



# **The Retek Merchandising System V9.0 Performance Benchmark**

March - April, 2000

Prepared by:  
Richard Agnew (Online Resources)



# Table of Contents

<b>EXECUTIVE SUMMARY .....</b>	<b>3</b>
<b>INTRODUCTION .....</b>	<b>4</b>
<b>OBJECTIVES .....</b>	<b>4</b>
<b>HARDWARE CONFIGURATION.....</b>	<b>4</b>
<b>SOFTWARE CONFIGURATION.....</b>	<b>5</b>
<b>BENCHMARK SCOPE .....</b>	<b>5</b>
REPLENISHMENT PROCESSES .....	5
DATA VOLUMES .....	5
ACTIVE ITEM/LOCATION COMBINATIONS .....	5
<b>BENCHMARK RESULTS .....</b>	<b>6</b>
BASELINE SCENARIO .....	6
RESULT MATRIX .....	8
Oracle 8.1.5 (64 bit).....	8
Oracle 8.1.6 (32 bit).....	9
OBSERVATIONS.....	10
<b>DATABASE DETAILS .....</b>	<b>12</b>
TABLESPACE/DISK CONFIGURATION .....	12
REDO LOG CONFIGURATION .....	12
INSTANCE PARAMETERS .....	12
TABLE/INDEX PARTITIONING .....	13
BLOCK LEVEL TUNING .....	14
<b>TUNING ENHANCEMENTS .....</b>	<b>16</b>
PROGRAM MODIFICATIONS .....	16
RPLEXT - The Replenishment Extract module.....	16
RPLBLD - The Replenishment Order Build module.....	16
REQEXT - The Item Requisition Extract module .....	17
Program Modifications between Oracle versions 8.1.5 and 8.1.6 .....	17
RPLEXT - The Replenishment Extract module.....	17
RPLBLD - The Replenishment Order Build module.....	17
RPLEXT - The Replenishment Extract module.....	17
DATABASE MODIFICATIONS .....	18
Table/Index Partitioning .....	18
Block Level Tuning.....	18
Index Tuning.....	18
Database Modifications between Oracle version 8.1.5 and 8.1.6.....	19
<b>ORACLE ISSUES.....</b>	<b>19</b>
ORACLE 8.1.5 (64 BIT).....	19
ORACLE 8.1.6 (32 BIT).....	20
<b>APPENDIX A – DISK CONFIGURATION.....</b>	<b>21</b>
<b>APPENDIX B – SYSTEM PERFORMANCE NOTES.....</b>	<b>22</b>
CPU .....	22
MEMORY .....	22



## Executive Summary

Inventory replenishment is the process of providing the right product at the right place at the right time. To have all products in stock at all times is not technically difficult, but it is prohibitively expensive. To replenish inventory profitably, a business must weigh the costs of stock outages against the costs of holding inventory and of ordering more.

Merchandise Management for the Retail industry involves very large volumes of data, putting significant strain on the entire OLTP/DSS environment (hardware, software and applications). To meet the demands of high growth and complex business requirements, it is critical for the system to process transactions and execute required tasks in an acceptable timeframe. This document summarizes the technical configuration and results for this benchmark.

The Retek Merchandising System (RMS) Replenishment benchmark was performed at IBM Poughkeepsie, USA, from March 2000 through to the end of April 2000.

Retek and IBM sponsored the benchmark to establish whether the RS/6000 S80 is capable of running Retek's Replenishment application at a rate of 13.7 million transactions in less than 120 minutes. It was estimated that the evaluation of 13.7 million location/item combinations in 120 minutes would establish a competitive position for the IBM S80. The target of 13.7 million transactions in 120 minutes was exceeded during the benchmark exercise with 194,029 transactions per minute or 13.7 million items evaluated and replenished in **59** minutes. These numbers were accomplished using 200 stores, 5 Warehouses and 67,200 items per location.

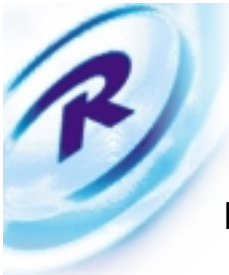
The benchmark was performed using commercially available hardware and software. A 220 Gigabyte Oracle 8i database was built using data that was representative of a large retailer's requirements. Under simulated but real-world conditions, replenishment transactions were executed against the RMS Application.

The replenishment process was broken down into 3 modules.

- The Replenishment Extract (rplext) module maintains optimum stock levels of replenished staple/fashion stock items by determining the Recommended Order Quantity for a location (ROQ).
- The Replenishment Order Build (rplbld) module builds the actual purchase orders after all the store and warehouse ROQs have been determined and written to the temporary order table. The Replenishment Order Build (rplbld) module creates a new order for each supplier.
- The Item Requisition Extract (reqext) module creates transfers for all SKU/store records for styles/staple SKUs that are being replenished, where the SKU/store is active and its stock category is Warehouse Stocked (W).

The parameters and results from the primary worst-case scenario of the benchmark appear in the following table:

Module	Item/Locations	Time (mins)
rplext	4,496,000	13:53
rplbld	N/A	7:36
reqext	9,280,000	38:04
<b>Total</b>	<b>13,776,000</b>	<b>59:33</b>



## Introduction

This document summarizes the technical configuration and the results of the RMS V9 replenishment performance benchmark that Retek/IBM performed. It details the objectives, the assumptions, the results, and the conclusions drawn from the benchmark.

## Objectives

The primary purpose of the Retek Merchandising System (RMS) Benchmark was to determine the performance and scalability of the RMS Application using massive data volumes and very large batch transaction rates typical for extremely large Retek customers.

The benchmark tested different hardware configurations under varying transaction loads for the batch environment. These tests were performed to test scalability and provide some insight into capacity planning, and to provide assistance to potential hardware vendors in determining hardware requirements to meet the client's RMS needs.

## Hardware Configuration

Retek performed the benchmark on an IBM RS/6000 model S80. It is a 64-bit symmetric multiprocessor system. Details of the configuration appear in the following table:

<b>RS/6000 S80</b>	
Maker	IBM
Model	RS/6000 - S80
Number of Processors	24
Processor Type	450 MHz PowerPC RS 64 III
RAM	64 GB
Hard Disk	1.7 TB
Operating System	AIX 4.3.3

The S80 was configured with 12 SSA Adapters with 32 MB fast write cache. Each adapter supported (2) loops, (8) disks per loop. A total of 160 – 9.1 gigabyte disk drives were used for the database although the S80 contained 192 hard drives. The 80 drives for the database (mirrored) were backed up on the remaining 32 drives.

We created 5 volume groups each containing 16 unique disks. Each of these disks was mirrored onto another unique disk. (giving us a total of 160 disks for the database)

“Appendix A” describes the disk configuration in detail.



## Software Configuration

The RMS software tested in this benchmark was the Version 9 pre-release. We installed and compiled all the RMS database objects. We only compiled the three Pro\*C programs necessary to complete the benchmark. (rplext, rplbld and reqext).

We started our testing on Oracle 8.1.5 (64bit) and completed the final benchmark tests on Oracle 8.1.6 (32bit). Due to problems encountered with 8.1.5 Pro\*C on AIX 4.3.3, we compiled the RMS programs using the 8.1.6 environment.

## Benchmark Scope

This section defines the scope of the benchmark. It identifies the processes evaluated and the characteristics of the data used.

### Replenishment Processes

The benchmark will measure performance for the following Replenishment processes:

- The Replenishment Extract (rplext) module maintains optimum stock levels of replenished staple/fashion stock items by determining the Recommended Order Quantity for a location (ROQ).
- The Replenishment Order Build (rplbld) module builds the actual purchase orders after all the store and warehouse ROQs have been determined and written to the temporary order table. The Replenishment Order Build (rplbld) module creates a new order for each supplier.
- The Item Requisition Extract (reqext) module replenishes items from warehouses to stores. It cycles through every item-store combination that is set to be reviewed on the current day, and calculates the quantity of the item that needs to be transferred to the store (if any).

### Data Volumes

The benchmark tested data for 200 stores and 5 warehouses. For the benchmark seven different workloads were defined, assuming a maximum of 67,200 SKUs to be evaluated per location for configuration IOR1. The following table lists the parameters of the different workloads defined as active SKUs to be evaluated per location during each replenishment run.

Module	IOR 1	IOR 2	IOR 3	IOR 4	IOR 5	IOR 6	IOR 7
Rplext	20,800	18,200	15,200	12,200	9,200	6,200	3,200
reqext	46,400	40,600	33,800	27,200	20,400	13,800	7,200

### Active Item/Location Combinations

The “Active Item/Location Combinations” for this benchmark can be defined as the percentage of Item/Location combinations that required the replenishment programs to either create a purchase order or a transfer. To be sure the benchmark would address the retailer’s real-world needs, Retek tested worst-case scenarios for the replenishment batch process. Retek set the target for this benchmark at 100% for all seven workloads.



## Benchmark Results

The results of the benchmark appear in this section as execution times for the different batch processes. Replenishment ran with the following baseline settings:

- We used the Min/Max Method
- Due Order Processing was not used
- Scaling was not used
- 9,280,000 SKU/Store Items set as Warehouse Replenished (100% Generating Transfers)
- 4,160,000 SKU/Store Items set as Cross-Dock Replenished (100% generating 232,000 Orders and 4,160,000 Allocations)
- 232,000 SKU/Warehouse Items for Vendor Replenishment (50% actually generated Orders)

Retek changed several key parameters from the baseline (that is, the most likely) scenario to measure their effect on execution times:

- Number of active Item/Location combinations (i.e., number of Item/Location combinations required to be evaluated)
- Number of threads to be run simultaneously on a given server
- Number of processors dedicated to RMS batch runs

By analyzing the effects of selectively changing these parameters, a retailer should be able to estimate accurate batch run-times under different circumstances.

### Baseline Scenario

The following table summarizes the characteristics of the most likely or “baseline scenario” (high water benchmark) and provides execution times for the Retek batch processes:

<b>Parameters</b>	<b>Scenario 1</b>
Item/Location Combinations	13,776,000
Active Item/Location	13,776,000
Percentage Active	100%
<b>Batch Process</b>	<b>Time (mins)</b>
Rplex	13:53
Rplbld	7:36
Reqext	38:04
<b>Total</b>	<b>59:33</b>

These results were the best that we obtained. We made numerous tuning modifications to the database prior to and when we upgraded from Oracle 8.1.5 (64 bit) to Oracle 8.1.6 (32 Bit). See the section “Modifications between 8.1.5 and 8.1.6” for more details.

Retek/IBM executed more than 80 separate replenishment benchmark runs to assist in estimating more accurately the hardware requirements for Retek clients.

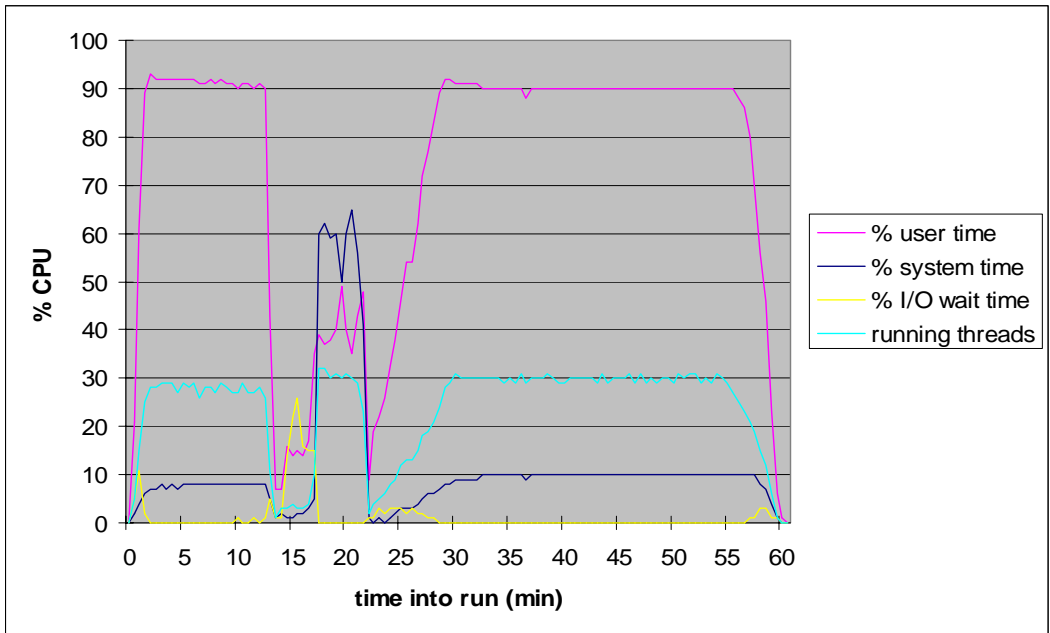
This baseline scenario was achieved on Oracle 8.1.6 (32 bit) with the following important runtime parameters:



Parameters		Scenario 1	
Item/Location Combinations		13,776,000	
Combinations Evaluated		13,776,000	
Percentage Evaluated		100%	
Batch Process	Threads	Sleep (secs)	Time (mins)
rplext	37	2	13:53
rplbld	33	0	7:36
reqext	33	8	38:04
<b>Total</b>		<b>59:33</b>	

We were able to reduce the overall runtime by increasing the total number of threads executing simultaneously for each module, and by reducing the amount of wait time between starting each thread. This was only possible by tuning the database to reduce contention. Initially due to the contention within the database we only ran a maximum of one thread per CPU, plus one.

The following graph was recorded during the run of the baseline scenario:



This graph illustrates a number of important points.

- The consistent smooth CPU profile achieved during the benchmark tests demonstrates the high User CPU utilization.
- The drop off after each module completes is rapid. This indicates that each thread is completing in approximately the same amount of time, this highlights that each thread has encountered little contention within the database.
- There is minimal I/O wait encountered throughout the entire process.
- The system CPU usage remains constant throughout each module's execution.



## Result Matrix

### Oracle 8.1.5 (64 bit)

The following matrix summarizes the initial results obtained running on Oracle 8.1.5 (64 bit)

CPU's	IOR 1	IOR 2	IOR 3	IOR 4	IOR 5	IOR 6	IOR 7
24	1:32:12	1:28:33	1:12:16	1:02:14	53:29	-	-
18	1:57:21	-	1:26:58	-	0:59:48	-	-
12	-	2:03:27	1:46:52	-	1:09:14	52:23	-
6	-	-	2:59:07	-	1:57:25	-	51:11

The following table summarizes the results obtained using 24 CPU's.

Parameters	IOR 1	IOR 2	IOR 3	IOR 4	IOR 5
Item/Location Combinations	13,776,000	13,776,000	13,776,000	13,776,000	13,776,000
Combinations Evaluated	13,776,000	12,054,000	10,045,000	8,077,000	6,068,000
Percentage Evaluated	100%	88%	73%	58%	44%
Batch Process Time	Mins	Mins	Mins	Mins	Mins
Rplex	19:00	17:06	14:46	12:25	9:56
Rplbld	6:12	5:52	5:11	4:17	2:40
Reqext	1:07:00	1:05:35	52:19	45:32	40:53
<b>Total</b>	<b>1:32:12</b>	<b>1:28:33</b>	<b>1:12:16</b>	<b>1:02:14</b>	<b>53:29</b>

The following table summarizes the results obtained using 18 CPU's.

Parameters	IOR 1	IOR 3	IOR 5
Item/Location Combinations	13,776,000	13,776,000	13,776,000
Combinations Evaluated	13,776,000	10,045,000	6,068,000
Percentage Evaluated	100%	73%	44%
Batch Process Time	Mins	Mins	Mins
Rplex	24:55	18:45	12:31
Rplbld	8:54	5:07	3:06
Reqext	1:23:32	1:03:06	44:11
<b>Total</b>	<b>1:57:21</b>	<b>1:26:58</b>	<b>0:59:48</b>

The following table summarizes the results obtained using 12 CPU's.

Parameters	IOR 2	IOR 3	IOR 5	IOR 6
Item/Location Combinations	13,776,000	13,776,000	13,776,000	13,776,000
Combinations Evaluated	12,054,000	10,045,000	6,068,000	4,100,000
Percentage Evaluated	88%	73%	44%	29%
Batch Process Time	Mins	Mins	Mins	Mins
Rplex	29:58	26:24	16:51	12:41
Rplbld	7:23	6:20	4:02	3:00
Reqext	1:26:06	1:14:08	48:21	36:42
<b>Total</b>	<b>2:03:27</b>	<b>1:46:52</b>	<b>1:09:14</b>	<b>52:23</b>





The following table summarizes the results obtained using 6 CPU's.

Parameters	IOR 3	IOR 5	IOR 7
Item/Location Combinations	13,776,000	13,776,000	13,776,000
Combinations Evaluated	10,045,000	6,068,000	2,132,000
Percentage Evaluated	73%	44%	29%
Batch Process Time	Mins	Mins	Mins
Rplext	43:32	29:07	13:53
Rplbld	10:03	6:26	3:09
Reqext	2:05:32	1:21:52	34:09
<b>Total</b>	<b>2:59:07</b>	<b>1:57:25</b>	<b>51:11</b>

### Oracle 8.1.6 (32 bit)

**NOTE:** We made numerous tuning modifications to the database when we upgraded from Oracle 8.1.5 (64 bit) to Oracle 8.1.6 (32 Bit). See the section "Program Modifications between 8.1.5 and 8.1.6" for more details.

The following matrix summarizes the results for the same workload as shown above obtained running on Oracle 8.1.5 (32 bit)

CPUs	IOR 1	IOR 2	IOR 3	IOR 4	IOR 5	IOR 6	IOR 7
24	1:11:48	1:05:26	56:23	47:43	39:10	-	-
18	1:27:16	-	1:08:35	-	46:26	-	-
12	-	1:42:22	1:27:38	-	59:29	45:38	-
6	-	-	2:24:53	-	1:36:21	-	47:29

The following table summarizes the results obtained using 24 CPU's.

Parameters	IOR 1	IOR 2	IOR 3	IOR 4	IOR 5
Item/Location Combinations	13,776,000	13,776,000	13,776,000	13,776,000	13,776,000
Combinations Evaluated	13,776,000	12,054,000	10,045,000	8,077,000	6,068,000
Percentage Evaluated	100%	88%	73%	58%	44%
Batch Process Time	Mins	Mins	Mins	Mins	Mins
rplext	16:42	14:58	12:53	10:48	8:51
rplbld	7:14	6:24	5:35	4:20	3:00
reqext	47:52	44:04	37:55	32:35	27:19
<b>Total</b>	<b>1:11:48</b>	<b>1:05:26</b>	<b>56:23</b>	<b>47:43</b>	<b>39:10</b>

The following table summarizes the results obtained using 18 CPU's.

Parameters	IOR 1	IOR 3	IOR 5
Item/Location Combinations	13,776,000	13,776,000	13,776,000
Combinations Evaluated	13,776,000	10,045,000	6,068,000
Percentage Evaluated	100%	73%	44%
Batch Process Time	Mins	Mins	Mins
rplext	21:25	16:29	11:21
rplbld	7:39	6:11	3:25
reqext	58:12	45:55	31:40
<b>Total</b>	<b>1:27:16</b>	<b>1:08:35</b>	<b>46:26</b>



The following table summarizes the results obtained using 12 CPU's.

Parameters	IOR 2	IOR 3	IOR 5	IOR 6
Item/Location Combinations	13,776,000	13,776,000	13,776,000	13,776,000
Combinations Evaluated	12,054,000	10,045,000	6,068,000	4,100,000
Percentage Evaluated	88%	73%	44%	29%
Batch Process Time	Mins	Mins	Mins	Mins
rplext	25:07	22:19	15:29	12:00
rplbld	6:51	6:33	3:21	2:23
reqext	1:10:34	58:46	40:39	31:15
<b>Total</b>	<b>1:42:32</b>	<b>1:27:38</b>	<b>59:29</b>	<b>45:38</b>

The following table summarizes the results obtained using 6 CPU's.

Parameters	IOR 3	IOR 5	IOR 7
Item/Location Combinations	13,776,000	13,776,000	13,776,000
Combinations Evaluated	10,045,000	6,068,000	4,100,000
Percentage Evaluated	73%	44%	29%
Batch Process Time	Mins	Mins	Mins
rplext	38:37	26:21	14:14
rplbld	8:50	5:13	2:07
reqext	1:37:26	1:04:37	31:08
<b>Total</b>	<b>2:24:53</b>	<b>1:36:21</b>	<b>47:29</b>

NOTE: These matrices only represent a small portion of the total test runs executed. The high water benchmark results were obtained after these tuning runs were executed. We should not compare the 8.1.5 results shown above against the 8.1.6 results because some database and code changes were applied to the 8.1.6 version after we completed the 8.1.5 test.

## Observations

In order to remove the bottlenecks, initial tuning runs were made to determine the internal Oracle instance and session contention. The tuning approach was based primarily on Oracle's wait event statistics, and tuning efforts were focused on minimizing the contention causing the highest waits. Utlestat reports and session tracing with wait event statistics generation revealed valuable results.

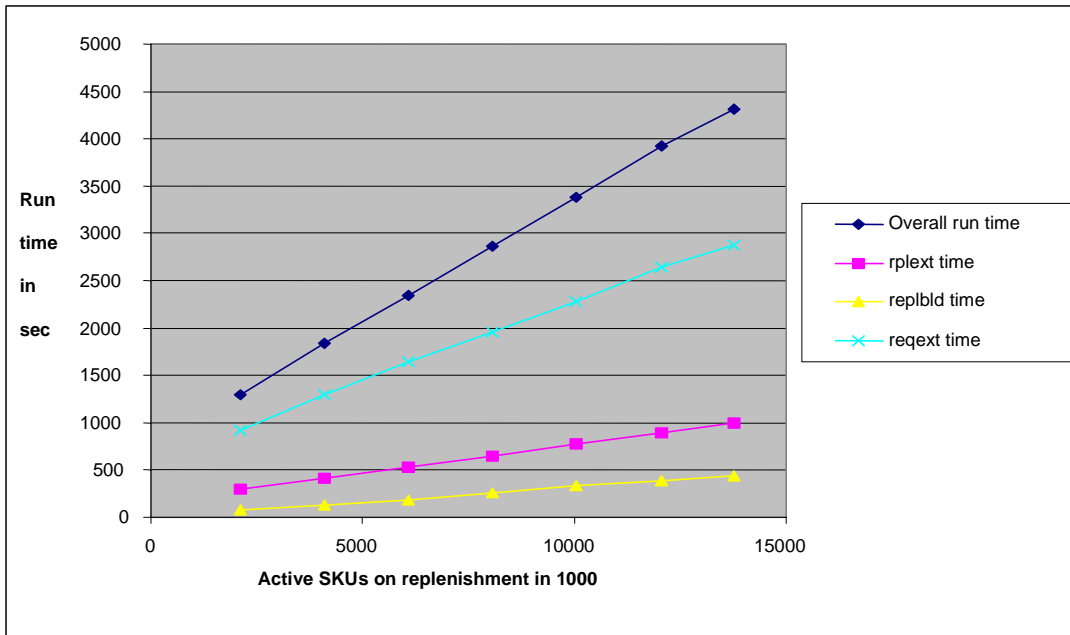
Using the utlestat reports, instance contention was minimized such that:

- Observed hit-ratios were nominal (buffer cache, library cache, and dictionary cache).
- Latch contention was minimal.
- Redo contention was minimal and online log operations were optimal.
- DBWR performance was optimal.
- Rollback segment contention was negligible.
- Data block contention was negligible.
- I/O was distributed as evenly as possible.

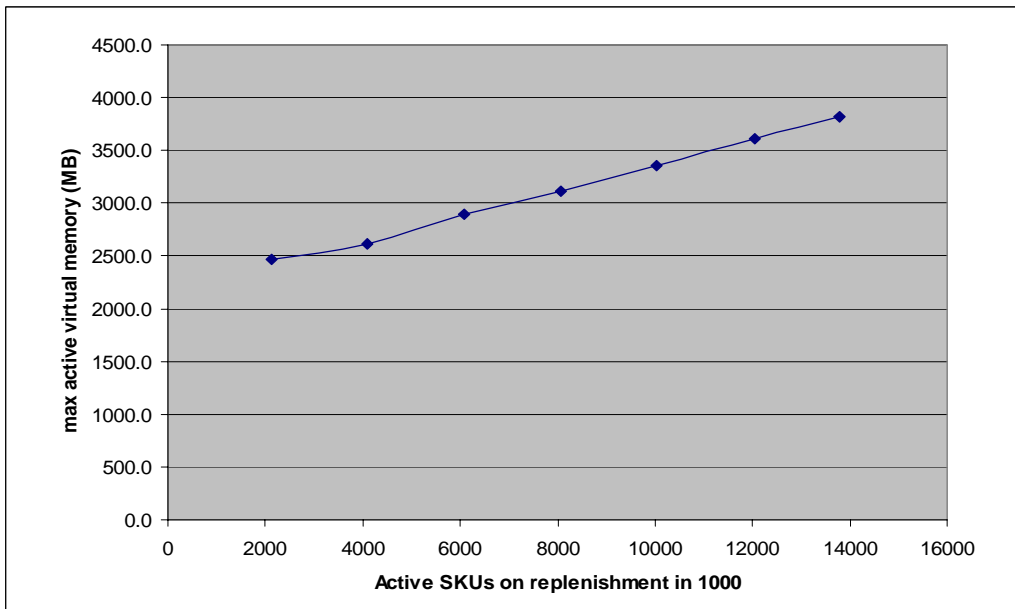
Upon completion of tuning the Oracle instance, the utlestat report showed that almost 75% of the total wait time for non-idle wait events were attributed to the 'db file sequential read' event. This is a wait for I/O completion while Oracle performs a sequential read (indexed access single block reads). Based on this information, further analyses were performed using Oracle's session tracing capabilities.



The following graph displays the number of seconds required to execute the 3 modules compared to the number of active SKUs on replenishment:



The following graph displays the maximum active virtual memory compared to the number of active SKUs on replenishment.





## Database Details

### Tablespace/Disk Configuration.

We decided to use Striped Raw disks when creating the database, utilizing a 64Kb stripe size. We used 5 Volume Groups (VG), with a total of 16 Tablespaces distributed across 80 raw disks (mirrored) and 140 LVG's.

The Temporary Tablespace was designating as type *temporary* used exclusively for sorts. Doing so effectively eliminates the space management operations involved in the allocation and de-allocation of sort space. Obviously we tried to eliminate sorts to disk.

All operations that use sorts, including joins, index builds, ordering (ORDER BY), the computation of aggregates (GROUP BY), and the ANALYZE command to collect optimizer statistics benefit from temporary tablespaces.

### Redo Log Configuration.

The database was configured to have 3 redo log groups containing 1 member per group. Each redo log was 4Gb in size.

Under normal circumstances this may seem excessive, however to reduce log switches, moving to larger redo logs can improve performance by reducing checkpoint and log switch frequency. The trade-off is the potential to increase instance recovery time, if large long running transactions require recovery after instance failure.

During the heaviest database activity Oracle was performing a log switch approximately every 10 minutes.

### Instance Parameters.

We made minimal parameter changes when switching between Oracle 8.1.5 and 8.1.6, besides the obvious 8.1.x specific. The most important consideration is the maximum size limitation permitted under the 32 bit version of Oracle. (approximately 2.5Gb) The following tables lists the important parameters modified:

Parameter	Value
compatible	8.1.5
db_block_size	8192
db_block_buffers	350000
db_block_max_dirty_target	174762
log_buffer	62914560
shared_pool_size	104857600
shared_pool_reserved_size	2621440
pre_page_sga	true
db_block_lru_latches	24
db_file_multiblock_read_count	32
db_writer_processes	1
dbwr_io_slaves	0
disk_asynch_io	true
open_cursors	1500
cursor_space_for_time	true
sessions	225



Parameter	Value
session_cached_cursors	150
dml_locks	7500
processes	500
enqueue_resources	6000
log_checkpoint_interval	994096000
parallel_server	False
_affinity_on	True
sort_area_size	104857600
timed_statistics	True
optimizer_features_enable	8.1.5
hash_multiblock_io_count	8
nls_date_format	DD-MON-RR

We changed the following parameters when we upgraded to Oracle 8.1.6 (32 bit)

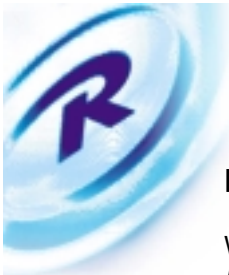
Parameter	Value
compatible	8.1.6
optimizer_features_enable	8.1.6
db_block_buffers	190000

We didn't use the full 2.5Gb of shared memory available when benchmarking with the 32 bit version of Oracle. The tests conducted showed minimal performance improvements. The buffer cache hit ratio remained optimal, with little "buffer busy waits" and latch contention.

### Table/Index Partitioning.

We analyzed each replenishment module identifying the tables and indexes most likely to cause data block and I/O contention. Due to the number of concurrent processes needing to access the same tables and indexes we isolated the following tables as good candidates for Oracle Partitioning.

Table Name	# of Partitions
ORD_TEMP	8
PRICE_HIST	200
REPL_DAY	7
REPL_ITEM_LOC	200
SA_TRAN_HEAD	8
SA_TRAN_ITEM	8
SA_TRAN_TENDER	4
TRAN_DATA	200
TRAN_DATA_HISTORY	200
TSFDETAIL	25
WIN_STORE	200
WIN_STORE_HIST	200
WIN_WH	8
WIN_WH_HIST	8



## Block Level Tuning

We made the some great performance gains at the block level by concentrating on the *PCT\_FREE*, *INI\_TRANS* and *FREELISTS* parameters when creating the tables and indexes.

The freelists parameter determines the number of buffers made available in the buffer cache for transactions to simultaneously insert into. For optimal performance we should set this parameter to the number of transactions that are likely to insert into an object simultaneously.

Changing the intrans storage parameter sets the initial number of transactions that can simultaneously update a block of data. If left at the default value of 1, when there is more than one transaction accessing a single block for update a new 23-byte slot will be created in the block to store the second transaction's identifier. This will cause a wait, thus impacting performance.

With close monitoring of "buffer busy waits" and "latch free" we could isolate the blocks were causing unnecessary contention. We increased these parameters while avoiding the possibility of chained rows. You should be able to practically eliminate this level of contention.

The following partitioned tables were modified to reduce block level contention:


Table Name	Pct Free	Initrans	Freelists
ORD_TEMP	40	25	25
REPL_DAY	10	1	24
REPL_ITEM_LOC	10	1	24
TRAN_DATA	20	24	24
TRAN_DATA_HISTORY	10	1	24
TSFDETAIL	10	24	24
WIN_STORE	20	24	24
WIN_STORE_HIST	5	24	24
WIN_WH	10	24	24
WIN_WH_HIST	5	24	24

The following non-partitioned tables were modified to reduce block level contention:

Table Name	Pct Free	Initrans	Freelists
ALLOC_DETAIL	10	24	24
ALLOC_HEADER	10	24	24
ORDHEAD	10	24	24
ORDLOC	10	24	24
ORDSKU	10	24	24
REPL_RESULTS	10	24	24
REV_ORDERS	10	24	24
TSFHEAD	50	24	24
WIN_SKUS	10	24	24

The following partitioned indexes were modified to reduce block level contention:

Table Name	Pct Free	Initrans	Freelists
PK_REPL_DAY	10	24	24
PK_WIN_STORE_HIST	10	24	24
PK_WIN_WH_HIST	10	24	24
TRAN_DATA_I1	10	24	24



The following non-partitioned indexes were modified to reduce block level contention:

Table Name	Pct Free	Initrans	Freelists
ALLOC_HEADER_I1	10	24	24
ALLOC_HEADER_I2	10	24	24
ALLOC_HEADER_I3	10	24	24
ALLOC_HEADER_I4	10	24	24
ORDHEAD_I1	10	24	24
ORDHEAD_I10	10	24	24
ORDHEAD_I2	10	24	24
ORDHEAD_I3	10	24	24
ORDHEAD_I4	10	24	24
ORDHEAD_I5	10	24	24
ORDHEAD_I6	10	24	24
ORDHEAD_I7	10	24	24
ORDHEAD_I8	10	24	24
ORDHEAD_I9	10	24	24
ORDLOC_I1	10	24	24
ORDSKU_I1	10	24	24
ORDSKU_I2	10	24	24
ORD_TEMP_I2	10	25	25
PK_ALLOC_DETAIL	10	24	24
PK_ALLOC_HEADER	10	24	24
PK_ORDHEAD	10	24	24
PK_ORDLOC	10	24	24
PK_ORDSKU	10	24	24
PK_REPL_ITEM_LOC	10	24	24
PK_TSFHEAD	10	24	24
PK_WIN_SKUS	10	24	24
PK_WIN_STORE	10	24	24
PK_WIN_WH	10	24	24
REPL_ITEM_LOC_I1	10	24	24
REPL_ITEM_LOC_I2	10	24	24
REPL_ITEM_LOC_I3	10	24	24
REPL_ITEM_LOC_I4	10	24	24
REPL_ITEM_LOC_I5	10	24	24
TRAN_DATA_HISTORY_I1	10	24	24
TRAN_DATA_I2	10	24	24
TSFDETAIL_I1	10	24	24
TSFDETAIL_I2	10	24	24
TSFHEAD_I1	10	24	24
TSFHEAD_I2	10	24	24
TSFHEAD_I3	10	24	24
WIN_SKUS_I1	10	24	24
WIN_SKUS_I2	10	24	24
WIN_SKUS_I3	10	24	24
WIN_SKUS_I4	10	24	24
WIN_STORE_I1	10	24	24
WIN_STORE_I2	10	24	24
WIN_WH_I1	10	24	24
WIN_WH_I3	10	24	24



## Tuning Enhancements

### Program Modifications

We made some small performance enhancements to the base Retek code. Other changes were necessary to overcome specific Oracle bugs. The changes we made in no way affected the functionality of the programs, some of these changes will be incorporated into the RMS Version 9 base code. The other changes will not be necessary due to Oracle bug fixes.

For more details on the bugs we encountered with Oracle on AIX please see the section "Oracle Issues".

#### **RPLEXT - The Replenishment Extract module.**

We made the following changes to this program under Oracle 8.1.5 (64 bit):

- Due to the Oracle bug # 884729, we needed to add indicator variables to all the columns fetched in the driving cursor. Indicator variables should only be necessary on columns that potentially return a null value from the database. I don't believe that this effected performance.
- The call to the database package GET\_ITEM\_LOC\_REVIEW\_TIME would only be executed when the item/location review was not equal to 1. This eliminated potentially 4.1 million unnecessary calls/executions in the database. The base product returns the value of 1 if the item/location review is equal to 1.
- The place\_xdock\_wh\_order() function was incorrectly setting the due\_ind variable causing incorrect data being written to the ord\_temp table.
- The call to the function to write out the repl\_results data was removed. This was an unnecessary performance overhead. This was an option in the previous releases of RMS. We believe that this modification will be incorporated into the Version 9 RMS base.
- The NEXT\_ORD\_TEMP\_SEQ\_NO function was changed to eliminate unnecessary database activity. See "Database Modifications" for more details.

#### **RPLBLD - The Replenishment Order Build module.**

We made the following change to this program under Oracle 8.1.5 (64 bit):

- Due to an Oracle bug, we needed to change use the strncmp function rather than the Retek MATCH macro when testing any column that was fetched via the driving cursor.
- Due to the Oracle bug # 907232, we encountered random core dumps of this program. To overcome this problem we installed the Oracle 8.1.6 Pro\*C software and recompiled this code against the Oracle 8.1.5 database.
- In the get\_sup\_info() function two cursors were changed to allow correct functionality. The inclusion of the Oracle to\_number function around the supplier variable in the c\_check\_dept, c\_sup\_dept and c\_no\_sup\_dept cursors was added.
- A new index was placed on the ADDR table to improved the performance of the cursors mention above. See "Database Modifications" for more details.





## **REQEXT - The Item Requisition Extract module.**

We made the following change to this program under Oracle 8.1.5 (64 bit):

- Due to the Oracle bug # 884729, we needed to add indicator variables to all the columns fetched in the driving cursor. Indicator variables should only be necessary on columns that potentially return a null value from the database. I don't believe that this effected performance.
- The call to the database package GET\_ITEM\_LOC\_REVIEW\_TIME would only be executed when the item/location review was not equal to 1. This eliminated potentially 9.2 million unnecessary calls/executions in the database. The base product returns the value of 1 if the item/location review is equal to 1.
- The get\_next\_seq\_no() function was changed. We replaced the existing c\_next\_seq\_no cursor with a select from a new Oracle sequence. This eliminates the unnecessary overhead with manually selecting a unique sequence from the transfer detail table. See "Database Modifications" for more details.

## **Program Modifications between Oracle versions 8.1.5 and 8.1.6**

We encountered several problems when upgrading from Oracle 8.1.5 (64 bit) to 8.1.6 (32 Bit). We created a new 8.1.6 database and repopulated it with the same Retek structure and data.

We were able to recompile the Retek Pro\*C programs under the new Oracle software, but encountered several problems during the program execution.

## **RPLEXT - The Replenishment Extract module.**

We made the following changes to this program under Oracle 8.1.6 (32 bit):

- We removed the additional indicator variables that we added to the columns fetched in the driving cursor because of a bug in version 8.1.5.
- Due to an Oracle bug, to enable successful execution we changed the driving cursors to do a sub-select on the v\_restart\_dept view rather than a join. This change also reduced the overall execution time of these programs by improving the Oracle execution plan.

## **RPLBLD - The Replenishment Order Build module.**

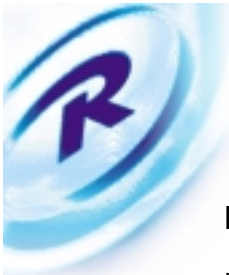
We made no additional changes to this program under Oracle 8.1.6 (32 bit):

## **RPLEXT - The Replenishment Extract module.**

We made the following changes to this program under Oracle 8.1.6 (32 bit):

- We removed the additional indicator variables that we added to the columns fetched in the driving cursor because of a bug in version 8.1.5.
- Due to an Oracle bug, to enable successful execution we changed the driving cursors to do a sub-select on the v\_restart\_dept view rather than a join.

For more details on the problems we encountered with Oracle on AIX please see the section "Oracle Issues".



## Database Modifications

In order to remove the bottlenecks, initial tuning runs were made to determine the internal Oracle instance and session contention. The tuning approach was based primarily on Oracle's wait event statistics, and tuning efforts were focused on minimizing the contention causing the highest waits. Utlestat reports and session tracing with wait event statistics generation revealed valuable results.

### Table/Index Partitioning

One of the key areas for improving performance was to understand how the various processes of this Benchmark were going to access the tables in the database, and how many processes will access the table.

The benefits of partitioning can potentially allow multiple processes (or threads) to process against the same table with minimal I/O contention.

The area most likely to benefit from this process is the Batch System, as these will typically be reading and writing high volumes of transactions, where disk contention is highly likely, thus creating performance issues.

The key criteria to getting this right are to understand the data intimately, and how the programs will be reading and writing. It may also mean that the method for accessing the data in the Batch System (the 'driving cursors') will require modification to map to the partitioning and smooth the volume of data to be processed by each thread.

### Block Level Tuning

We achieved the some great performance gains at the block level by concentrating on the *PCT\_FREE*, *INI\_TRANS* and *FREELISTS* parameters when creating the tables and indexes.

With close monitoring of "buffer busy waits" and "latch free" we could isolate the blocks were causing unnecessary contention. We increased these parameters while avoiding the possibility of chained rows. You should be able to practically eliminate this level of contention.

See "Database Details" for more information on which objects we applied this level of tuning.

### Index Tuning

We used the new Reversed Index feature of Oracle 8 where applicable. This feature is especially useful for primary keys that are sequenced generated. By reversing the keys of the index, the insertions become distributed across all leaf keys in the index.

We also utilized the ability to create a primary key on a non-unique index reducing the index maintenance overhead. This should only be used on primary key constraints that do not have any foreign key relationships.

We created 3 additional indexes to improve the overall access times on the following tables:

- tsfhead
- tsfdetail
- addr



The following indexes were created as a reversed key index:

Index Name
SA_ERROR_I1
SA_TRAN_HEAD_I1
SA_TRAN_ITEM_I1
SA_TRAN_TENDER_I1
TSFDETAIL_I6

### Database Modifications between Oracle version 8.1.5 and 8.1.6

We encountered several problems when upgrading from Oracle 8.1.5 (64 bit) to 8.1.6 (32 Bit). The first problem was a bug with the Oracle upgrade scripts (# 1208625) that meant we could not simply upgrade the database. We created a new 8.1.6 database and repopulated it with the same Retek structure and data.

For more details on the problems we encountered with Oracle on AIX please see the section "Oracle Issues".


Given the opportunity to reorganize the tables and indexes when we recreated the database we modified some storage clauses to eliminate dynamic extension. Dynamic extensive is the process of acquiring more extents of disk space for tables and indexes that have not been allocated a large enough initial extent. When extents are thrown, the information is written immediately to disk to maintain database address consistency. Throwing many extents can cause a bottleneck against the data dictionary structures within Oracle.

## Oracle Issues

As mention in the section "Tuning Modifications" not all changes made to the Pro\*C programs were performance enhancements. We encountered several bugs with Oracle 8.1.x on AIX 4.3.3. The following is a list of the symptoms, work-arounds or bug fixes.

### Oracle 8.1.5 (64 bit)

- We encountered a bug when attempting an array fetch into a structure of arrays. This is a know problem on 8.1.5 Pro\*C. There are several methods of overcoming this problem. Initially we opted to add indicator variables to all the columns in the fetch. This by-passed the problem but due to other bugs with 8.1.5 we installed the 8.1.6 version of Pro\*C in a separate ORACLE\_HOME, then recompiled the programs against the 8.1.5 database.
- We were unable to utilize the large amount of memory available due to a bug that caused the server to crash randomly. The server became unstable when the SGA was larger than 3.5Gb. The crashes occurred spasmodically. Sometimes when starting the database, sometimes when running the Pro\*C programs and once when shutting the database after a successful run.
- Another problem encountered randomly during execution of the Pro\*C programs, was unusual core dumps occurring at different times. These core dumps seemed to be causes by any memory manipulation call. (malloc, calloc or realloc). This problem was overcome by installing the 8.1.6 version of Pro\*C in a separate ORACLE\_HOME, then recompiled the programs against the 8.1.5 database.



## Oracle 8.1.6 (32 bit)

- The first problem encountered was a bug with the Oracle upgrade scripts. We were unable to upgrade the database from Oracle 8.1.5 (64 bit) to 8.1.6 (32 bit). We tried installing Oracle 8.1.5 (32 bit) and then downsizing the word size as recommended by Oracle. This also failed with another bug encountered. The only way we could overcome this problem was to create a new database under 8.1.6. This problem had been fixed on Solaris and HP-UX, but the fix was not yet ported to AIX.
- We encountered another bug when trying a joined query against the 8.1.6 database. We encountered this problem initially in Pro\*C, but during further testing it also occurred in SQL\*Plus. This was also logged with Oracle Support as a bug. (We received an ORA-600 with this problem).
- The final problem discovered was the appearance of a single white space appended to any varchar column fetched in Pro\*C via an array fetch. Fortunately we were able to simply strip this additional white space off.



## Appendix A – Disk Configuration

The following table describes the disk layout in detail:

		Loop A							Loop B								
I/O	ssa	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Draw1	ssa1	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
	ssa2	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
	ssa15	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181
I/O	ssa4	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69
Draw2	ssa6	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85
	ssa8	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
	ssa16	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197
I/O	ssa9	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117
Draw3	ssa11	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133
	ssa12	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149
	ssa14	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165

VG\_Name PP\_Size Hdisk - Primary : Mirror

rettek1 16 6,22,38,166,54,70,86,182,102,118,134,150,12,28,60,108:  
62,78,94,190,110,126,142,158,14,30,46,174,109,125,156,188

rettek2a 16 7,23,39,167,55,71,87,183,103,119,135,151,44,172,92,140:  
63,79,95,191,111,127,143,159,15,31,47,175,141,157,124,76

rettek2b 16 8,24,40,168,56,72,88,184,104,120,136,152,13,61,116,132:  
64,80,96,192,112,128,144,160,16,32,48,176,189,173,21,37

rettek3a 16 9,25,41,169,57,73,89,185,105,121,137,153,29,77,148,164:  
65,81,97,193,113,129,145,161,17,33,49,177,93,45,53,181

rettek3b 16 10,26,42,170,58,74,90,186,106,122,138,154,20,68,149,165:  
66,82,98,194,114,130,146,162,18,34,50,178,196,180,85,69



## Appendix B – System Performance Notes

### CPU

The CPU usage data demonstrated a healthy and smooth use of the available processors for the entire benchmark.

Using 24-process, 24-CPU configuration as base for all load levels (IOR1 - 7), The CPU usage for the application was 85 to 90 percent throughout. We noticed almost no I/O wait, with only 2-3% on system idle. The rest (less than 10 to 15%) was on system kernel time.

Another observation was based on the same work load (IOR1) on the different system configurations, which includes 24-CPU/24-processes, 18-CPU/18-processes, 12-CPU/12-process, 6-CPU/6-process and 24-CPU/33-process combinations.

The result was interesting enough to indicate that the CPU usage was at almost a constant value of 85-88% for application, with only the 33-process run reaching slightly higher than 90%. The kernel time stayed at 10% for all the tests. This gives a clear picture of the superior scalability of the S80 server. That is, when the load increase is proportional to the number of CPU increase, there was no noticeable CPU kernel time increase due to SMP overhead. The benchmark result showed that all the increased CPU capacity went to serve the application in a "no-cost" manner. This result reflects a fact that the S80's balanced internal design, the cross-bar switch and the AIX 4.3.3 together have made a system is scalable enough to handle huge IS application as well as medium or small ones. This capability is especially important for multi-task or multi-thread applications.

### Memory

Throughout the entire benchmark there has been no sign of memory shortage, in fact, we did not even notice any major paging activities. With 64G RAM installed on the test machine, it is warranted that the memory is sufficient for most of today's applications. Then the question is, what would be a typical memory requirement for a Retek customer environment?

Below, we show the memory requirement under different combinations of workload, number of processes and number of processors:

Max active memory (MB)

	CPU24	CPU18	CPU12	CPU06
<b>IOR7</b>	2469	2338.2	2205.6	2193.3
<b>IOR6</b>	2613.4		2420.4	2347.8
<b>IOR5</b>	2891.2	2743.9	2666.9	2684.5
<b>IOR4</b>	3116.1	2997.1	2921.4	2940.3
<b>IOR3</b>	3361.5	3240.2	2921.4	2940.4
<b>IOR2</b>	3614.5	3492.9	3234.9	2941.3
<b>IOR1</b>	3820.2	3791.8	3234.9	3056.6

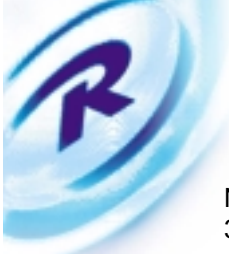
Multi-task level:

CPU24 - 25 processes

CPU18 - 19 processes

CPU12 - 13 processes

CPU06 - 7 processes



Note: with the optimized run (run #62, 24CPU, IOR1, 33/32/33 processes, 59min) we used up to 3870MB. Refer to the main report for run details.

From the above table, we can predict that for the load level we have tested, the IOR1/CPU24 combination would require the highest memory amount of up to 4G, while the lowest combination (IOR7/CPU6) needed a little more than 2G. This is the minimum amount of memory that is required for the Retek application and Oracle database to run without major paging activities.

It is important to indicate, however, any additional system real memory is not wasted. AIX operating system will always use them for JFS file caching to maintain a high performance for random and sequential files. For example, a sequential file and/or a file system on disk will become fragmented over time due to allocation and reallocation of the disk partitions, inode structures and data blocks. Once a large file is mapped into memory, the access speed becomes memory speed, and the fragmentation goes away, delivering a much higher level of I/O performance. The additional real memory is also important to support other applications running concurrently.

## **I/O**

The CPU report showed minimal I/O wait for the entire benchmark process and very light disk busy rate from the I/O report. This result was contributed by the following factors:

- A large number of fast SSA disks
- A well designed database table/table space layout
- Disk Adapters with large write cache
- The AIX logical volumes span across a large number of physical disks. (This forms the basis for parallel disk access)
- File striping supported directly at the AIX operating system level