over extended rating requests. The first rate request was rating against the SoftRater server starting without caching, subsequent requests were performed to demonstrate the improved performance of the SoftRater system.

Each rate request contained a specific number of rates per policy.

All times listed below are in milliseconds.

Table 3 Cache Enabled Test

Number of Items per Policy	Rate Request #1	#2	#3	#4	#5	Performance Gain
10	531	31	31	31	31	94.16
100	5138	241	241	341	241	95.31
500	24287	1353	1312	1413	1363	94.39
1000	48352	2705	2715	2865	2644	94.53
2000	95172	5559	5679	5599	5659	94.05

As can be seen in table 3 above, a 10 item policy rated the first time took 531 ms, while all subsequent requests were performed in 31 ms for an improvement of 94.16 percent. The 2000 item policy on first request took 95172 ms, while all subsequent requests of 2000 item policy took around 5600 ms.

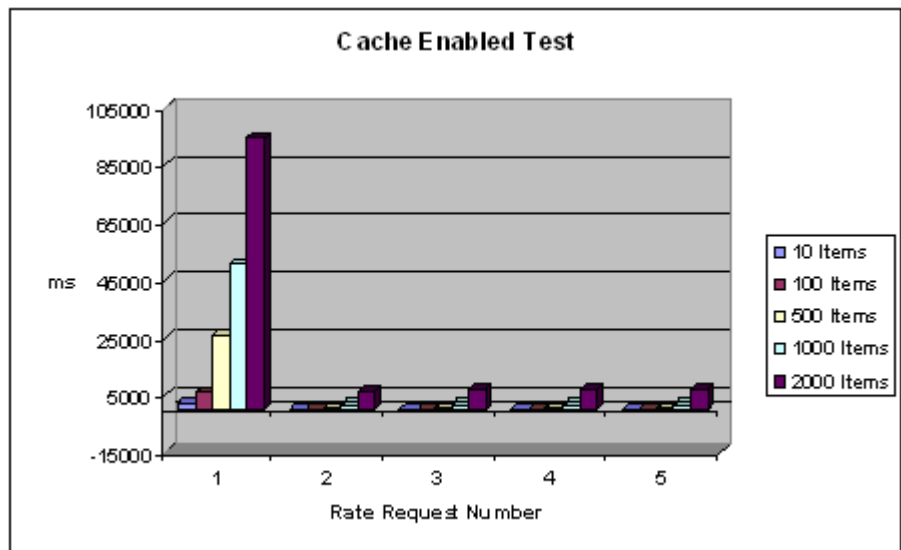


Figure 8. Cache Enabled Test Table

Use Case #2: Policies Per Second Per Line of Business

Using the above test methodology and programs, we were able to determine the total number of policies per line of business (“LOB”) that the SoftRater server could perform under load. Each LOB test was performed individually of the other LOBs using a single testing and SoftRater server. A number of initial tests were performed to determine the amount of rating requests that would provide a realistic basis for performance numbers. It was determined that 20 client processes should be used per LOB.

Each was configured to rate 50 policies with 20 clients for a total of 1,000 policies per LOB. The table shows the total time in milliseconds it took for each client process to perform 50 rate requests; The average of this total across the 20 clients; Average total time converted to seconds; divided by the number of rates performed (1,000) to arrive at the policies per second results shown below.

All times listed below are in milliseconds.

Table 4 Policies per Second per Line of Business

Client Number	Auto	Home	Fire	Valuables	Umbrella	Watercraft	General Liability	Commercial Property
1 ¹	49890	32252	34262	51497	28561	28611	19533	23047
2	56624	32295	36201	50859	29001	33329	25812	23094
3	58748	32515	35095	61155	30530	32922	26173	23922
4	57497	31966	36736	53187	29800	32345	26688	26546
5	59122	31997	37488	54312	29202	35113	27280	26091
6	55309	34358	34301	58454	31749	36251	26080	29736
7	56989	32516	34423	53641	29548	35580	27112	27983
8	55699	35362	37299	60112	31123	36202	26953	28986
9	51416	35202	37252	57109	29111	34845	26484	28860
10	58207	36183	32763	56953	31215	36014	27827	29860
11	59545	30659	33109	55185	30918	34829	27312	28892
12	57206	29652	36886	56625	29887	36734	26984	29767
13	58008	30624	36997	52749	29966	35907	27703	28562
14	58844	34107	36595	56594	30620	32889	26701	29096
15	58405	29731	35529	54720	30266	32420	27140	29299
16	58750	32410	34982	54531	30124	33641	27283	28704
17	58595	32612	35375	57907	30937	33845	26969	28951
18	58657	31362	35229	58656	30439	33706	27109	28704
19	57800	31624	34554	55973	30189	33191	26983	28924
20	56375	31137	35989	59251	29013	33378	26969	28641
Avg. Total ²	57084.3	32428.2	35553.25	55973.5	30109.95	34087.6	26554.75	27883.25
Avg. Total ³ seconds	57.0843	32.4282	35.55325	55.9735	30.10995	34.0876	26.55475	27.88325
Policies/sec ⁴	17.51795	30.83736	28.12682	17.8656	33.21161	29.33618	37.65805	35.86383

As can be seen in table 4, the number of policies per second varies across each LOB. This can be explained in the complexity of the policy rate request being performed. For example, private passenger auto executes driver assignment logic in addition to the coverages included, while valuable items included a number of separate valuables per policy.

¹ The total time in milliseconds it took for each client process to perform 50 rate requests

² The average of the total time per client process across all 20 clients (SUM/20=TOTAL)

³ The average total time then converted to seconds (TOTAL/1000=AVG_TOTAL)

⁴ The number of rates performed divided by the average total time (1000/AVG_TOTAL=POLICY_SEC)

The disability line of business was not included in the above list because of the overall difference in the number of amount of the lives included in each policy. More detail is provided in Use Case #3.

Use Case #3: Commercial / Large Volume Item Policies

The disability line of business is a special scenario because each policy could contain a wide variety of the number of lives per policy. For instance, some policies may only include 10 lives whereas a larger policy could include 20,000 (or more) lives. Based on these parameters, we asked current customers the general makeup of the overall percentages of the number of lives per policy request on average generated. These averages were then used to generate the 1000 policy rate requests. Below is the makeup of the percentages of the policy rate requests utilized for this test.

Test Policy Distribution Used for Table 5.

Table 5 Commercial / Large Volume Item Policies

Number of Lives per Policy	10-300	301-1000	1001-1999	2000+
Percentage Used	74	19	6	1

Because of the significant processing required by policies with large numbers of “lives”, it was determined that the optimum number of client rate processes that could be performed and still return accurate results was 5 and 10.

Table 6 Optimum Number Client Rate Processes

Client Number	Test 1	Test 2
1	403499	176107
2	422938	190377
3	427296	179984
4	422124	179374
5	408624	177203
6	379107	
7	429281	
8	423061	
9	408641	
10	426844	
Avg. Total	415141.5	180609
Avg. Total seconds	830.283	722.436
Policies/sec	1.204409	1.384206

The average number of lives rated per policy was 304.2.

Use Case #4: Enterprise Services Object Pooling

Providers strive for consistent rating performance so that client A and client B sending a request at the same time receive a response in relatively the same amount of time. Below are two graphs that show the average response times from 20 clients. One is shown without enterprise object pooling while the other demonstrates performance with enterprise object pooling.

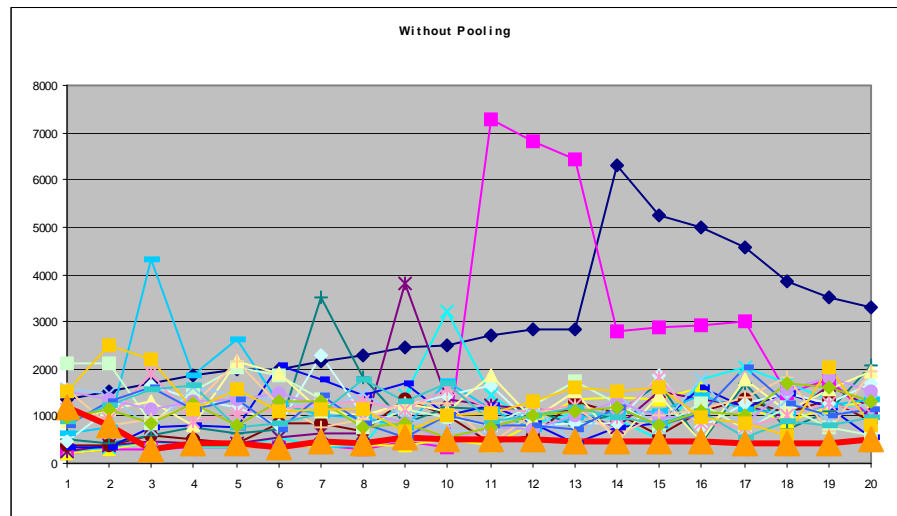


Figure 9. Enterprise Services Object Without Pooling

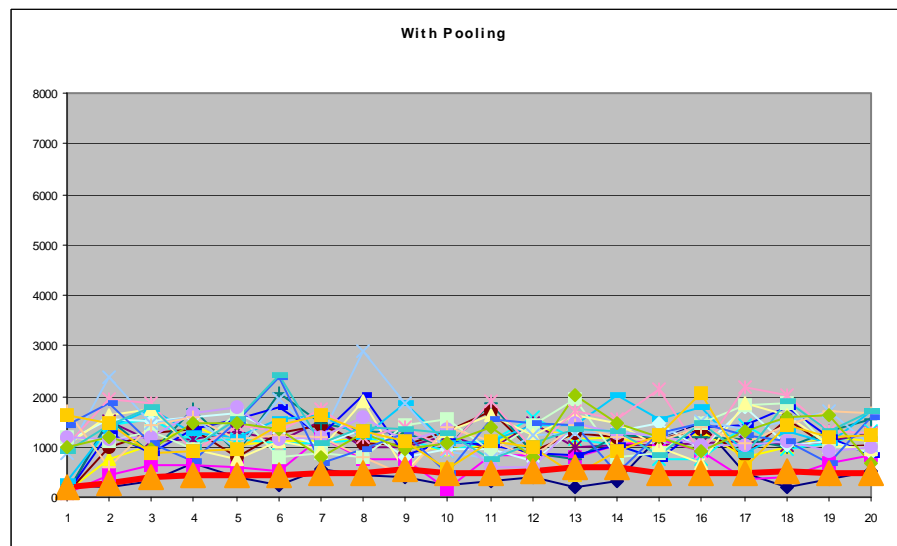


Figure 10. Enterprise Services Object With Pooling

What is interesting about the two graphs above is that even though the average response times across all lines of business was more consistent with object pooling enabled, the overall rate requests per second was quicker without pooling. There could be some mitigating factors in the reasons for this including the efficiency of the Microsoft .NET framework.

Use Case #5: Average Response Times

Another issue faced by most rating systems is the average response times provided over an extended load. The graph below displays the combined data from the performance testing in test #3 above with the average response times per client request per line of business. The graph shows moderate average variance for all the lines of business selected as part of the test.

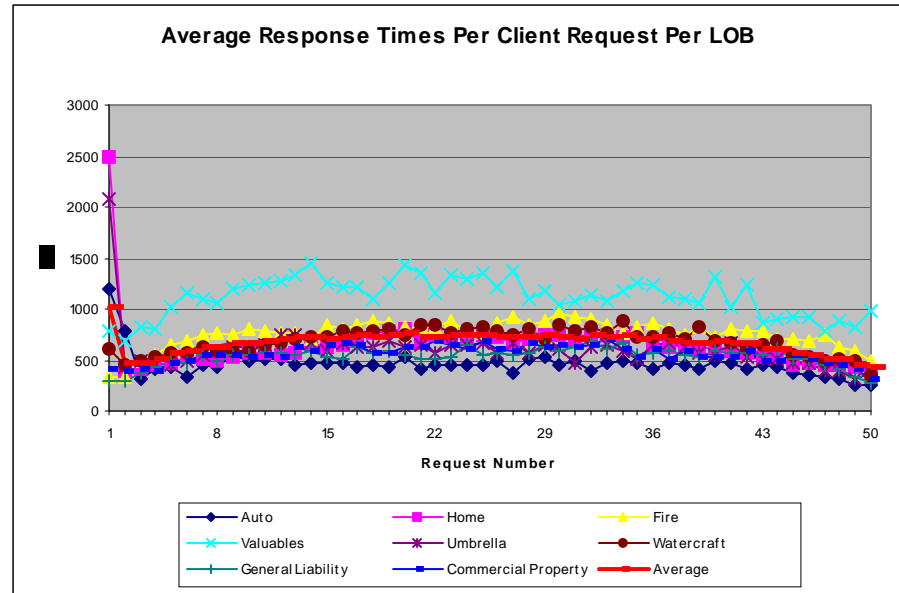


Figure 11. Average Response Times Table

SUMMARY

Metrics provided demonstrate that the SoftRater engine is a flexible and scalable rating and underwriting solution. As expected, the system shows scalable and variable operating efficiency by line of business, reflecting real world business use case scenarios.

The different testing methodologies used in this paper support aspects of the adaptability of the SoftRater engine to solving a variety of business and technical use case scenarios while continuing to provide consistent, superior execution, and performance.



White Paper Title SoftRater for Windows Load and Performance Testing
November 2008

Author:

Contributing Authors:

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:

Phone: +1.650.506.7000

Fax: +1.650.506.7200

oracle.com

Copyright © 2008, Oracle and/or its affiliates. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners. 0908