19c ORACLE Database

Oracle Database 19c: Quality of Service Management

Monitoring and Managing Oracle RAC Database Performance

ORACLE WHITE PAPER | FEBRUARY 2019

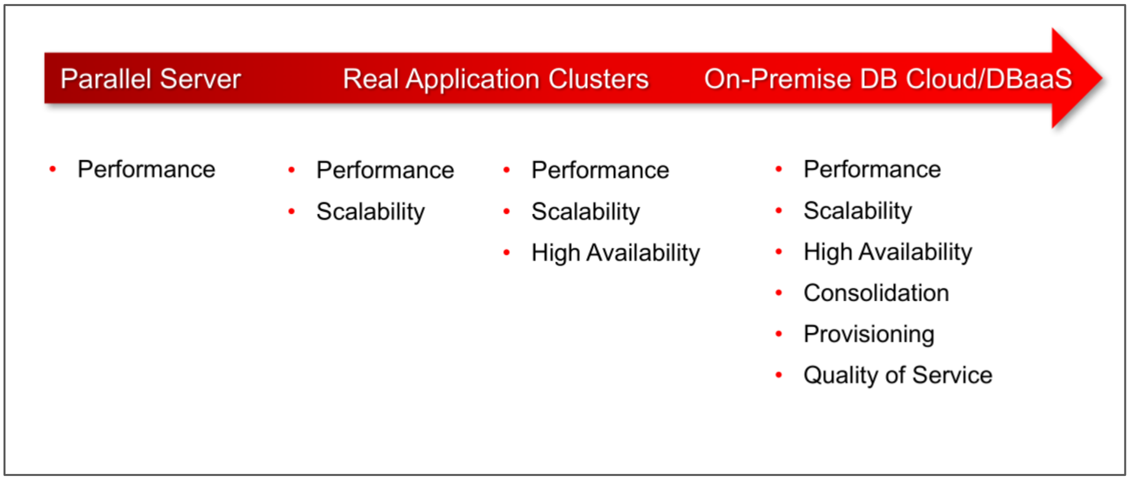
ORACLE

Table of Contents

Introduction

The database is no longer the center of the universe. Such a statement would have been heretical just a short time ago. However, the introduction of the on-premise database cloud and DBaaS has altered the deployment strategy and database management requirements. It is no longer sufficient to plan for simply performance, scalability and high availability. These new deployment models must also consider consolidation, provisioning, patching and quality of service. Oracle Real Application Cluster databases whether in single node form as RAC One Node or multi-node RAC Cluster, provide the level of performance, availability and manageability to be the foundation of modern consolidated on-premise database clouds or Database-as-a-Service deployments.

Figure 1: Evolution of Oracle RAC Database



The ability to manage complex highly available database service deployments in real time is now a common requirement as enterprises adopt a database service-centric deployment model where multiple databases share common physical resources and are no longer silo'd on dedicated hardware. Where resource utilization has improved and IT spend optimized, runtime management complexity has increased. Oracle has addressed this in the Oracle 18c RAC release with Oracle Database Quality of Service Management (QoS) functionality to support all deployment types.

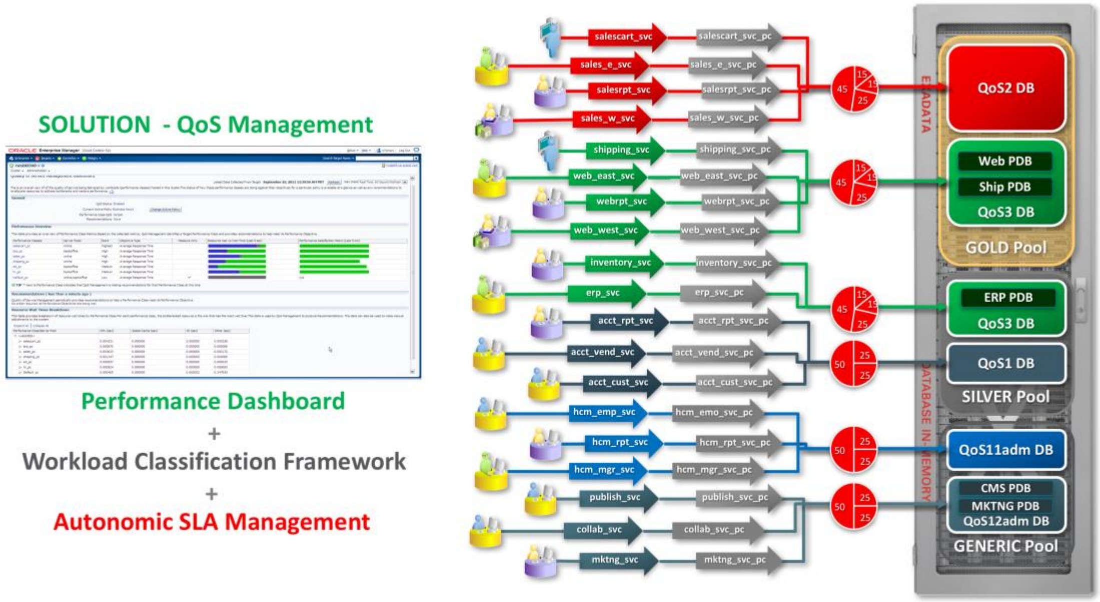


Figure 2: On-Premise Database Cloud Runtime Management

This functionality is included in the Oracle RAC and RAC One Node license and its management interface is integrated into the Enterprise Manager Cloud Control Database Plug-in. Therefore, no additional management packs are required.

Datacenter Runtime Management Requirements

The basic tenants under which Oracle Database QoS Management was developed can be distilled into three statements that must be able to be achieved in real time:

» When resources across the datacenter are sufficient they are continuously deployed to ensure performance and availability objectives will be met.

» When resources are insufficient to meet demand more business-critical objectives will be met at the expense of less critical ones.

» When load conditions severely exceed capacity, resources remain available.

In order to achieve these goals, specific functionality must be built into the entire software stack to include accurate measurement of performance, resource bottleneck analysis, resource trade-off evaluation and online dynamic resource allocation.

In the end, the effectiveness of achieving the above goals is evaluated by each application's performance over time. When examining modern multi-tier applications, it should not be unexpected that most of a transaction's response time is contained in the database tier and its associated storage. This performance can be distilled at a high level into the following simple equation:

Resource Use + Resource Wait = Application Performance

It's important to realize that once an application is deployed, there is almost no ability for online management of its use of resources, as these were the responsibility of design, development, Q/A, and test teams. However, there is the potential for the online management of the amount of time needed to wait for resources, whether these are CPU, memory, or I/O.

Fortunately, the Oracle software stack, especially the database tier, has rich resource management capabilities that have been enhanced in Oracle 18c to facilitate this when used in concert with QoS Management.

Runtime Management Best Practices – The Phases

The best practices for runtime management of an Oracle RAC-based on-premise database cloud or Database-as-a-Service deployment may be applied in discrete phases to gain insight into the actual workloads and their use of resources as well as confidence in setting realistic service level agreements (SLAs) and the ability to manage to them. The four phases that will be discussed are as follows:

1. Plan the deployment
2. Runtime measure the deployment
3. Runtime monitor the deployment
4. Runtime manage the deployment

These phases should be implemented serially, and not combined to accelerate deployment as each captures necessary data that is used in the next phase.

Phase 1: Plan the Deployment

Planning the deployment may start at various points, but for this paper we will assume the deployment is to be a on-premise database cloud offering database services to applications each of which has an importance or criticality to the business that may vary due to calendar or events. This paper is not intended to focus on this particular task, but will introduce its elements.

Since the introduction of Oracle Database 11.2, customers have had a choice of three different cluster database deployment types – administrator-managed, policy-managed or a hybrid of the two. While it is beyond the scope of this paper to explore the pros and cons of each type, as a general rule if the databases to be deployed are 11.2 or greater, then policy-managed should be fully evaluated as it provides the most flexibility as well as deterministic high availability for on-premise database clouds. Please refer to the Appendix for additional information resources.

The next high level step is to determine the service groupings and base sizing. This will involve answering such questions as which services need to run on the same servers, which must be exclusive, or which must be dispersed as well as services that are required to be singletons.

When sizing a on-premise database cloud or DBaaS deployment, the tendency is to make use of multi-threaded CPU cores in order to increase the effective number of CPUs that each database sees in the hope that more databases can be hosted per node. The curves in Figure 3 should be observed as a warning that the level of requests per CPU is significantly reduced before response time goes to infinity and the system is in overload. It should also be noted that predicable performance is no longer achievable because the OS scheduler is now directing database workload scheduling and not the database's resource manager. This results in the CPU cost per database call rising with utilization instead of staying constant as with a single threaded core.

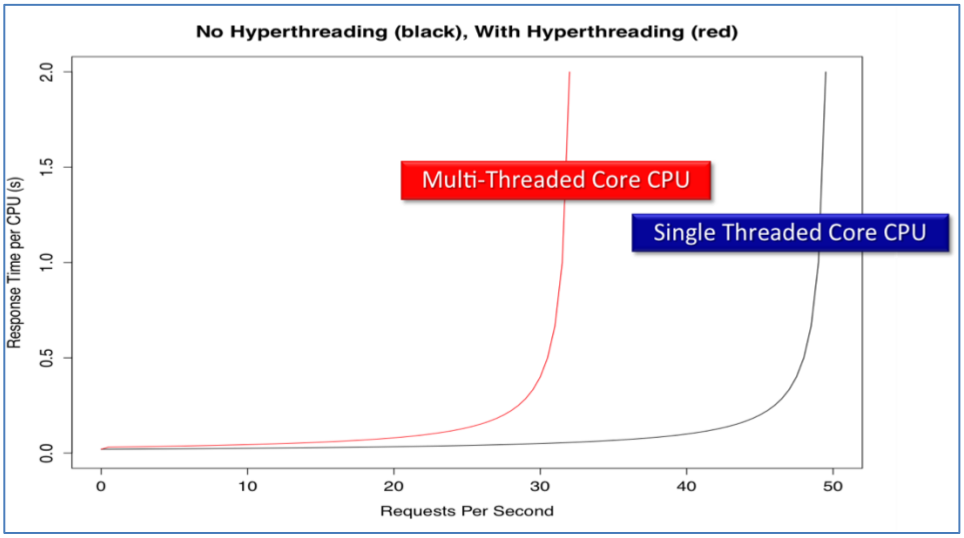


Figure 3: Consolidation Management Problem

Perhaps the most challenging step is to establish the business criticality of each service, answering such questions as:

- Which are the services that need to be online first?
- Which are the services that need to be the last standing?
- Which services can I borrow resources from should a workload surge occur?
- Which services can I shut down should a surge or failure occur?

Fortunately, these questions don't have to have static answers if a policy-managed deployment type is selected, as different business priorities can be expressed in different policies that can be switched in when appropriate. At the same time, legacy databases can coexist within their fenced servers within the Generic server pool yet still be fully supported.

Finally, services need to be group or "classified" into those that need to be tracked for performance and those that simply need to be measured. This classification may be performed by using the QoS Management Policy Editor integrated into Enterprise Manager Cloud Control to create user-defined labels or tags that group workloads for both measurement and assigning performance objectives that can be monitored or managed to as will be described in later phases.

Figure 4 shows where the QoS Management functionality can be found in Enterprise Manager Cloud Control. Note that it is accessed from the Cluster target Administration menu. This is because the scope of management is currently the entire cluster of RAC databases.

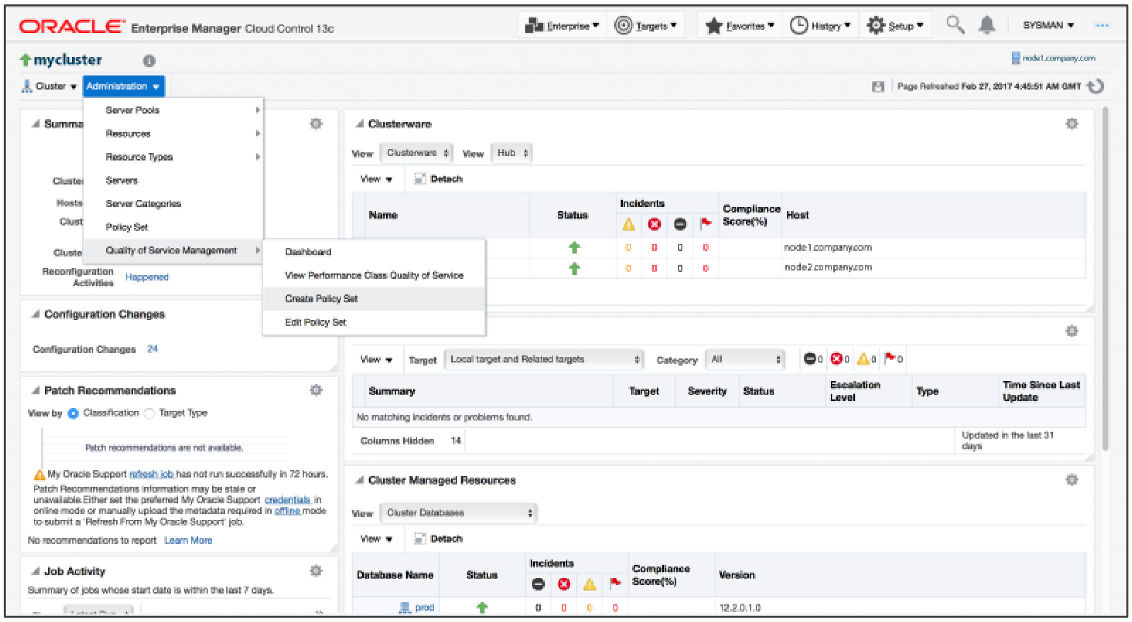


Figure 4: Creating a QoS Management Policy Set

QoS Management generates a default policy set by discovering the entire set of cluster-managed database services currently registered and creating a performance class for each one. This can be seen in Figure 5. Each Performance Class has one or more Classifiers which are the Boolean set expressions shown in the figure.

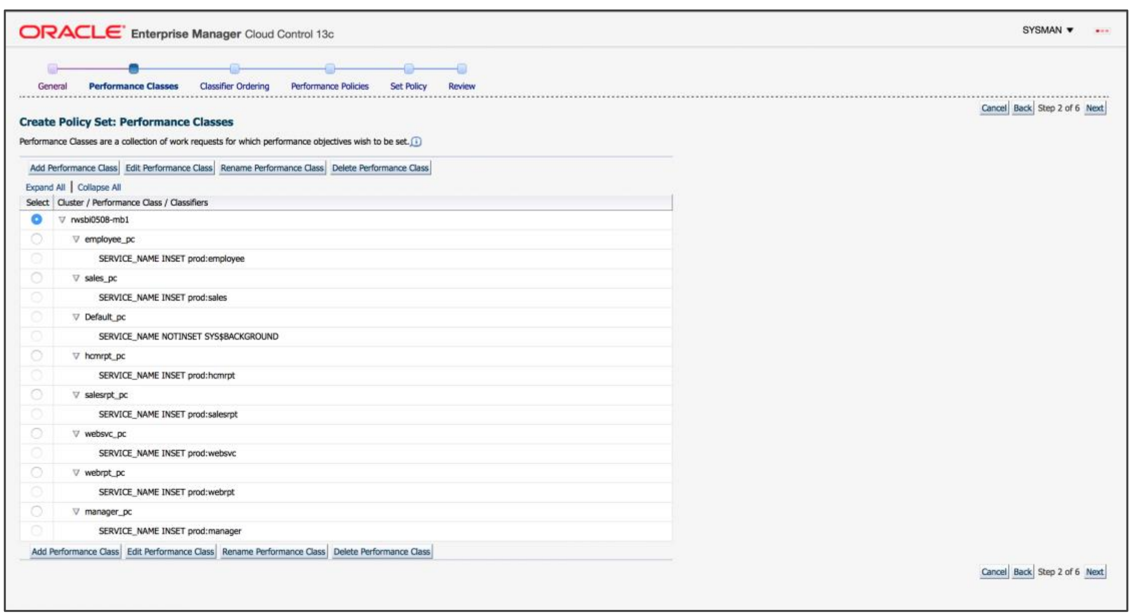


Figure 5: Overview of Performance Classes and Classifiers

In some cases, there may be significantly different types of workloads using the same service. Under this condition additional performance classes may be created that can differentiate the workloads using the database session parameters if Module, Action UserName and Program. An example of differentiating browsing users of the sales service from those who are purchasing is seen in Figure 6 where salescart_pc Performance Class is being created specifying a different database user. Session Module, Action and Program can be populated in the same way.

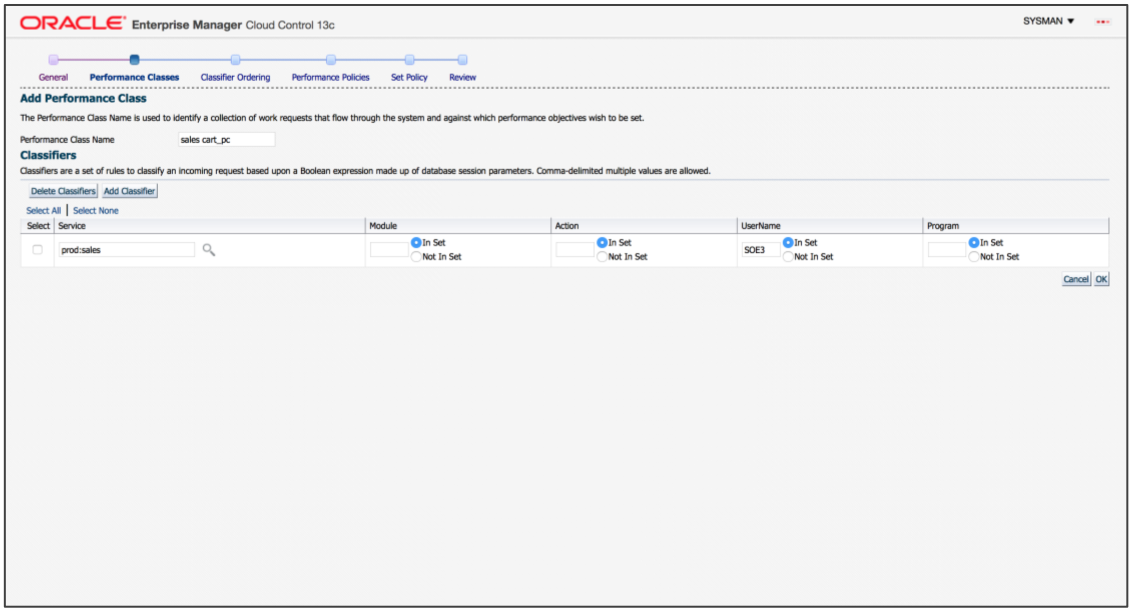


Figure 6: Creating a Performance Class

Should a group of services have similar resource use and performance objectives where it is desired to manage them together, this can be done by adding additional classifiers to a single performance class.

Phase 2: Runtime Measure the Deployment

Once the planning phase is completed, the measurement phase may begin. This is not the same type of measurement that occurs in single application Q/A or testing but in either the production environment or a test one where all databases and these services are running as they would in production. To set up the ability to perform these actual runtime measurements, a measure-only Performance Policy is created in the same QoS Management Policy Editor. This is shown in Figure 7. What distinguishes this policy from others is that no performance objectives are specified and the Measure Only box for each performance class is checked.

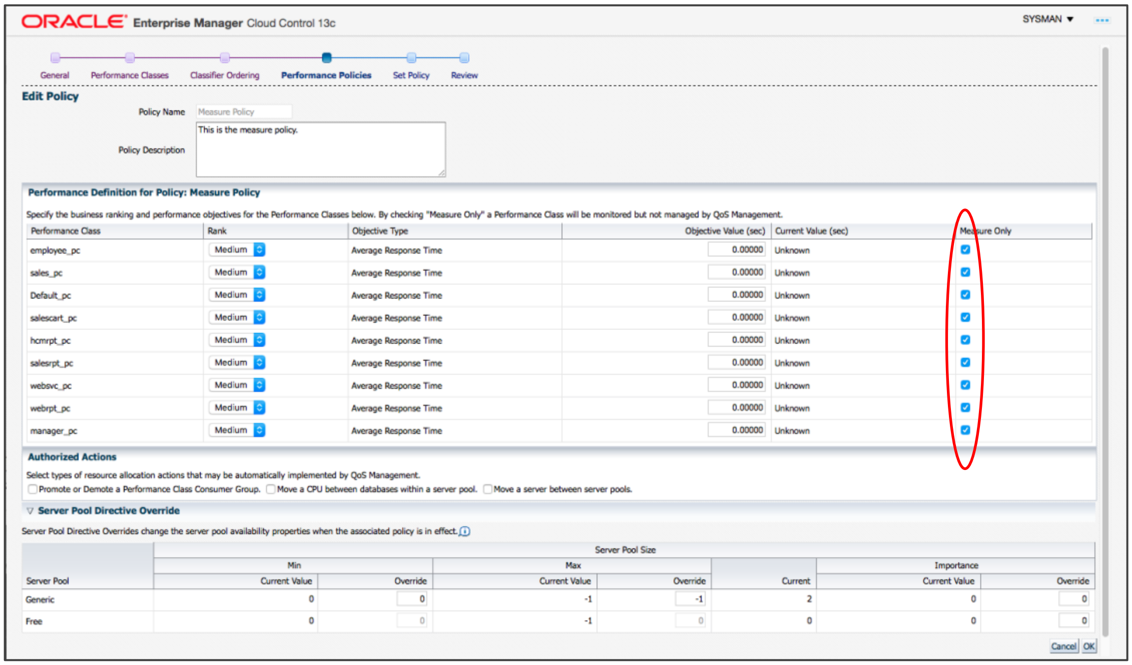


Figure 7: Creating a QoS Management - Measure-Only Policy

Once the Policy Editor wizard is completed and the policy set submitted to the QoS Management server with this measure-only policy activated, the QoS Management Dashboard is displayed as seen in Figure 8. Note that all of the performance classes are listed and the actual server pools where work is occurring are specified.

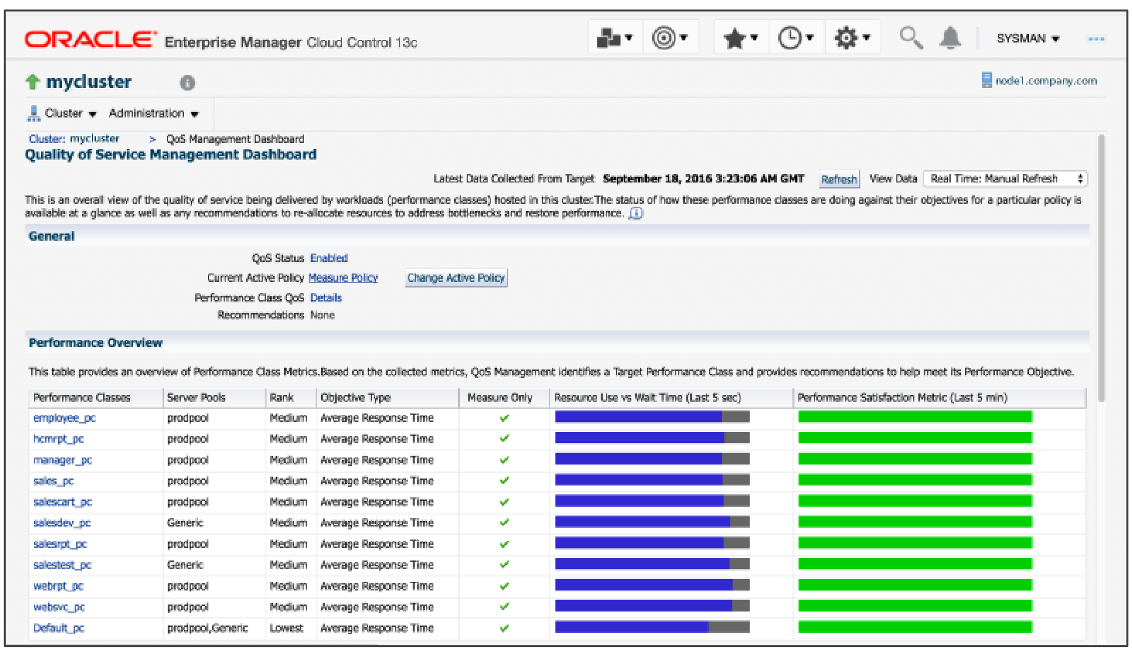


Figure 8: QoS Management Runtime Measure Dashboard

Examining the displayed bar graph for each performance class in Figure 9, two important metrics are displayed. The blue bar shows the actual portion of the response time that represents the use of system resources such as CPU memory and I/O. When hovered over the value displayed in seconds represents the absolute best performance that can be achieved with the deployed resource capability. The gray bar shows the actual portion of the response time that represents the wait for system resources. This time is a function of how busy the system resources are and may be altered via runtime resource management controls. When added together the two represent the actual performance which would be the minimum recommended performance objective set for this performance class given the other workloads.

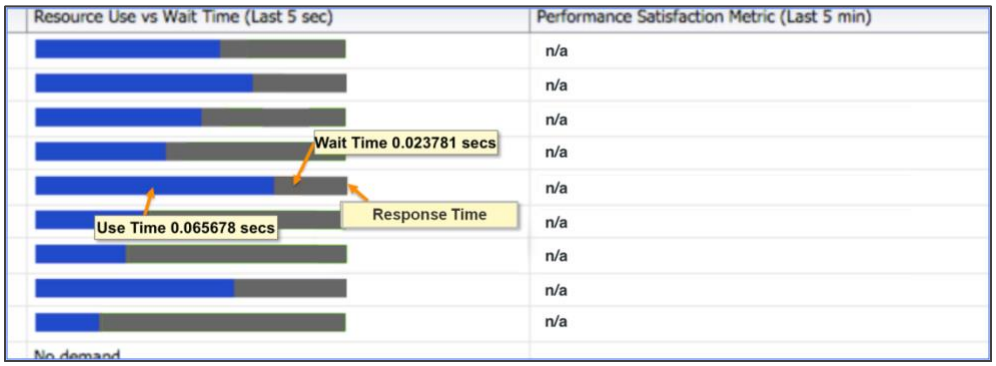


Figure 9: Runtime Measurement Detail

Moving down the QoS Management Dashboard is a table that breaks down the resource wait time for each performance class into four categories as shown in Figure 10 – CPU, Global Cache, IO, and Other.

Performance Class/Server Pool	CPU (sec)	Global Cache (sec)	IO (sec)	Other (sec)
▽ **mycluster**				
▷ salescart_pc	0.091057	0.000000	0.000000	0.000911
▷ manager_pc	0.019457	0.000000	0.000000	0.000242
▷ websvc_pc	0.019811	0.000000	0.000000	0.000397
▷ employee_pc	0.018620	0.000000	0.000000	0.000162
▷ hcmrpt_pc	0.016843	0.000000	0.000000	0.000152
▷ sales_pc	0.032019	0.000000	0.000000	0.000333
▷ salesrpt_pc	0.028000	0.000000	0.000000	0.000145
▷ webrpt_pc	0.017762	0.000000	0.000000	0.000077
▷ Default_pc	0.000000	0.000000	0.000000	0.000000

Figure 10: Resource Wait Detail by Performance Class

These metrics are very useful in understanding whether there are runtime issues beyond simple resource availability with a workload. For example, if Global Cache wait time was the largest and thus the bottleneck, it is most likely that the workload doesn't scale well across more than one instance and its service should be a singleton or the data should be partitioned. If Other wait is the bottleneck, this means that there are SQL issues in the database that should be investigated via an AWR report.

Once this phase is run during all the different workload periods, the metrics will provide a baseline set of minimum performance objective values that may be used in the next phase. It will also provide data that will help determine if base sizing and resources are sufficient to meet the business objectives and whether multiple policies may be useful in meeting these.

Phase 3: Runtime Monitor the Deployment

The third phase is to actually monitor the deployment using performance objectives derived from the previous phase. This requires a different QoS Management performance policy which can be added to the policy set using the Policy Editor in EM Cloud Control. Figure 11 shows an example of such a policy. Note that this monitor policy is quite similar to the previous measure-only policy. The difference is that now actual performance objective values are entered.

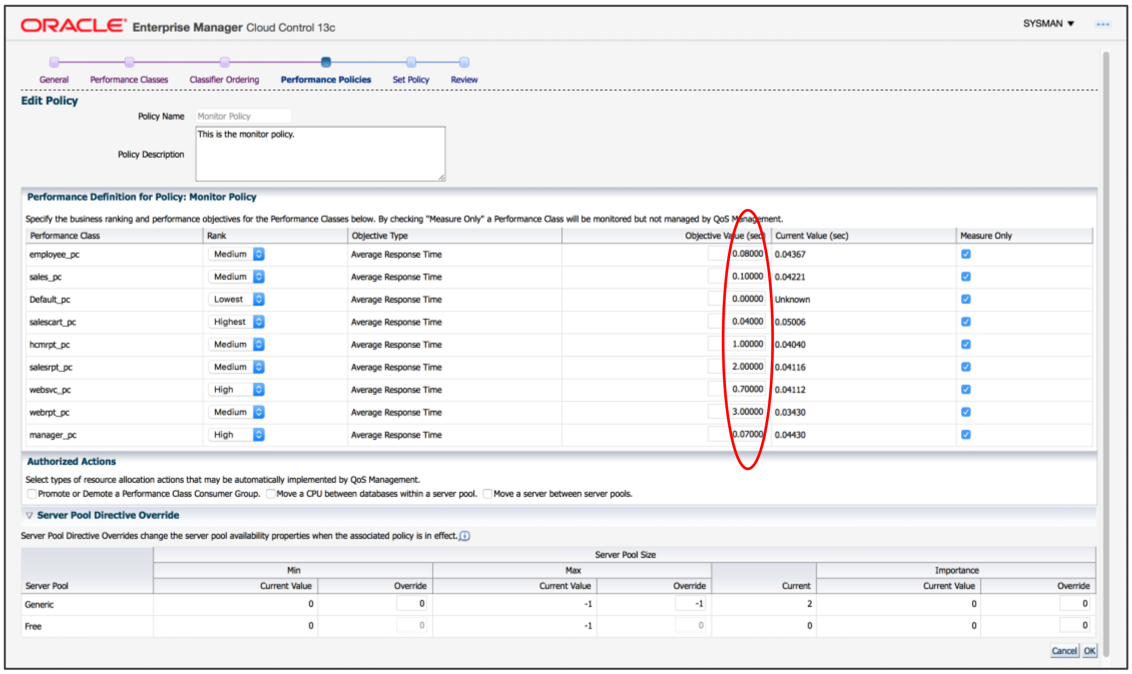


Figure 11: QoS Management - Monitor Policy

Once this policy is submitted and activated, the QoS Management Dashboard changes as displayed in Figure 12. Additional colored bars appear and the Performance Satisfaction Metric column also becomes relevant.

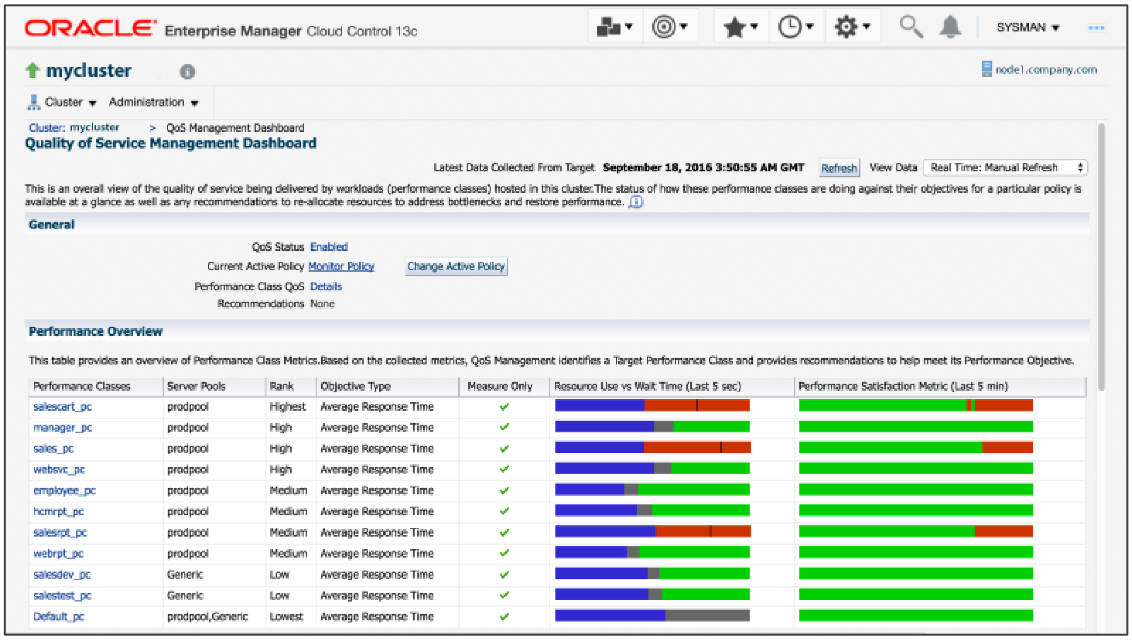


Figure 12: QoS Management Monitor Dashboard

Figure 13 provides a close look at the right-most two columns. The Resource Use vs Wait Time column now has additional bars. Since there is now a performance objective specified in the policy that is represented by the right end of the entire bar. As the response time is the sum of the blue and gray bars, the green bar represents the additional time until the performance objective is met. This is known as the headroom and can be viewed as the amount of shareable resources that can be contributed without exceeding the performance objective. This will be clear in the next phase.

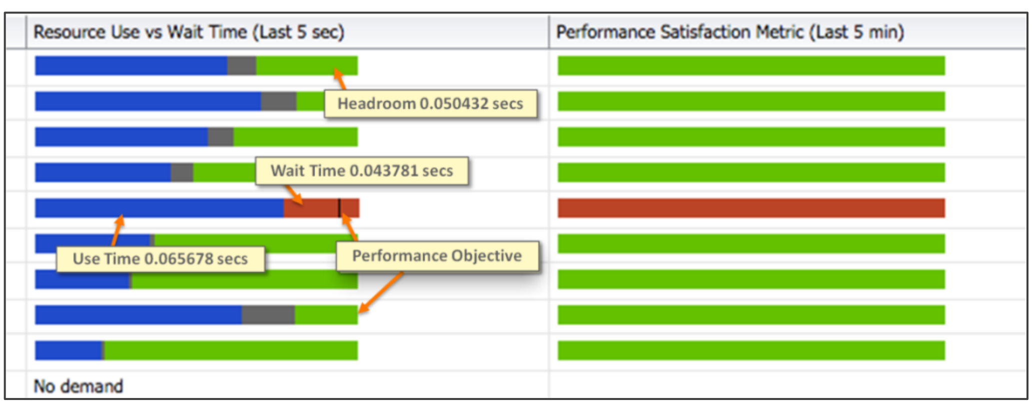


Figure 13: Monitor Measurement Detail

If a performance objective is specified too optimistically and is exceeded the gray bar becomes red to indicate that the wait for resources has caused the response tome of the associated performance class to exceed its performance objective. To provide an indicator of how far it has exceeded, the performance objective is displayed as a blue line with the red bar.

The Performance Satisfaction Metric (PSM) is a unique QoS Management metric as it quantifies in a normalized way how the performance class is doing against its objective. Whether a performance objective is 5ms or 5s, it reports a value between -100% and +100% to indicate the degree response time is meeting or violating its objective. The PSM column is a binary indication as to how the performance classes have been doing against their objectives for the last 5 minutes thereby providing trending information. In this example, the continuous red indicates that the performance objective should be re-evaluated if the load is in expected normal range.

It is obviously not convenient or efficient to require constant monitoring of the QoS Management Dashboard; therefore, support is provided in the EMCC notification system for reporting negative PSMs that persist for user-specified times. Both warning and critical levels can be alerted based upon specified durations for each performance class as shown in Figure 14.

▽ QoS Management - Performance Satisfaction Metric			
▽ Negative PSM Duration (seconds)			
salescart_pc	>	120	300
manager_pc	>	240	480
websvc_pc	>	240	480
All others	>	300	600

Figure 14: EM Negative PSM Duration Notification Setup

This phase will likely need to be executed in an iterative fashion to establish realistic performance objectives. During this process, it may be determined that a single policy is not sufficient to capture the workload phases such as daytime, nighttime, weekends, end of quarter, etc. In that case create multiple policies which can be either switched manually or via a scheduler such as EMCC or CRON and the included qosctl command line utility.

Once this phase has been completed, the decision can be made to move to the final phase. If, for example there are sufficient resources under all demand phases, it may not be necessary to transition to the runtime management phase. However, if this on-premise database cloud is exposed to open workloads such as the Internet, the ability to respond with just-in-time intelligent resource allocation may be critical to maintain business continuity.

Phase 4: Runtime Manage the Deployment

This final phase brings resource agility into runtime management of a on-premise database cloud. Many resource management systems are, in the end, simply issue-response sets of thresholds and rules. While they may work for simple systems, they are inadequate for the complexity of an enterprise database cloud as resources cannot be provisioned on the fly. Instead resource trade-offs and agility within the existing deployment must be able to be evaluated, taking into account business priorities. This is where the expert system in the QoS Management server comes into play.

As with the other phases, this one requires a different policy. Figure 15 is an example of one such policy. It is differentiated from the previous ones in several important ways. First, it takes into account the rank of each performance class which is settable to one of 5 levels. This rank expresses how critical to the business it is for a performance class to continue to meet its objective. Starting with release 12.1.0.2, this ranking also governs the order in which the performance classes' hosting databases are started and allotted real-time LMS processes.

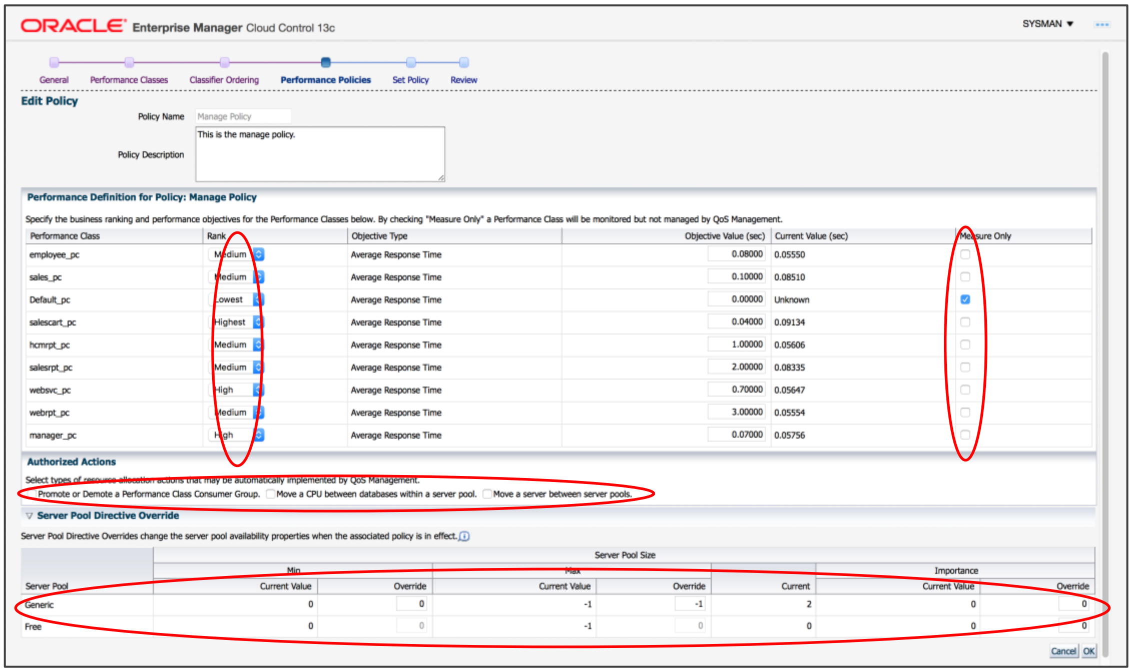


Figure 15: QoS Management – Manage Policy

Second, all performance classes that are to be managed have their Measure Only checkmarks removed. Any that remain checked will be considered donors should resources be required by those classes that are managed.

Third, a list of resource management type actions is offered to authorize QoS Management to autonomously that that action should it be required. This option is not recommended to be enabled until the accuracy and effectiveness of the recommendations and actions have been in production for some time.

Finally, there is the Server Pool Override Directive section which is used when multiple policies require different known base resource allocations. An example would be adding a server to a pool responsible for end of quarter reporting.

Once this type of policy is active, instead of simply being alerted to a performance class problem and viewing the extent of the issue as in the monitor phase, the QoS Management Dashboard displays a recommended action which will have a positive effect on relieving the bottleneck by trading off resources between workloads. Figure 16 shows an example of one such recommendation. In this case the performance class, salescart_pc, is experiencing a resource bottleneck in trying to get access to the CPU. The QoS Management performance model evaluation, determined that moving 15 CPU shares from the prod HCMPDB database to the SALESPDB database will provide more CPU time to the more critical workload and reduce its bottleneck thereby improving response time.

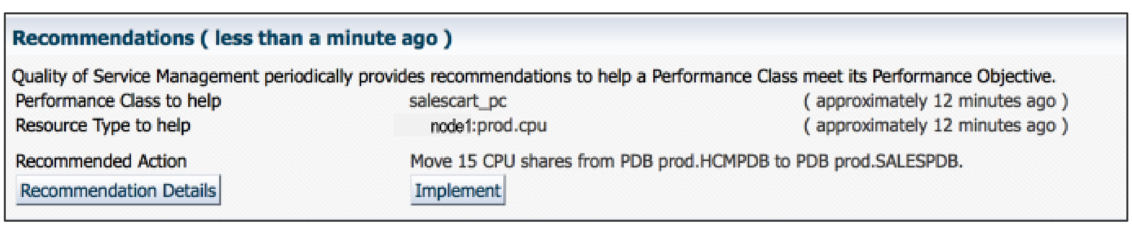
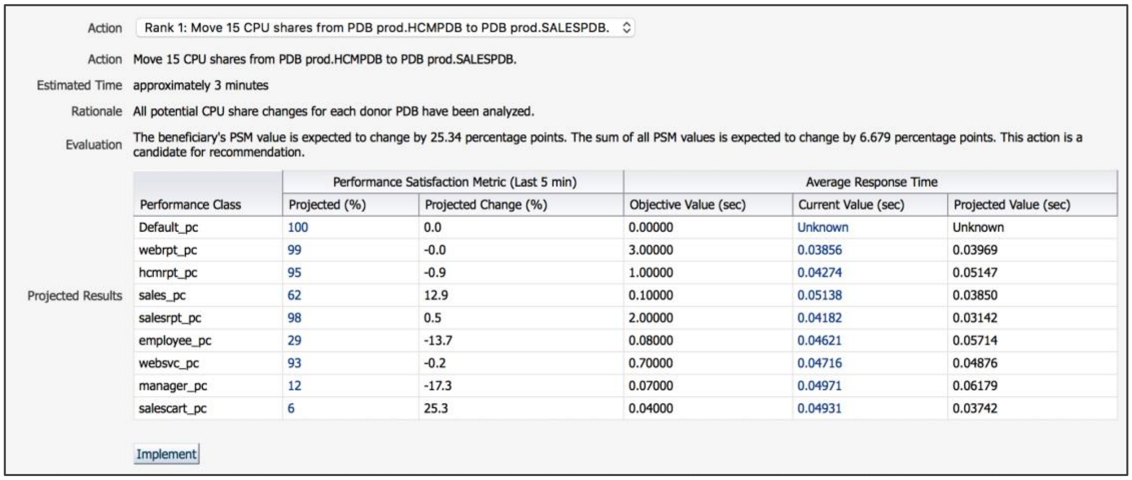


Figure 16: QoS Management Recommendation

If greater insight is desired before clicking the Implement button, full details on the recommendation are available as shown in Figure 17. Since this is a trade-off evaluation a full view of the projected positive and negative impacts to each performance class is presented along with the improvement to the target class.



Action: Rank 1: Move 15 CPU shares from PDB prod.HCMPDB to PDB prod.SALESPDB.

Action: Move 15 CPU shares from PDB prod.HCMPDB to PDB prod.SALESPDB.

Estimated Time: approximately 3 minutes

Rationale: All potential CPU share changes for each donor PDB have been analyzed.

Evaluation: The beneficiary's PSM value is expected to change by 25.34 percentage points. The sum of all PSM values is expected to change by 6.679 percentage points. This action is a candidate for recommendation.

Projected Results

	Performance Satisfaction Metric (Last 5 min)		Average Response Time		
Performance Class	Projected (%)	Projected Change (%)	Objective Value (sec)	Current Value (sec)	Projected Value (sec)
Default_pc	100	0.0	0.00000	Unknown	Unknown
webrpt_pc	99	-0.0	3.00000	0.03856	0.03969
hcmrpt_pc	95	-0.9	1.00000	0.04274	0.05147
sales_pc	62	12.9	0.10000	0.05138	0.03850
salesrpt_pc	98	0.5	2.00000	0.04182	0.03142
employee_pc	29	-13.7	0.08000	0.04621	0.05714
websvc_pc	93	-0.2	0.70000	0.04716	0.04876
manager_pc	12	-17.3	0.07000	0.04971	0.06179
salescart_pc	6	25.3	0.04000	0.04931	0.03742

Implement

Figure 17: Recommendation Details and Performance Projections

QoS Management acts as a governor on existing Oracle resource management functionality. It can adjust CPU shares within a single database to manage schema consolidated services, the number of CPU shares allocated to each PDB in a CDB multitenant database as in the example, the movement of CPUs between databases permitting management of multiple databases sharing the same servers and finally in policy-managed deployments, the ability to move servers between server pools allows for cluster consolidation.

When altering resource allocations, it's critically important to track the performance over time. Either singularly or overall, various graphical metric views can be access from the QoS Management Dashboard. Figure 18 is an overall view of the demand that the cluster is seeing and the obvious surge to the salescart_pc that caused it to violate its objective.

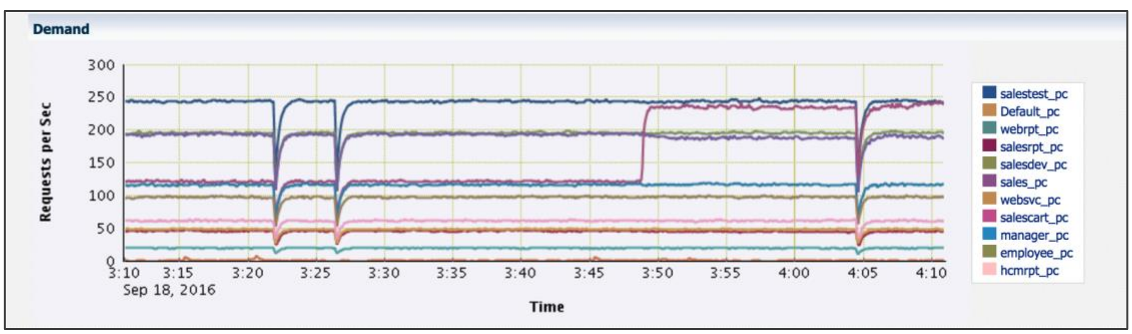


Figure 18: Overall Demand per Performance Class

Figure 19 shows a companion graph of the PSM values during the same period and how the demand surge impacted multiple performance classes. It also shows how through multiple recommendations and reallocations, performance was ultimately restored.

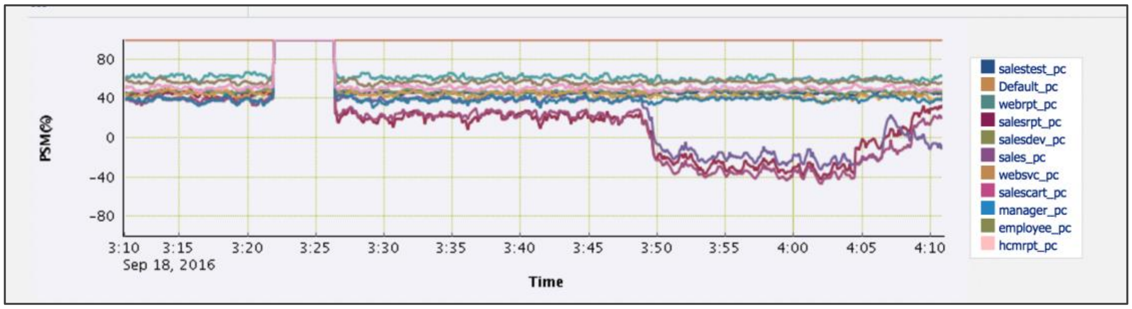


Figure 19: Overall PSM Status per Performance Class

Currently this management phase is targeted at managing workloads whose demand is independent of the response time. These open workloads are the dominant type on the Internet and are particularly difficult to provision for. By clustering resources and making them agile, workload surges can be efficiently accommodated thereby minimizing idle capacity.

Baselining and Tracking Performance

While EMCC provides performance graphs for the most current hour, it is valuable to be able to track performance over days or weeks, especially when determining a baseline set of performance objectives or whether more than one policy is required. Beginning in Oracle 19c, historical data is stored in the Grid Infrastructure Management Repository (GIMR) that resides as part of the grid infrastructure. Reports can be generated in interactive HTML format using the **qosctl -gethistory** command. An example output of the historical performance overview is shown in Figure 20.

Figure 20: Historical Performance Report - Overview

This report can be interacted from a time axis as well as Performance Class dimension. In addition to Performance Satisfaction Metric, Demand and Average Response Time graphs, the Resource Use Time and Resource Wait Time can be explored to provide increased insight into the nature of any performance bottlenecks. This data is also presented for each discrete data point as seen in Figure 21 using your mouse, as well as available for machine processing in JSON format in its data.js file located in the report output directory.

Figure 21: Historical Performance Report - Detail

Conclusion

The desire for database consolidation and a provision-on-demand DBaaS to meet the growing demand without the costs of growing datacenters brings with it new infrastructure functionality and a management paradigm that is both flexible in its resource allocation and deterministic in its operational and failure behavior. There is a learning curve with this type of change, but the ability to implement these types of deployments in managed best practice phases mitigates risk while delivering greater resource utilization and subsequent higher return on investment.

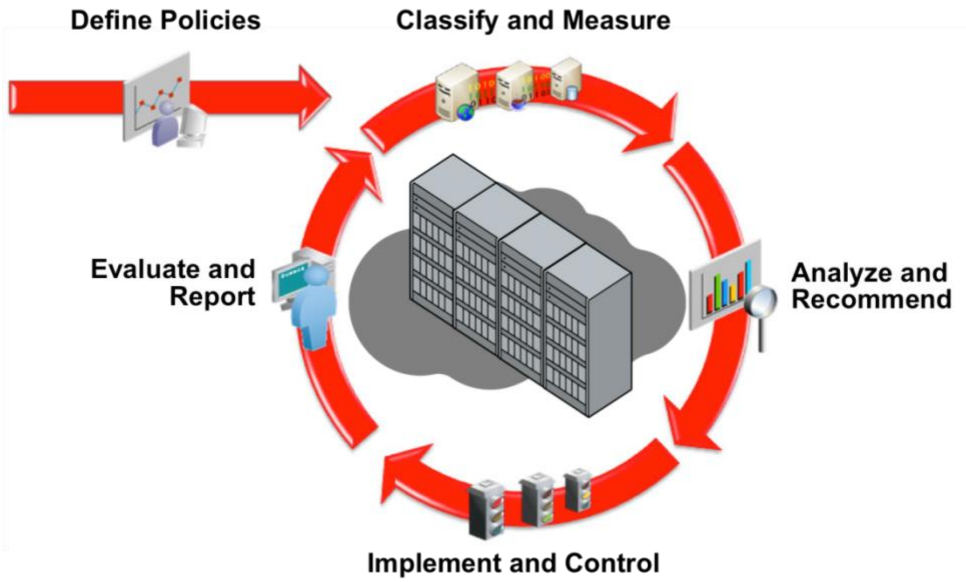


Figure 20: QoS Management in Action

Through its continuous measuring, monitoring and managing of the Oracle RAC on-premise database cloud or DBaaS deployments as illustrated in Figure 20, Oracle Database Quality of Service Management delivers the following key elements for runtime management:

- Cluster-wide real-time dashboard view of all database workloads
- Continuous workload health view
- Real-time resource bottleneck identification
- Workload-specific notifications of performance issues
- Intelligent and targeted bottleneck resolution recommendations
- Action audit trail and performance history

Appendix

Further information on Policy-Managed RAC databases, Clusterware policies and server pools as well as QoS Management is available from the following links:

Oracle Autonomous Health Framework 19c Documentation

Oracle QoS Management on OTN

Oracle QoS Management FAQ

Oracle Database 12c: Why and How You Should Be Using Policy-Managed Oracle RAC Databases

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

Oracle Database 18c: Quality of Service Management - Monitoring and Managing Oracle RAC Database Performance
January 2018
Author: Mark Scardina, Oracle Corporation