OpenVMS Technical Journal V7

Oracle Rdb Monitoring and Alarming on OpenVMS

Kostas G. Gavrielidis, Master Technologist HP Services

Overview

Several layered products and utilities exist for monitoring Oracle Rdb databases on OpenVMS platforms. In this article, we review products available from other companies and propose an HP internally-developed solution developed and deployed in customer production environments.

There are several Rdb monitoring and alarming utilities and tools available today:

- `RMU/SHOW STATISTICS` Utility

  The `RMU/SHOW STATISTICS` utility, one of the most powerful and useful tools available to database administrators (DBAs), *analyzes* performance characteristics of a database. However, the ability to analyze performance problems is only one aspect of a performance analysis tool. The DBA also needs to *detect* problems in a timely manner, then analyze the problems and immediately perform *corrective* actions.

- PATROL Knowledge Module for Rdb

  PATROL monitors and manages the resources in the environment using information from special files known as Knowledge Modules (KM) that are loaded in the console. If PATROL detects a problem with a particular computer or application that it is monitoring, these modules provide information for PATROL to attempt to fix the problem. If the problem escalates or requires human intervention, PATROL displays a warning or alarm condition for every resource affected by the problem.

- Oracle Rdb Trace and Rdb Expert

  Oracle Trace is a layered product that gathers and reports event-based data from OpenVMS layered products and application programs that contain its service routine calls. Oracle Trace provides Oracle Expert for Rdb along with data for optimizing existing Rdb databases. Event-based data can be collected from products that contain Oracle Trace service routine calls. In addition, Oracle Trace routine calls can be added to other user-developed applications to

collect data from them. This process of adding Oracle Trace service routine calls to an application is called *instrumenting the application*.

- Internally-Developed Solutions

  Internally-developed solutions consist of the `RMU/SHOW STATISTICS` Utility along with alarming and notification features based on rules customized by the database administrator.

This article is written for those who implement enterprise solutions that involve HP customer's Oracle Rdb and OpenVMS production environment configurations.

Database Monitoring Objectives

Database objectives include providing monitoring and alarming capabilities that support overall Enterprise Management (EM) activities. These objectives maximize the availability of applications and support the monitoring and alarming requirements that meet the following objectives:

- Develop software scripts, as necessary, to facilitate the monitoring and alarming of important databases by focusing on specific areas previously identified by the customer. These include (but are not limited to) the following areas:
  a. database status (i.e., up or down)
  b. elapsed transaction processing time
  c. transaction rate
  d. file system fragmentation
  e. record locking
  f. record level fragmentation
  g. critical transaction rate
  h. lock rate
  i. transaction duration
  j. number of fetches
  k. number of stores
  l. number of erases
- Document all developed software and the installation and deployment processes that be performed in the customer production environment.

RMU/SHOW STATISTICS Utility

Introduction

Most database products provide tools to analyze the performance characteristics of the database and the application. These tools often require constant manual attention or extensive post-processing of recorded data in order to detect critical events that might be detrimental to the smooth operation of the database.

In some cases, the DBA is not always available to constantly monitor the utility output, especially for 24*7 production databases. Post-processing recorded data is certainly not timely enough to prevent or resolve potential real-time database downtime situations, particularly with real-world, mission-critical requirements. The DBA needs the ability to be *notified* automatically when time-critical or interesting events occur to the database.

RMU/SHOW STATISTICS Events

The RMU/SHOW STATISTICS utility has been dramatically enhanced for Rdb7 to include many new features that provide DBAs with the information necessary to *detect*, *analyze*, and *correct* performance problems in the database. Even more enhancements have been added to the Rdb7A release, including the *Configuration File* facility, which provides an extremely powerful and easy-to-use *User-Defined Event* facility.

Using the RMU/SHOW STATISTICS configuration file, you can persistently define special *events* that specify the action to be performed when something of interest occurs to the runtime database. An event is an identification of a particular statistic value on which the RMU/SHOW STATISTICS utility will perform some user-defined action. In other words, an event is signaled when a statistic's value exceeds a user-defined set of thresholds.

Using events, the DBA can be automatically notified by an RMU/SHOW STATISTICS utility server running on behalf of a particular database. You define an event by specifying a threshold against a specific statistic and by optionally specifying the attributes the event is to have.

Event Definition Syntax

The RMU/SHOW STATISTICS utility configuration file is a text file that can be maintained using the editor of your choice. The configuration file typically resides in the database directory, although it can reside anywhere that you desire.

Each entry in the configuration file uses the general format `variable=value;`. The equal-sign (=) separating the variable and value is required. Note also that each definition is terminated with a semicolon character (`;`).

User-defined events are specified using the EVENT_DESCRIPTION variable. Events themselves are *not* named; rather, they are defined on behalf of a specific statistic for a given threshold. The EVENT_DESCRIPTION variable's value is a free-format description of the user-specified event. The event definition consists of three required, position-dependent components and an optional component. The following example describes the general format:

```
EVENT_DESCRIPTION="operation \
    statistic_name \
    threshold_name \
    [ attribute_list ]";
```

Event "Operation" Clause

The event *operation* clause identifies the action to be performed for the EVENT_DESCRIPTION operation. The keyword ENABLE is used to enable a new event or change an existing event definition. The keyword DISABLE is used to disable a previously defined event. This keyword is typically used when importing a new configuration file.

Even though an event may be enabled, it may not be active. For an event to be *active*, you must also specify either a program to be invoked or one or more operator classes to be notified.

During runtime, events can be disabled only by the RMU/SHOW STATISTICS utility or by importing a new configuration file that explicitly disables an event.

Event "Statistic Name" Clause

The event *statistic name* clause identifies the particular valid statistic field for which the event will be enabled or disabled. Note that some statistic names are valid only when certain database attributes

are enabled, such as global buffers or record caching. The names of a particular statistic field can be found on the individual screen of interest. When statistic field names contain multiple words, such as *process attaches*, the statistic name must be either single- or double-quoted; failure to quote the statistic name may result in a syntax error.

Certain statistic fields have leading blanks. This white space is considered part of the statistic name and is necessary to identify a unique statistic. However, the use of leading blanks is often difficult to discern in the configuration file. Therefore, the underscore character (_) or dash character (-) can be used in place of spaces in statistic names that have leading spaces. For example, the statistic field name " *file extend*" can also be specified as "_ _ *file_extend*" or "- - *file-extend*". This method improves the readability of difficult statistic field names.

Most general events are defined using the *summary* statistics screens. However, it is sometimes necessary to define an event on a specific table or index, or even a particular partition of a table. The AREA attribute allows you to specify this type of *drill-down event* and indicate the name of a particular storage area. When this clause is specified, the statistic field selected must be from the *IO Statistics (by file)* or *Locking Statistics (by file)* screens. The identified statistic name can be also qualified with the LAREA attribute to specify the name of a particular logical area, such as a table, btree index, hash index, or blob. When this clause is specified, the statistic field selected must be from the Logical Area screens. Further, if the selected logical area is partitioned across multiple storage areas, the AREA clause can also be used to identify a specific partition against which to define the event.

The following table explains the semantics for specifying the AREA and LAREA clauses to qualify a statistic field name:

AREA	LAREA	Description
No	No	General Statistic Field
Yes	No	Storage Area Statistic Field
No	Yes	Logical Area Statistic Field, all partitions
Yes	Yes	Logical Area Statistic Field, single partition

The AREA and LAREA clauses are attributes and must follow the *Threshold Name* component of the event definition.

Event "Threshold Name" Clause

The event *threshold name* clause identifies the particular event threshold for which the specified statistic will be enabled or disabled. The thresholds are essentially the columns in the numeric version of the statistics screens.

Up to eight different thresholds can be specified for a particular statistic field, although each individual event name must be specified in its own EVENT_DESCRIPTION variable definition. The *threshold name* clauses are as follows:

- MAX_RATE – The maximum "current" occurrence-per-second rate collected. This threshold only *increases* as each event is signaled.
- MAX_CUR_TOTAL – The maximum "total" value collected since the database was opened. This threshold only *increases* as each event is signaled.
- MIN_CUR_RATE – The lowest rate currently being sustained. This threshold remains constant.
- MAX_CUR_RATE – The highest rate currently being sustained. This threshold remains constant.

- MIN_AVG_RATE – The lowest average rate. This threshold only *decreases* as each event is signaled.
- MAX_AVG_RATE – The highest average rate. This threshold only *increases* as each event is signaled.
- MIN_PER_TX – The lowest per-transaction rate. This threshold only *decreases* as each event is signaled.
- MAX_PER_TX – The highest per-transaction rate. This threshold only *increases* as each event is signaled.

Event "Attribute List" Clause

The optional event *attribute list* clause provides additional characteristics for *enabled* event thresholds. In general, these attributes are ignored when disabling an event. Any or all of the event attributes can be specified for each event name within the same EVENT_DESCRIPTION variable definition. The *attribute list* clauses are as follows:

- AREA *storage_area_name* – Defines the name of a particular storage area. When this clause is specified, the statistic field selected must be from the "IO Statistics (by file)" or "Locking Statistics (by file)" screens, unless the LAREA clause is also specified. This clause is not ignored when disabling the event.
- LAREA *logical_area_name* – Defines the name of a particular logical area, such as a table, btree index, hash index, or blob. When this clause is specified, the statistic field selected must be from the "Logical Area" screens. This clause is not ignored when disabling the event.
- INITIAL *value* – Defines the initial value of the "current" event threshold. The default value is zero (0) for MAX_XXX thresholds and "very big number" for MIN_XXX thresholds. The default value guarantees that at least one event will be signaled, thereby initializing the new "current" threshold value.
- EVERY *value* – Defines the value by which the initial threshold will be incremented or decremented when an event is signaled. If this value is the default value zero (0) for any event except the MIN_CUR_RATE and MAX_CUR_RATE events, then the event will be signaled only once.
- LIMIT *value* – Defines the maximum number of times the event can be signaled. If the value is the default value zero (0), events can be signaled indefinitely providing that the EVERY clause is specified with a non-zero value.
- SKIP *value* – Defines the number of event notifications to *ignore* before performing an actual notification. This clause is extremely useful for the MIN_CUR_RATE and MAX_CUR_RATE events, as the thresholds for these events are not reset upon being signaled. The default value zero (0) ensures that all events are notified.
- NOTIFY *oper_class_list* – Defines the *quoted* comma-separated list of operators to be notified for all events defined on the specified statistic. Valid operator keywords are CENTRAL, DISKS, CLUSTER, SECURITY and OPER1 through OPER12.
- INVOKE *program_name* – Defines the user-supplied program to be invoked for all events defined on the specified statistic. On OpenVMS, the program name is specified as a DCL process global symbol known to the RMU/SHOW STATISTICS utility.

Event Description Readability

Very long configuration file lines can be *continued* on the next line by terminating the line with a back-slash (\). As the last character of a line, this continuation character is used to indicate that the configuration entry is continued on the next line exactly as if it were entered as a single line. Lines can be continued practically indefinitely, up to 2048 characters, even within quoted values. The following example demonstrates how to define a multi-line event description:

```
EVENT_DESCRIPTION="ENABLE 'pages checked' \
    MAX_CUR_TOTAL \
    INITIAL 7 \
    EVERY 11 \
    LIMIT 100 \
    INVOKE DB_ALERT";
```

The continuation character is not limited to the `EVENT_DESCRIPTION` variable; it can be used for any configuration variable.

Comments can be embedded in continued lines if they start at the beginning of the next line. The following example demonstrates two event descriptions containing embedded comments. The comment in the second event description takes precedence over the line continuation character.

```
EVENT_DESCRIPTION="ENABLE ' (Asynch. reads)' \
    MAX_CUR_TOTAL \
    AREA EMPIDS_OVER \
! this will work as expected
    INITIAL 6 EVERY 10 LIMIT 100 \
    INVOKE DB_ALERT";
EVENT_DESCRIPTION="ENABLE ' (Asynch. reads)' \
    MAX_CUR_TOTAL ! this will NOT work as expected \
    AREA EMPIDS_OVER \
    INITIAL 6 EVERY 10 LIMIT 100 \
    INVOKE DB_ALERT";
```

Event Semantics

For an event to be active, you must specify either one or both of the `NOTIFY` or `INVOKE` attribute clauses. When using the `INVOKE` attribute clause, the program must be specified by defining a process-global symbol pointing to the DCL command procedure or image to be invoked. The `INVOKE` program and `NOTIFY` operator classes apply to all events defined for the statistic field. Therefore, these clauses need to be defined only *once* per statistic field, no matter how many events thresholds are defined for that statistic. By specifying multiple programs or operator classes, only the last-specified attribute is used.

Once an event has been signaled, it will only be re-signaled if the `EVERY` attribute clause was specified with a non-zero value. The current threshold value, originally initialized to the `INITIAL` value, will be advanced for `MAX_XXX` thresholds and declined for `MIN_XXX` thresholds. The exceptions to this rule are the *current rate* thresholds `MIN_CUR_RATE` and `MAX_CUR_RATE`, which are never advanced nor declined. The `MIN_XXX` thresholds disable themselves once the `INITIAL` value reaches zero (0), while the `MAX_XXX` thresholds never disable themselves.

Once an event has been disabled, it can be re-enabled only by importing a new configuration file or by manually using the *Statistics Event Information* screen, *Re-enable all disabled events* configuration sub-menu option. Individual events cannot be re-enabled on line.

How User-Defined Events Work

The user-defined events are analyzed by the `RMU/SHOW STATISTICS` utility at the specified screen refresh rate. The default screen refresh rate of 3-seconds is ideal for most databases. However, using a 1-second refresh rate will produce a finer granularity event signaling mechanism. Multiple events defined for the same statistic field may cause the specified program to be invoked multiple times (once for each affected event).

As the `RMU/SHOW STATISTICS` utility identifies a statistic field whose current value or average value is changing, it examines any defined event thresholds established for that statistic field. This manner of examination minimizes the impact of event analysis, since the analysis is performed as part of the normal statistics collection process.

When the utility determines that a specified event threshold has been exceeded, an event is signaled. The signaling of the event means that any specified programs will be invoked and any specified operators will be notified. The event notification occurs immediately.

If you defined a program that will be invoked when an event is signaled, the program will be invoked with eight parameters. Some of the parameters contain multiple words that must be quoted if the parameters are passed to other utilities.

The parameters passed to invoked programs are as follows:

- P1 – This parameter is the date and time the event occurred. This parameter contains embedded blanks.
- P2 – This parameter is the statistic field name. This parameter may contain embedded blanks.
- P3 – This parameter is the event name.
- P4 – This parameter is the current event numeric value, expressed to the nearest tenth.
- P5 – This parameter is the word *above* or *below*.
- P6 – This parameter is the current event threshold value.
- P7 – This parameter is the event occurrence count.
- P8 – This parameter is the optional physical area and/or logical area name for the statistic field.

The P8 parameter is either null (blank) or contains the name of the affected storage area and/or logical area. The following example contains a log file sample output where an event for a partitioned logical area was signaled. Note that when both the storage area and logical area names are specified, they are separated by a period (.).

```
        pages checked MAX_CUR_TOTAL 6.0 above 4.0 count is 1
        area is EMPIDS_MID.EMPLOYEES
        pages checked MAX_CUR_TOTAL 32820.0 above 5.0 count is 1
        area is EMPIDS_OVER.EMPLOYEES
```

Runtime Event Status Information

The current runtime status of the user-defined events can be examined using the new *Statistics Event Information* screen, located in the *Database Parameters* sub-menu. Note that you do *not* have to be viewing this screen to signal events. Note also that the physical area and logical area identifiers are only displayed in *Full* mode.

Real-Life User-Defined Event Example

Nothing demonstrates a new feature better than a real-life example explained in step-by-step detail. For the purposes of this example, suppose that the DBA wants to be sent email whenever a database *freeze* occurs. A database *freeze* occurs when an application process on the database prematurely terminates (i.e., "dies"). Such an event results in all application activity being temporarily suspended until the recovery operation for the terminated process has been completed. This is a very significant and serious runtime event that should be immediately detected.

Using events to notify the DBA when a process terminates prematurely is very easy to accomplish. The following steps describe how this can be achieved using the `RMU/SHOW STATISTICS` utility *User-Defined Events*:

1. Identify the operation. Because you are going to define a new event, specify the `ENABLE` operation keyword.
2. Identify the statistic name to which the event will be assigned. Use the "process failures" statistic from the "Recovery Statistics" screen, which is located in the "AIJ Information" sub-menu. This statistic is available even if you are not using after-image journaling.
3. Identify the threshold name to use. Use the `MAX_CUR_TOTAL` threshold, since this represents the current number of processes that have failed.
4. Identify the event attributes to use. This is probably the hardest part of defining an event. You want to be alerted to *any* process failure, so you must set the INITIAL attribute to zero (0). Since you want to be notified on each and every process failure, set the `EVERY` attribute to one (1) and the `LIMIT` attribute to zero (0).
5. Define how you will be alerted about the event. Since you want to be sent mail, use the `INVOKE` clause. Invoking a program on OpenVMS requires that you define a "DCL process-global symbol" to identify the actual DCL script, as is demonstrated by the following example:

```
$ DBR_LOGGER :==@SYS$SYSTEM:DBR_LOGGER.COM
```

6. Write the program to be invoked. Since you want to be sent mail with a clear description of the event actually signaled, use the simple DCL script in the following example:

```
$ set noon
$ create /nolog sys$scratch:dbr_logger.tmp
EOD
$ open /write dbr_logger sys$scratch:dbr_logger.tmp
$ write dbr_logger " '''p1' ''p2' ''p3' ''p4' ", -
" ''p5' ''p6' (count is ''p7') area is ''p8' "
$ close dbr_logger
$ mail sys$scratch:dbr_logger.tmp -
          RDB_DBA_USER /subject="DBR notification"
```

7. Combine all of this information into the configuration file entry. The following example contains the final event description as you would enter it in the configuration file:

```
EVENT_DESCRIPTION="ENABLE 'process failures'\
 MAX_CUR_TOTAL \
 INITIAL 0 EVERY 1 LIMIT 0 \
 INVOKE DBR_LOGGER";
```

8. Invoke the RMU/SHOW STATISTICS utility as a server, using the configuration file. Be sure to use the /CLUSTER command qualifier if you want to be notified of cluster-wide events. The following example demonstrates the command-line to perform this operation:

```
$ RMU/SHOW STATISTIC -
    /CONFIG=CONFIG.CFG -
    /NOINTERACTIVE /UNTIL=HH:MM:SS -
    MF_PERSONNEL
```

Because applications increasingly require 24*7 availability, the rate at which DBAs are expected to react to potential downtime increases accordingly. The RMU/SHOW STATISTICS utility *User-Defined Events* provides the means by which DBAs can be automatically alerted when such critical situations arise, therefore enabling timely corrective actions.

Glossary of Terms for this Section

Configuration Variable – A symbolic name that defines a value that can be used to define other variables' values, or can be used by the RMU/SHOW STATISTICS utility.

Drill-Down Event – An event defined on a specific storage area, logical area, or logical area partition statistic.

Event – The identification of a particular statistic value on which the RMU/SHOW STATISTICS utility is to perform some user-defined action.

Oracle Rdb – A high-end client/server relational database for Alpha AXP and VAX.

Statistic Name – The valid statistic field for which the event is to be enabled or disabled.

Summary Event – A general-purpose event defined on a "summary" statistic field.

Threshold Name – The columns in the numeric version of the statistics screens for the specified statistic to be enabled or disabled.

PATROL Knowledge Module for Rdb

PATROL is a systems, applications, and event management tool for database and system administrators. It provides an object-oriented graphical workspace where you can view the status of every vital resource in the distributed environment that it is managing. PATROL monitors and manages the resources in the environment using information obtained from special files called Knowledge Modules (KM) that are loaded in the console. If PATROL detects a problem with a particular computer or application that it is monitoring, then these modules provide "knowledge" information that PATROL will use to attempt to fix the problem. If the problem escalates or requires human intervention, PATROL displays every resource affected by the problem in a warning or alarm condition. **Table 1** (see Appendix) includes a list of all the Rdb KM parameters that can be monitored. PATROL is made up of three major components: the PATROL Console, the PATROL Agent, and the Knowledge Modules.

Rdb Trace and Rdb Expert

Oracle Trace is a layered product that gathers and reports event-based data from any combination of OpenVMS layered products and application programs containing Oracle Trace service routine calls. Currently, the only way to perform gathering and reporting on Rdb databases on OpenVMS is through the Oracle Trace for OpenVMS layered product. Application programs that can contain Oracle Trace service routines are considered to be facilities that include the following products: DEC ACMS, DEC ALL-IN-1, DECforms, Oracle CODASYL DBMS, Oracle RALLY, and Oracle Rdb.

Oracle Trace provides Oracle Expert for Rdb along with data that it uses to optimize existing Rdb databases. Event-based data can be collected from products that contain Oracle Trace service routine calls. In addition, Oracle Trace routine calls can be added to other user-developed applications to collect data from them. The process of adding Oracle Trace service routine calls to an application is called *instrumenting the application*.

The Oracle Trace software operates with minimum performance impact on the system. It can run with both the development and production versions of your application to give you information about the behavior of your application.

Features of the Oracle Trace

Using Oracle Trace, the event data collected from applications can be used for different purposes, including the following:

- Tuning and performance improvements of applications
- Planning for hardware resources, (i.e., capacity planning, and so on)
- Tuning the performance of the databases
- Debugging applications
- Logging errors

Tuning the Performance of Databases

Oracle Trace provides request and transactional-level information from Oracle Rdb. This information allows a database administrator to examine a wide variety of performance statistics and usage information related to actual transactions and DML requests.

Oracle Trace provides RdbExpert information that RdbExpert uses to produce more efficient database designs. Database administrators no longer have to guess about the database workload because Oracle Trace collects actual workload information.

Transaction Processing

Oracle Trace tabular reports identify occurrences such as transaction with the higher virtual memory usage or the 95th percentile disk I/O for each transaction. For transaction processing, Oracle trace gathers information for DEC ACMS events to provide task-level performance information.

Relating Events Among Facilities

Oracle Trace allows the instrumentation of routines that use the cross-facility capability. For example, using the cross-facility, Oracle Trace associates the ACMS Procedure Call event with its related Oracle Rdb transactions and requests in order to provide a greater understanding of the total resources the ACMS Procedure Call uses.

Internally Developed Tool

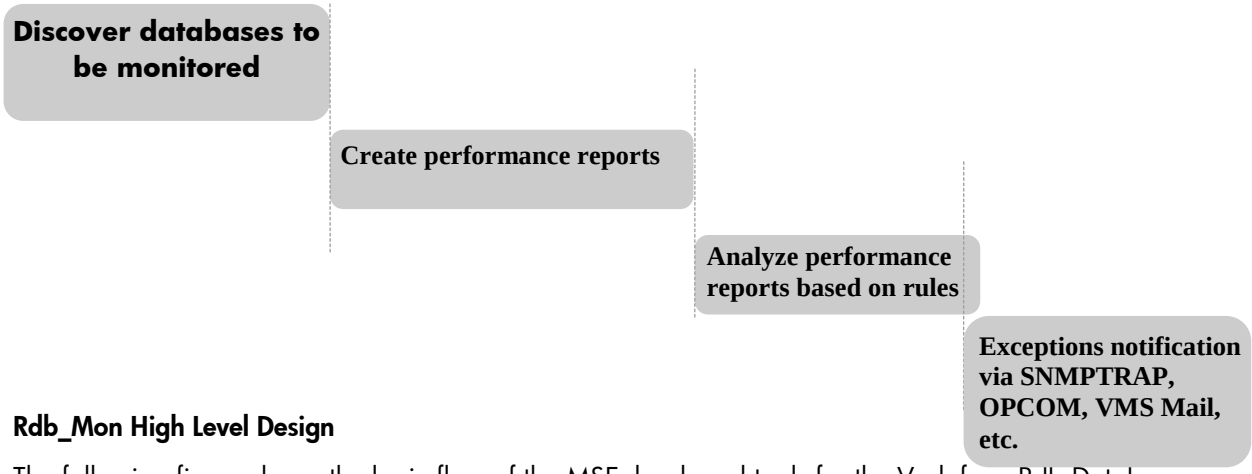
The MSE `Rdb_Mon` utility is an nternally-developed application that makes use of the Oracle Rdb `RMU/SHOW STATISTICS` utility and adds alarming and notifications. Developed by MSE, this tool

contains a number of DCL procedures that can be adapted to the needs and requirements of the Oracle Rdb DBA.

Several forms of the Rdb_Mon utility currently exist. One form is a standalone configuration, comprising a number of DCL command procedures, where the tool is not integrated with any other enterprise management system. Another form integrates with an enterprise-wide management system. Currently, the latter from has been integrated with the Heroix Enterprise Management products RoboMon, RoboEDA, and so on. Rdb_Mon can be customized to be included and integrated under any other suite of enterprise management products.

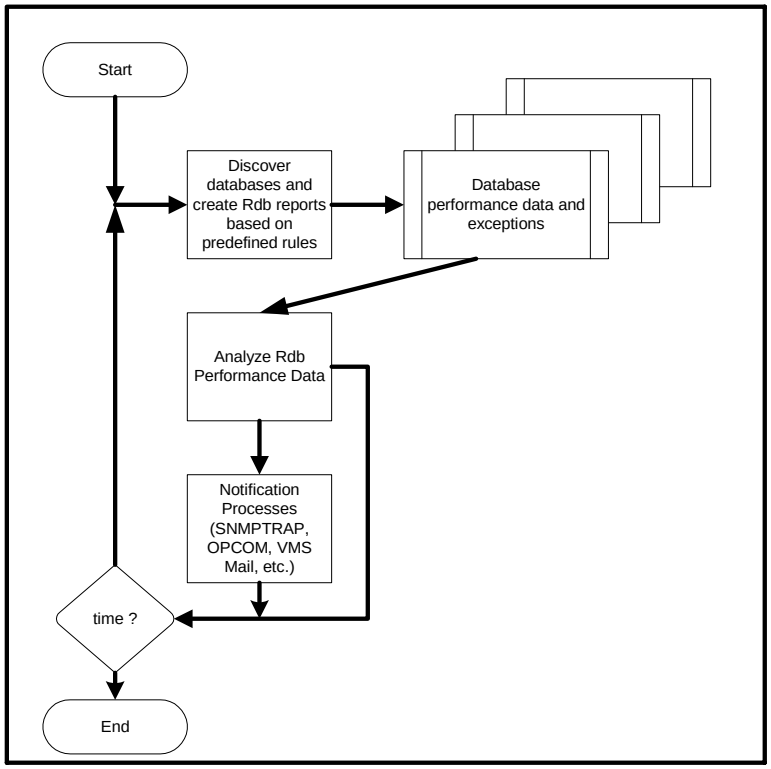
Database Monitoring and Alarming Basic Sequence

The following figure illustrates the sequence for monitoring databases:



Rdb_Mon High Level Design

The following figure shows the logic flow of the MSE-developed tools for the Vodafone Rdb Database Monitoring and Alarming project:



Summary

This article described the monitoring and alarming capabilities of existing utilities and products for the Oracle Rdb database engine on OpenVMS platforms. It presented the MSE `Rdb_Mon` utility as one of the internally-developed alternatives for this technology sector.

References

http://www.oracle.com/technology/products/rdb/index.html - Oracle Rdb

http://documents.bmc.com/supportu/documents/55/16/5516/5516.pdf - PATROL Knowledge Module for Rdb User Guide

Kostas G. Gavrielidis works in HP Services Customer Support and has been with HP for more than 20 years. Currently, and for the last 10 years, he is involved with the MSE proactive consulting projects for our customer production Database Management systems, and works on the analysis and performance improvements for SAP R/3, Oracle, Rdb, Ingres, SYBASE, SQL Server on UNIX, OpenVMS, and Windows platforms.

Appendix - Rdb KM parameters that can be monitored

Table 1: Rdb KM parameters that can be monitored

Parameter	Description
RDB_aij_reads	Displays the number of read QIOs issued to the database .AIJ file (if after-image journaling is enabled).
RDB_aij_writes	Displays the total number of QIOs issued to the database after-image journal file (if after-image journaling is enabled).
RDB_attaches	Displays the number of current attaches to the database.
RDB_blasts	Monitors the number of blocking AST's delivered to Rdb by the OpenVMS lock manager.
RDB_buf_unmark	This parameter is incremented each time a modified buffer is written back to disk. Its value is equal to the sum of the 14 fields: transaction, pool overflow, blocking AST, lock quota, lock conflict, user unbind, batch rollback, new area mode, larea change, incr backup, no aij access, truncate snaps, checkpoint, and aij_backup.
RDB_check_pts	Displays the current number of checkpoints per minute.
RDB_df_reads	Displays the number of read QIOs issued to the database storage area for a single-file and multifile databases and snapshot files.
RDB_df_writes	Displays the number of write QIOs issued to the database storage area for a single-file and multifile databases and snapshot files.
RDB_dup_nd_ins	Displays the number of duplicate index keys inserted into the database's indexes. There should be a one-to-one correspondence to the number of duplicate records being stored in the table.
RDB_fetch_read	Displays the number of synchronous data page requests to the PIO subsystem where only read privileges are being requested for the page.
RDB_fetch_upd	Displays the number of data page requests to the PIO subsystem where update and read privileges are being requested for the page.
RDB_free_global	This parameter displays the current percentage of free global buffers.
RDB_hash_del	Displays the number of hash key deletions from the database's hashed indexes. It includes unique key deletions and duplicate key deletions.
RDB_hash_dup_ins	Displays the number of duplicate hash key insertions in the database's hashed indexes.
RDB_hash_ins	Displays the number of hash key insertions in the database's hashed indexes. It includes unique key insertions and duplicate key insertions.
RDB_lck_conf_unmask	This parameter is incremented by 1 for each modified buffer that is written back to the disk to reduce the possibility of a deadlock when Rdb discovers a lock conflict.
RDB_lock_dem	Displays the number of $ENQ lock requests to demote an existing lock to a lower lock mode. These requests always succeed.
RDB_lock_req	Displays the number of lock requests to new locks. Whether the lock request succeeds or fails, it is included in this count.
RDB_overflow_unmark	This parameter is incremented by 1 for each modified buffer that is written back to disk as a result of a request to read in a new page from disk.
RDB_recoveries	Displays the current number of detached recovery (DBR) processes acting on this database.
RDB_rt_nd_rem	Displays the number of index entries removed from a root node because of deletion of entries within lower-level nodes. If an index consists of only one node, removals from this node are not included in this field; but are included in the leaf removals field.
RDB_rt_nd_ins	Displays the number of index entries inserted into the root index node. The

	number of insertions should be small except when you load a database. If an index consists of only one node, insertions into this node are not included in this field; but are included in the leaf insertion field.
RDB_rt_reads	Displays the number of read QIOs issued to the database root (.RDB) file. Rdb reads the .RDB file when a new user attaches to the database and when an .RDB file control block needs to be updated because of database activity on another OpenVMS cluster node.
RDB_rt_writes	Displays the number of write QIOs issued to the database root (.RDB) file. Rdb writes to the .RDB file when a user issues a COMMIT or ROLLBACK statements. Other events also cause updates to the .RDB file.
RDB_ruj_reads	Displays the number of read QIOs issued to the database recovery unit journal (.RUJ) file. This operation reads before-image records from the .RUJ file to roll back a verb or a transaction.
RDB_ruj_writes	Displays the number of write QIOs issued to the database recovery unit journal (.RUJ) file. This operation writes before-image records to the .RUJ file in case a verb or a transaction must be rolled back. Before-image must be written to the RUJ file before the corresponding database page can be written back to the database.
RDB_trans_cnt	Displays the number of completed database transactions. It is the count of the COMMIT and ROLLBACK statements that have executed.
RDB_txn_unmark	This parameter is incremented by 1 for each modified buffer that is written back to disk as a result of a COMMIT or ROLLBACK statement.
RDBMON_attaches	Displays the number of current attaches to all databases on this system.
RDBMON_databases	Displays the number of open databases on this system.
RDBMON_recoveries	Displays the current number of detached database recovery (DBR) processes on this system.
RMU_stats	This parameter is the collector for all Rdb parameters.

For more information

For more information on this article and to make suggestions and comments for improvements, please email the author.