OpenVMS Technical Journal

V8, June 2006

Table of Contents

OpenVMS Technical Journal V8

Disaster Tolerance Proof of Concept: HP Volume Shadowing for OpenVMS and Oracle9*i*® RAC over Extended Distances

Melanie Hubbard - OpenVMS Sustaining Engineer
Carlton Davis- OpenVMS Partner Relationship Manager
Craig Showers - OpenVMS Solutions Center Manager
John Andruszkiewicz - OpenVMS Engineer
Keith Parris - Systems Engineer

Overview

In today's high-risk world, the goal of information technology's contribution to business continuity is to ensure that an enterprise's data, transactions, and IT infrastructure continue to be available, regardless of adverse conditions. This is generally known as *disaster tolerance*. These conditions can go as far as the loss of an entire data center. When considering various business continuity implementations, an enterprise must take into account the potential cost of losing its IT capabilities. A generally recommended implementation for mitigating this risk is to deploy at least two geographically separate data centers, often at distances measured in hundreds of miles.[1]

The measure of a disaster-tolerant environment is its ability to keep working as the database or operating system recovers from any failure. The overall goal for this project is to demonstrate that HP Volume Shadowing for OpenVMS along with Oracle9*i*® RAC, as well as the connectivity technology from LightSand and Digital Networks, can ensure high availability and data integrity over a variety of distances, including those formally supported by OpenVMS Clusters, as well as distances longer than what is currently supported.[2]

---

[1] *Interagency Paper on Sound Practices to Strengthen the Resilience of the U.S. Financial System* Federal Reserve System [Docket No. R-1128], Department of the Treasury Office of the Comptroller of the Currency [Docket No. 03-05], Securities and Exchange Commission [Release No. 34-47638; File No. S7-32-02]. Refer to http://www.sec.gov/news/studies/34-47638.htm for more information.

[2] The current maximum supported distance for OpenVMS is 250 km (150 mi); the OpenVMS Disaster Tolerant Cluster Solution (DTCS) services package generally supports internode distances up to 500 mi (800 km) but can span greater distances depending on requirements and

These distances can be thought of as representing what is sometimes referred to as a Local or Campus configuration (0 mi/0 km – 18 mi/30 km between nodes), a Regional configuration (upwards of 372 mi/600 km between nodes), and a Geographically Extended configuration (upwards of 621 mi/1000 km between nodes). This proof of concept validates that long distance disaster-tolerant Oracle RAC on OpenVMS systems can be successfully deployed.

For more information on LightSand Communications, Inc., Digital Networks, and Oracle RAC, see Appendix B.

Business Needs

When data security and availability are critical to their success, enterprises require a computing solution that protects their information systems from disasters such as power outages, earthquakes, fires, floods, or acts of vandalism. The effects of a disaster range from temporary loss of availability to outright physical destruction of an entire facility and its assets. In the event of such a disaster, the system design must allow organizations to shift their information-processing activities to another site as quickly as possible, with minimal loss of function and data. Therefore, procedures for disaster recovery must be predictable, well-defined, and immune to human error.

Disaster tolerance is characterized by a short recovery time (low Recovery Time Objective -- RTO) and avoidance of data loss (low Recovery Point Objective -- RPO). In a disaster-tolerant system based on this approach, redundant, active servers and client interconnects are located at geographically separated sites. Should the environment at one site suffer a disaster, applications that were running at the now-disabled site can continue to run at the surviving site.

While the cost of lost IT capabilities is the key consideration, affordability is also a factor. Deploying the network infrastructure over a long distance, depending on its configuration, can be prohibitively expensive, especially when it has to handle cluster traffic, Oracle RAC operations, and data replication. This is an analysis that should be taken by any organization considering a disaster-tolerant IT environment. Cost-benefit, however, is not the subject of this proof of concept. Rather, it is intended to demonstrate the operational capability of one specific and popular architecture over distances that are considered appropriate for disaster tolerance.

Business Solution

HP OpenVMS, Digital Networks, and LightSand Communications, Inc. have joined to test Oracle9*i* RAC in disaster-tolerant configurations over a variety of distances using Volume Shadowing[3] technology under the control of an OpenVMS host system. The goal for this proof of concept was to observe and record the behavior of an Oracle9*i* RAC server on a clustered OpenVMS system using Volume Shadowing across extended distances. The distances tested are 0 mi (0 km), 372 mi (600 km), and 621 mi (1000 km).

OpenVMS cluster uptimes are often measured in years. Active-active cluster technology at both the operating system and database level (Oracle RAC) and cluster-aware applications provide the capability to shut down individual systems for proactive reasons with zero application availability impact. *Multi-site* clusters can include the capability to proactively shut down an entire site with zero application availability impact. No applications or end-user connections would need to fail over as they would already be running on other systems.

In the event of unexpected or unplanned outages, only the connections to that single system would be impacted from an availability perspective. Other servers would continue to process their existing connections. The failed server connections would then automatically reconnect to other servers that are already running in the cluster. Because the applications and storage devices are already running

circumstances. Basic cluster protection and data protection can be between distances as great as 60,000 mi (97,000 km), however, the latency at greater distances may not be acceptable for specific customer implementations.

[3] HBVS uses RAID-1 technology. Refer to http://h71000.www7.hp.com/openvms/products/volume-shadowing/index.html for more information.

and available on the shared file system on other servers, the failed server connections fail over extremely quickly.[4]

The connectivity technology used in this proof of concept provides an example of a cost-effective *multi-site* network configuration. There are multiple ways to simulate a long-distance cluster without the actual expense of real long-distance inter-site links; such as delaying packets and thus simulating distance via latency. The Spirent/AdTech product, the Shunra STORM network emulator, or a PC running the free NIST Net software from the National Institutes of Technology can be used to delay traffic. Most of these tools can delay IP traffic only, not LAN traffic in general.[5] Since OpenVMS Clusters use the SCS (sometimes called SCA) protocol on LANs and SCS is not an IP family protocol, you need a method to convert SCS traffic into IP format.

One method of converting SCS traffic is to use routers (such as Cisco) running a Layer 2 Tunneling Protocol version 3 (L2TPv3) tunnel to encapsulate LAN traffic (including SCS) and send it over IP. The ability of the LightSand boxes to bridge both Fibre Channel and LAN (including SCS) traffic over IP provided a solution for encapsulating SCS without the need for routers and an L2TPv3 tunnel. Thee latter, less complicated configuration was chosen for our series of tests.

Business Summary

Results demonstrate that, under testing, Oracle RAC in conjunction with Volume Shadowing works across extended distances. All components continued to function without interruption over distances of up to 621 mi (1000 km). Longer distances could be used but every environment will be different depending on such factors as workload, transaction size, required transaction rate, user response requirements, database size, and site hardware. Each site would need to evaluate the effect of latency on their application and make decisions based on their current functional needs.

The chart for Test 4E7, Host-Based Minimerge (HBMM) Forced Merge, shows a noticeably longer time to return to steady state than the other HBMM tests. This was due to the fact that, although both nodes had bitmaps enabled, only one bitmap was active and it was running on the node that crashed. Effective disaster-tolerant configurations must ensure that multiple active bitmaps are defined. This is done using the DCL command shown under Test 4E1, and fully documented in the Volume Shadowing/Host Based Minimerge documentation referenced in that test description.

Test timing data for all distances tested shows that the time it takes for a Shadow Set member to return to steady state using HBMM (Host Based Minimerge) is significantly less than that for a full merge; therefore, HBMM is the best option to return members to steady state.

Test Environment

The hardware and software were configured in a two-node disaster-tolerant configuration that would be typical of an enterprise meeting the stringent requirements for a live remote datacenter DT environment. This included commercially configured servers, storage, and infrastructure components that are frequently seen in production environments. In addition, a network delay component was added to realistically emulate network latency behavior over the test distances.

See Appendix A for a list of the selected hardware and software configurations for this proof of concept. Figure 1 shows the hardware configuration.

[4] Refer to www.hp.com/go/openvms/availability for more information.

[5] We believe that with a new release of firmware, the Shunra box can delay ordinary LAN traffic, not just IP. If LAN traffic in general can be delayed, then a simpler test environment is possible – you merely need to connect the delay box between a couple of LANs. Storage traffic can be put on the LAN and delayed through the same box, either via MSCP-serving or using a technique called "SAN Extension." MSCP-serving can be used for remote storage access in an OpenVMS Cluster. If it is desired to bridge Fibre Channel SANs between sites instead of using MSCP Serving as the primary remote disk access method, then any of the SAN Extension boxes on the market which can bridge Fibre Channel over a LAN (or an IP network running on a LAN) can be used.

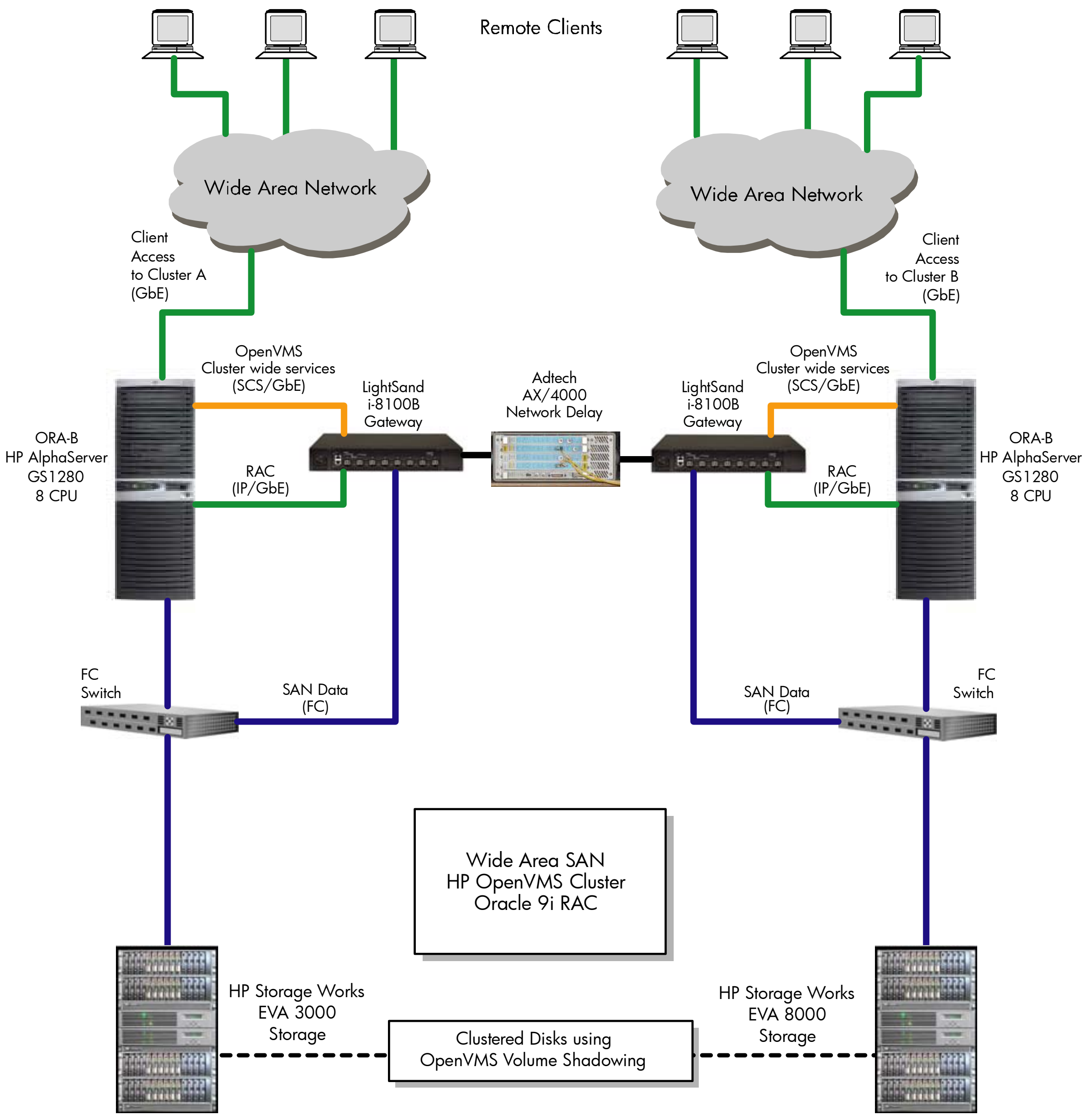

Remote Clients
Wide Area Network
Wide Area Network
Client Access to Cluster A (GbE)
Client Access to Cluster B (GbE)
OpenVMS Cluster wide services (SCS/GbE)
OpenVMS Cluster wide services (SCS/GbE)
LightSand i-8100B Gateway
Adtech AX/4000 Network Delay
LightSand i-8100B Gateway
ORA-B HP AlphaServer GS1280 8 CPU
ORA-B HP AlphaServer GS1280 8 CPU
RAC (IP/GbE)
RAC (IP/GbE)
FC Switch
SAN Data (FC)
SAN Data (FC)
FC Switch
Wide Area SAN HP OpenVMS Cluster Oracle 9i RAC
HP Storage Works EVA 3000 Storage
HP Storage Works EVA 8000 Storage
Clustered Disks using OpenVMS Volume Shadowing

Figure 1 - Hardware Configuration Diagram

Test Description

The testing goal was to provide a specific, repeatable work load (via Swingbench) running on various Volume Shadowing configurations with the OpenVMS Cluster nodes separated by various distances and record the results of the work load. The work load generated by Swingbench simulated typical order-entry functions of 300 remote clients and generating 600,000 transactions (sometimes referred to as 600K transactions).

The testing goal was accomplished by having a baseline copy of the system and database created before the tests were started, which was restored at the beginning of each test run to ensure a common starting point for each test configuration. Also, the network delay hardware was set for the distance that was being tested before each test run.

The data collected for analysis from each run included:

- The output of the Swingbench benchmark software, which includes the total number of transactions, total time to completion, as well as the minimum, maximum, and average response times for each type of transaction.
- The output of T4, an OpenVMS timeline tracking tool, which captures and consolidates important OpenVMS system performance statistics.
- The output of system- and device-specific DCL commands, which shows the current state of those devices.

Note that no performance tuning was done at any time during these tests nor was it ever intended to be done. Tuning recommendations are typically very specific to an application and the system it is running on, and that was not part of this project.

Initial tests were done and a decision made that the site identifier for the disks should be defined as local to their node, and that each node would have a unique identifier. This ensures that the read cost to the local Volume Shadow members is optimized. This is the default behavior of Volume Shadowing. Refer to the HP Volume Shadowing[6] for OpenVMS documentation for complete details.

There are a countless number of test variations which can be run, using varying numbers of local and remote members of each Shadow Volume Set. We chose the following tests to represent an appropriate cross-section of those variations.

The following tests were performed for various Shadow Set Member (SSM) configurations and their results were recorded.

- Steady State: Two Shadow Set Members (Test 7A1)
- Full Copy State: Two Shadow Set Members (Test 2B1)
- Minicopy Logging/Recovery: Two Shadow Set Members (Test 2C1)
- Full Forced Merge State: Two Shadow Set Members (Test 4D2)
- Host-Based Minimerge (HBMM) Demand Merge: Two Shadow Set Members (Test 4E1)
- Host-Based Minimerge (HBMM) Forced Merge: Two Shadow Set Members (Test 4E7)
- Host-Based Minimerge (HBMM) Demand Merge: Three Shadow Set Members (Test 4F1)
- Host-Based Minimerge (HBMM) Forced Merge: Three Shadow Set Members (Test 6F5)

Test Results

For each test, two tables are shown. The first table shows the total time (in `hh:mm:ss` format) to complete 600,000 transactions for each distance specified. The distances are emulated by adding known network latency or delays (in milliseconds) in the data transmission between the nodes.[7] The second table shows the time it takes to return the shadow set member to steady state (fully copied or merged).

The distance equivalents are as follows:

```
00 ms = 0 mi/0 km   03 ms = 372 mi/600 km   05ms = 621 mi/1000 km
```

Test 7A1. Steady State: Two SSMs

This test consists of a two-member Shadow Set Volume, with one disk local to each node and one disk remote to each node. The test is started only after each disk is 100% copied (also known as in a Steady State) and is a full member of the Shadow Volume set.

[6] Refer to http://h71000.www7.hp.com/openvms/products/volume-shadowing/index.html for more information.

[7] Although the speed of light is used as the general speed of data through a network, there is some minor additional delay due to the laws of physics. Thus, the speed of data across the network is not the theoretical speed of light 0.66 ms/100 km per round trip, but instead a slightly slower (and generally accepted) speed of 0.5 ms/100 km per round trip. Thus, a 5 ms delay emulates a distance of 1000 km between nodes.

The following table shows the amount of time to complete 600,000 transactions for the specific delays and distances.

Simulated distance (mi/km)	**Network Latency (ms)**	**7A1 – Time for Completion (HH:MM:SS)**
00 /00	00	1:32:04
372 /600	03	5:00:00
621 /1000	05	8:01:11

The following table shows the amount of time to complete a full (100%) disk copy and return to steady state before the transactions are started.

Simulated distance (mi/km)	**Network Latency (ms)**	**DSA10 – 5 GB – 28% full (HH:MM:SS)**	**DSA14 – 10 GB – 54% full (HH:MM:SS)**	**DSA20 – 5 GB – 14% full (HH:MM:SS)**
00 /00	00	00:10:00	00:50:00	01:20:00
372 /600	03	00:10:00	00:50:00	01:25:00
621 /1000	05	00:15:00	01:35:00	02:15:00

Test 2B1. Full Copy State: Two SSMs

This test consists of a two-member Shadow Set Volume, with one disk local to each node and one disk remote to each node. One disk is mounted and is a full member of the set. The test is started and then the second member is mounted.

The following table shows the amount of time to complete 600,000 transactions for the specific delays and distances.

Simulated distance (mi/km)	**Network Latency (ms)**	**2B1 – Time for Completion (HH:MM:SS)**
00 /00	00	1:40:16
372 /600	03	5:57:39
621 /1000	05	8:24:58

The following table shows the amount of time it took for the disks to become full members (this is for a copy operation starting after the test is running) and return to steady state.

Simulated distance (mi/km)	**Network Latency (ms)**	**DSA10 – 5 GB – 28% full (HH:MM:SS)**	**DSA14 – 10 GB – 54% full (HH:MM:SS)**	**DSA20 – 5 GB – 14% full (HH:MM:SS)**
00 /00	00	00:02:00	00:12:30	00:07:54
372 /600	03	00:14:12	01:02:23	00:25:42
621 /1000	05	01:08:47	02:23:47	01:13:47

Test 2C1. Minicopy Logging/ Recovery: Two SSMs

This test consists of a two-member Shadow Set Volume, with one disk local to each node and one disk remote to each node. Both members are 100% copied (in a Steady State). At that time, the remote member is removed from the Shadow Set volume (using the OpenVMS DCL command:

```
DISMOUNT <Device-name> /POLICY=MINICOPY)
```

The test is then started and run until completion. The remote member is then mounted and a minicopy takes place.

The following table shows the amount of time to complete 600,000 transactions for the specific delays and distances.

Simulated distance (mi/km)	**Network Latency (ms)**	**2C1 – Time for Completion (HH:MM:SS)**
00 /00	00	1:24:18
372 /600	03	5:27:48
621 /1000	05	8:50:45

The following table shows the amount of time to return the second shadow volume member to steady state at the end of the task completion using the `MINICOPY` policy.

Simulated distance (mi/km)	**Network Latency (ms)**	**DSA10 – 5 GB – 28% full (HH:MM:SS)**	**DSA14 – 10 GB – 54% full (HH:MM:SS)**	**DSA20 – 5 GB – 14% full (HH:MM:SS)**
00 /00	00	00:01:48	00:00:04	00:01:32
372 /600	03	00:04:01	00:00:24	00:04:32
621 /1000	05	00:07:33	00:01:01	00:07:05

Test 4D2. Full Forced Merge State: Two SSMs

This test consists of a two-member Shadow Set Volume, with one disk local to each node and one disk remote to each node. Both members are 100% copied (in a steady state). The merge state can be entered when a system crashes (using the VU mounted crash or by using a DCL command requesting a merge). The system crash will force Oracle9*i* RAC to fail over to the other system in the cluster; therefore, Oracle TAF must be enabled and functioning. After the start of the test, one system is crashed, which forces a demand merge of the remaining Shadow Set members.

The following table shows the amount of time to complete 600,000 transactions for the specific delays and distances.

Simulated distance (mi/km)	**Network Latency (ms)**	**4D2 – Time for Completion (HH:MM:SS)**
00 /00	00	1:22:57
372 /600	03	5:03:08
621 /1000	05	8:49:55

The following table shows the amount of time to resolve the demand merge on the remaining node and return to steady state.

Simulated distance (mi/km)	**Network Latency (ms)**	**DSA10 – 5 GB – 28% full (HH:MM:SS)**	**DSA14 – 10 GB – 54% full (HH:MM:SS)**	**DSA20 – 5 GB – 14% full (HH:MM:SS)**
00 /00	00	00:08:51	00:16:36	00:23:54
372 /600	03	00:09:19	00:17:20	00:27:45
621 /1000	05	00:17:01	00:35:57	00:47:39

Test 4E1. Host-Based Minimerge (HBMM) Demand Merge: Two SSMs

The E and F tests are based on OpenVMS Volume Shadowing Host-Based Minimerge functionality. The HBMM policy creates a selected number of bitmaps on selected systems in the cluster. For this test, `SYS1` is the Active Instance and `SYS2` is the Passive Instance. A policy is assigned to each of the Shadow Set Volumes. The Active Instance manages all Minimerge recovery operations by virtue of the following HBMM policy declarations shown below.

Before the test started, a run was done with an extremely high threshold value, to help establish an appropriate reset threshold. We determined for our testing purposes that a reset value of 1,000,000 blocks was appropriate for our system. This value reflected approximately 50% of the writes done during a test run. Refer to http://h71000.www7.hp.com/news/hbmm.html for more information. A policy with that value was created and assigned to the Shadow Volumes using the following DCL commands:

```
$! Create a policy named RECOVERY_on_ACTIVE_NODES_DSAn
SET SHADOW /POLICY=HBMM=(master_list=(SYS1,SYS2),
reset_threshold=1000000) /NAME=RECOVERY_on_ACTIVE_NODES_DSAn

$! Associate a policy named HBMM_DSAn with DSAn:
SET SHADOW DSAn:/POLICY=HBMM=RECOVERY_on_ACTIVE_NODES_DSAn
```

Test 4E1 consists of a two-member Shadow Set Volume, with one disk local to each node and one disk remote to each node. Both members are 100% copied (in a Steady State). The test run was started and before completion, a demand merge was initiated with the following OpenVMS DCL command:

```
SET SHADOW/DEMAND_MERGE DSAn:
```

The following table shows the amount of time to complete 600,000 transactions for the specific delays and distances.

Simulated distance (mi/km)	**Network Latency (ms)**	**4E1 – Time for Completion (HH:MM:SS)**
00 /00	00	1:25:47
372 /600	03	5:07:05
621 /1000	05	8:34:04

The following table shows the amount of time for the demand merge to be completed and return to steady state.

Simulated distance (mi/km)	**Network Latency (ms)**	**DSA10 – 5 GB – 28% full (HH:MM:SS)**	**DSA14 – 10 GB – 54% full (HH:MM:SS)**	**DSA20 – 5 GB – 14% full (HH:MM:SS)**
00 /00	00	00:00:28	00:01:39	00:00:49
372 /600	03	00:00:07	00:00:05	00:00:05
621 /1000	05	00:00:29	00:01:36	00:00:47

Test 4E7. Host-Based Minimerge (HBMM) Forced Merge: Two SSMs

This test is similar to test 4E1 in all ways, except that instead of starting the merge by a DCL command, the merge is started by a forced crash of one of the nodes at some point after the start of the test run.

The following table shows the amount of time to complete 600,000 transactions for the specific delays and distances.

Simulated distance (mi/km)	**Network Latency (ms)**	**4E7 – Time for Completion (HH:MM:SS)**
00 /00	00	1:23:54
372 /600	03	5:41:45
621 /1000	05	8:25:45

The following table shows the amount of time for the forced merge to be completed and return to steady state.

Simulated distance (mi/km)	**Network Latency (ms)**	**DSA10 – 5 GB – 28% full (HH:MM:SS)**	**DSA14 – 10 GB – 54% full (HH:MM:SS)**	**DSA20 – 5 GB – 14% full (HH:MM:SS)**
00 /00	00	00:09:02	00:18:21	00:17:36
372 /600	03	00:08:19	00:15:03	00:16:27
621 /1000	05	00:26:21	00:47:47	00:17:13

Test 4F1. Host-Based Minimerge (HBMM) Demand Merge: Three SSMs

This test is similar to test 4E1 in all ways, except that there are now two local full Shadow Set Members and one remote full Shadow Set Member, making a total of three disks instead of two. A demand merge is initiated by DCL command.

The following table shows the amount of time to complete 600,000 transactions for the specific delays and distances.

Simulated distance (mi/km)	**Network Latency (ms)**	**4F1 – Time for Completion (HH:MM:SS)**
00 /00	00	1:28:35
372 /600	03	5:36:09
621 /1000	05	8:51:06

The following table shows the amount of time for the demand merge to be completed and return to steady state.

Simulated distance (mi/km)	Network Latency (ms)	DSA10 – 5 GB – 28% full (HH:MM:SS)	DSA14 – 10 GB – 54% full (HH:MM:SS)	DSA20 – 5 GB – 14% full (HH:MM:SS)
00 /00	00	00:00:48	00:01:30	00:00:29
372 /600	03	00:00:06	00:00:05	00:00:05
621 /1000	05	00:00:40	00:01:57	00:00:06

Test 6F5. Host-Based Minimerge (HBMM) Forced Merge: Three SSMs

This test is similar to test 4F1 in all ways, except that instead of starting the merge by a DCL command, the merge is initiated by a forced crash of one of the nodes at some point after the start of the test run. Also, there is one local full Shadow Set member and two remote full Shadow Set members.

The following table shows the amount of time to complete 600,000 transactions for the specific delays and distances.

Simulated distance (mi/km)	Network Latency (ms)	6F5 – Time for Completion (HH:MM:SS)
00 /00	00	1:23:31
372 /600	03	5:42:00
621 /1000	05	8:03:21

The following table shows the amount of time for the forced merge to be completed and return to steady state.

Simulated distance (mi/km)	Network Latency (ms)	DSA10 – 5 GB – 28% full (HH:MM:SS)	DSA14 – 10 GB – 54% full (HH:MM:SS)	DSA20 – 5 GB – 14% full (HH:MM:SS)
00 /00	00	00:00:13	00:00:20	00:00:21
372 /600	03	00:00:11	00:00:14	00:00:07
621 /1000	05	00:00:09	00:00:07	00:00:10

Appendix A

This appendix lists the selected hardware and software configurations for this proof of concept.

Hardware Configuration

This section lists the hardware configuration used for this proof of concept.

Servers

- 2 AlphaServer GS1280 (EV7, 1.5GHz) systems
- 8 CPU
- 8 GB memory

Storage

- EVA3000 and EVA8000 configured as RAID 1 volumes
- KGPSA disk fibre cards, PCI 133 MHz bus, 2 GB fibre speed
- DSGGB SANswitch connecting the disk fibre, SANswitch 2/16

Network

- LightSand i-8100B gateway
- Adtech AX/4000 network delay simulator[8]
- DEGXA Network Interface Card (NIC), PCI 133 MHz bus, 1 GB network speed
- Procurve 9308m network switch with EP J4895a 100/1000T modules and J4885a EP Mini-GBIC

Software Configuration

This section lists the software configuration used for this proof of concept.

Operating System - OpenVMS 7.3-2:

- HP Volume Shadowing for OpenVMS
- Latest system patches; HBMM patches are required
- TCPIP 5.4, ECO5

Database

Oracle9*i* RAC Version 9.2.0.5, running as active-active instances.

Load Generator

Swingbench 2.1f (for more information on Swingbench, see Appendix B).

Appendix B

This appendix describes the various vendors involved in this project as well as the software used for testing.

Digital Networks

Digital Networks has a rich history in providing network infrastructure for OpenVMS clusters in Disaster Tolerant environments. LightSand Communications Inc. has been an industry leader in SAN Over Distance products and technology. The recent technology partnership between Digital Networks and LightSand has generated the capability to support SAN extensions that will concurrently support OpenVMS Cluster infrastructure with HP Volume Shadowing for OpenVMS over the same physical wide area interconnect.

LightSand

LightSand Communications, Inc. is a company pioneering SAN connectivity and routing solutions. The LightSand S-8100B gateway used for testing provides cluster connectivity, IP connectivity, and Fibre Channel (FC) connectivity. In addition to FC SAN connectivity, the LightSand 8100 family switches support network connectivity for both Layer 2 Ethernet, including non-IP Cluster SCS traffic, and Layer 3 IP traffic over the same physical wide area interconnect.

Oracle RAC

Oracle9*i* Real Application Clusters (RAC) is an option to the Oracle9*i* Database Enterprise Edition, Release 2. It is a cluster database with a shared cache architecture that overcomes the limitations of traditional shared nothing and shared disk approaches to provide highly scalable and available database solutions. Oracle9*i* RAC allows large transactions to be separated into smaller ones for fast parallel execution, providing high throughput for large workloads. Through the introduction of a quorum disk, network failure and node failure are detected and resolved faster, resulting in faster completion of cluster reconfiguration. Lock remastering due to instance failure and instance recovery are concurrent. Failover capability is consolidated and enhanced to provide more robust and generic solutions. Oracle9*i* RAC can now function as a failover cluster with active instances on all nodes. It does require supporting clusterware software from the operating system that manages the cluster.

[8] This component was used for the test environment and would not be needed in a production environment.

Swingbench

Swingbench 2.1f was used as a load generator with typical order-entry functions[9] of 300 remote clients and 600,000 transactions.

Swingbench is an extensible database benchmarking harness designed to stress test Oracle databases via Java Database Connectivity (JDBC). It consists of a load generator, a coordinator, and a cluster overview. The software enables a load to be generated and the transaction response times are recorded. It was written internally by Oracle developers, primarily to demonstrate Real Application Clusters, but can also be used to demonstrate functionality such as online table rebuilds, standby databases, and online backup and recovery. It is not an official Oracle product, but is available for general use at no cost to the end user.

The code that ships with Swingbench includes two benchmarks: OrderEntry and CallingCircle. The testload used for this project is the OrderEntry. It is based on the oe schema that ships with Oracle9*i* Database and Oracle Database 10*g*. It has been modified so that the Spatial, Intermedia, and Oracle9*i* schemas do not need to be installed. It can be run continuously, that is, until you run out of space. It introduces heavy contention on a small number of tables and is designed to stress interconnects and memory. Both benchmarks are heavily CPU-intensive. The entire framework is developed in Java and as a result can be run on a wide variety of platforms. It also provides a simple API to allow developers to build their own benchmarks.

Glossary

600K	600,000 'typical' Order Entry transactions generated by Swingbench, the Oracle informal load-generating software (http://www.dominicgiles.com/swingbench.php). (This is the software that will be used to generate the I/O traffic that will be monitored in this project.)
Active	Clients are connected to that node.
Active-Active	A description of a type of Oracle RAC operational configuration composed of two nearly identical infrastructures logically sitting side by side. In this type, one node acts as a primary to a database instance and another one acts as a secondary node for failover purposes. At the same time, the secondary node acts as the primary for another instance and the primary node acts as the backup and secondary node. Clients typically connect in a distributed or round-robin fashion to both nodes.
Active-Passive	A description of a type of Oracle RAC operational configuration composed of two nearly identical infrastructures logically sitting side by side. One node hosts the dataset service or application, to which all clients connect, while the other rests idly waiting in case the primary system goes down. Upon failure, the primary server gracefully turns over control of the database and application to the other server or node who in turn becomes the primary server.
Copy State	Duplicate data on a source disk to a target disk. At the end of a copy operation, both disks contain identical information. Copy can be a full copy or a minicopy. Only Full Members and Merge Members can service user read requests.

[9] Typical order-entry functions are: new customers, product order, process order, browse of order, and browse for products.

Demand Merge	Merge process initiated by the DCL command `SET SHADOW/DEMAND_MERGE`.
Full SSM	SSM that is a full shadow set member, responding to user read and write I/O.
Forced Merge	Merge process initiated by system failure or crash.
HBMM	Host-Based Minimerge
HBVS	Host-Based Volume Shadowing
Local FCPY SSM	SSM that is a copy shadow set member, responding to user write I/O. SSM requires a full copy operation to make it a full member.
Local Full SSM	SSM that is a full shadow set member, responding to user read and write I/O. SSM is at the same site that the Active Oracle Instance is running at.
Local Full Merge SSM	SSM that is one of several merge shadow set members, responding to user read I/O using merge semantics and requiring full merge operation to transition VU to a steady state.
Local MCPY SSM	SSM that is a copy shadow set member, responding to user write I/O. SSM requires a minicopy operation to make it a full member, per master bitmap. Logging of all write I/O to shadow sets with bitmaps use messages that generate SCS traffic.
Local Minimerge SSM	SSM that is one of several merge shadow set members responding to user read I/O using merge semantics, which requires a Minimerge operation to transition VU to a steady state.
Merge State	Compare data on shadow set members and to ensure that inconsistencies are resolved. Merge can be a Full Merge or a Minimerge.
NIST	OpenSource software available from the National Institute of Standards and Technology that introduces a delay in transmitting data across a network in order to simulate long distances between hardware.
OpenVMS Server	Cluster member that serves directly connected devices to other cluster members.

Oracle RAC failover	The ability to resume work on an alternate instance upon instance failure.
Oracle TAF	Run-time failover that enables client applications to automatically reconnect to the database if the connection fails
Passive	No clients are connected to that node.
Remote FCPY SSM	SSM that is a copy shadow set member responding to user write I/O. SSM requires a full copy operation to make it a full member. Remote SSM is at a site that is not physically local to the Local node. In our testing, the remote site was simulated (via network delay) at distances of 600 km and 1000 km away from the local site.
Remote Full Merge SSM	SSM that is one of several merge shadow set members responding to user read I/O. Using merge semantics requires full merge operation to transition VU to a steady state. Remote SSM is at a site that is not physically local to the Local node. In our testing, the remote site was simulated (via network delay) at distances of 600 km and 1000 km away from the local site.
Remote Full SSM	SSM that is a full shadow set member responding to user read and write I/O. Remote SSM is at a site that is not physically local to the Local node. In our testing, the remote site was simulated (via network delay) at distances of 600 km and 1000 km away from the local site.
Remote MCPY SSM	SSM that is a copy shadow set members responding to user write I/O. SSM requires a minicopy operation to make it a full member. Remote SSM is at a site that is not physically local to the Local node. In our testing, the remote site was simulated (via network delay) at distances of 600 km and 1000 km away from the local site.
Remote Minimerge SSM	SSM that is one of several merge shadow sets responding to user read I/O. Using merge semantics requires a Minimerge operation to transition VU to a steady state. Remote SSM is at a site that is not physically local to the Local node. In our testing, the remote site was simulated (via network delay) at distances of 600 km and 1000 km away from the local site.
RPO	Recovery Point Objective. The maximum acceptable data loss in the event of a system outage.
RTO	Recovery Time Objective. The maximum acceptable time between a system outage and the continuation of operations.

Served FCPY SSM	SSM that is a copy shadow set member responding to user write I/O. SSM requires a full copy operation to make it a full member. Served SSM is at a site that is not physically local to the Local node. In our testing, the served site was simulated (via network delay) at distances of 600 km and 1000 km away from the local site. SSM is served to this system by another OpenVMS system in this cluster via a WAN.
Served Full Merge SSM	SSM that is one of several merge shadow set members responding to user read I/O. Using merge semantics requires a full merge operation to transition VU to a steady state. Served SSM is at a site that is not physically local to the Local node. In our testing, the served site was simulated (via network delay) at distances of 600 km and 1000 km away from the local site.
Served Full SSM	SSM that is a full shadow set member responding to user read and write I/O. Served SSM is at a site that is not physically local to the Local node. In our testing, the served site was simulated (via network delay) at distances of 600 km and 1000 km away from the local site.
Served MCPY SSM	SSM that is a copy shadow set member responding to user write I/O. SSM requires a minicopy operation to make it a full member. Served SSM is at a site that is not physically local to the Local node. In our testing, the served site was simulated (via network delay) at distances of 600 km and 1000 km away from the local site.
Served Minimerge SSM	SSM that is one of several merge shadow set members responding to user read I/O. Using merge semantics requires a Minimerge operation to transition VU to a steady state. Served SSM is at a site that is not physically local to the Local node. In our testing, the served site was simulated (via network delay) at distances of 600 km and 1000 km away from the local site.
Site ID	A `sysgen` value that volume shadowing uses to determine the best device to perform read I/O operations, thereby improving applications performance. This nonzero value indicates to the shadowing driver the site location of the specified shadow set or virtual unit (`DSAnnnn`). A value of zero is not considered valid.
SSM	Shadow Set Member. A SSM can either be a Full Member (no copy or merge in progress) or a Copy Member (copy in progress) or a Merge Member (merge in progress).
Steady State	A VU which has no SSMs in either a Merge or a Copy state.
T4	Timeline data gathering tool that makes use of OpenVMS MONITOR and creates `.CSV` files, which contain the MONITOR data gathered via batch jobs. This data can then be displayed using TLVIZ or made use of by any program that can read a `.CSV` file. For more information, refer

	to: http://h71000.www7.hp.com/openvms/products/t4/index.html
TLVIZ	Timeline Visualization tool that displays T4 data in graphical format.
VU	Virtual Unit. Represents the physical devices that make up the mounted Shadow Set.

For more information

- For questions, contact:
  openvms-info@hp.com and put DT RAC POC on the subject line.

- For Digital Networks, refer to:
  www.digitalnetworks.net

- For LightSand, refer to:
  www.lightsand.com

- For Oracle RAC, refer to:
  www.oracle.com

- For Swingbench, refer to:
  http://www.dominicgiles.com/swingbench.php

OpenVMS Technical Journal V8

Information Lifecycle Management (ILM) Strategies and OpenVMS Storage Applications

Ted Saul - Offsite Support Consultant

Introduction

Information Lifecycle Management (ILM) planning is a new approach to data storage tracking and management. ILM encompasses the life of data from its creation and how and when it will be accessed, to its point of deletion. The total data life cycle should be viewed from two points -- its access value and its retention/restore value -- while answering the following two questions:

1. How fast is this data required to be available?

2. How long must the data be saved and restorable?

Data security must always be considered as well, not only when it's online, but also when it has been backed up and archived.

The following article will help you answer those two questions in order to properly manage the life of your data. Additionally, the article looks at two HP OpenVMS products currently available and covers how they can help manage data. This includes data that is located on disk for immediate access and data in need of regular backup and archiving cycles. These two products will be discussed as examples of how applications may make ILM achievable and more efficient.

Terms Related to ILM

The following terms related to data location are essential for the discussion of ILM:

- Online Storage – Data is kept on disk for immediate access by users. A directory command will show the files and all the information about the files.

- Near-line Storage – Data is not kept immediately online but considered shelved and located in a format either on another disk or on tape within an automatic loader device. To the user, the data will show on a directory command, but upon access there is a slight delay as the data is automatically "unshelved" from the near-line device. This is an optional step.

- Off-line Storage – This is the traditional backup scenario where data is backed up to tape. The tape may then have been removed from the library and stored in a safe location. Information about the files backed up on the tape may be kept by manual methods or by using an application cataloging facility. Users will not see the files in their directory and a strategy needs to be in place for retrieval of this data.

How Fast is Data Required to be Available?

The term access value may be defined as the weight data carries and corresponds to how quickly it needs to be accessed by the end user. Data with a high access value will require an immediate response. Current patient records, for example, have a high access value to doctors, nurses, and other medical technicians requiring immediate information during the diagnosis and treatment of the patient. Storing this data off-line on tape for retrieval would be unacceptable. The same would be true for data in any real-time transaction processing environments.

Retrieval times for data with a high access value is measured in seconds and tenths of seconds. Information with a lower access value might include statistical data such as weather data to help determine the record high and low temperature for a particular day. This data probably won't be considered critical and its retrieval could be scheduled and planned. Another good example of data that may fall into this category is historical phone charges by a cell phone provider. This data might be classified as medium-level access and kept near-line or off-line. Immediate access that is measured in seconds will not be as critical, allowing for an off-line data storage strategy. Contrast this with data from the current cycle required for billing, reporting, and even criminal investigation purposes.

It may also be helpful to sub-categorize high access values into some strategy such as the following.

- High demand access where the data is required to be available almost immediately. Any delay in response will be noticeable and may adversely affect this business it is supporting.
- Medium demand where a slight delay in retrieval may or may not be noticeable but still tolerable.
- Low access where a request for retrieval may be made to another organization and lag time may be measured in a day or so.

These additional access classes help determine which type of software to use to manage data at certain points in its life. Data with the highest access value always needs to be kept online and available. The use of large disk arrays along with database applications can help to manage this data and ensure that it is always available when needed. Online data protection such as RAID strategies and volume shadowing are available to protect the vulnerability of this data between backups.

How Long must Data be Saved and Restorable?

How long data is to be retained or its retention requirement is the second important point to consider in its lifecycle. The period of time that may be required to retrieve and restore will also fall into this category. It is always important to keep backups of your data, but a prioritization of backups should be put into place so it is easy to know what type of recovery is available. Some data may also reach a point when the retention/restore value is not as high. This typically happens at the end of the data's lifecycle.

Retention and restore values consider how long data must be available in one form or another for auditing purposes and how easily the data can be restored. As with access value, retention might be sub-categorized as:

- High -- where restores may be required within a few minutes. Backups may be sent to disk rather than tape or to libraries where the tapes can be quickly mounted to restore data.
- Medium -- where restores may take a day to complete. Typically this media is left onsite and stored in some type of secure area.
- Low -- where restores may take a day or two particularly if the data is located at an offsite facility.

Medical data is a prime example of data that has a long retention period. In most cases, patient health information by law must be retained for seven years or -- in the case of minors -- until they reach the age of twenty-one. Medical data may not require immediate access but must be restored upon request.

A system of date-based backups may be setup and retained with daily, weekly and monthly backups. A weekly backup may replace a set of daily backups and a monthly backup may replace a set of weekly backups. Yearly backups may also be captured for long-term storage. Duplicate copies of long-term backups may be kept as well for disaster recovery purposes.

When thinking long-term, the media used to store data must be taken into account as well. Consider asking questions such as:

- Is the life of the media (tape, etc) in use expected to last for the required life of the data?
- Will the hardware be available to handle your current media in the long-term future?
- Should a plan for migration of data be put into place should advanced technology become available?

During an actual disaster is not the ideal time to test your backup strategy. The process for retrieving media, installing software, and recovering data both onsite and at DR sites should be regularly tested to ensure the quickest recovery time possible. It will be important to be able to identify where each critical piece of data exists in its lifecycle to ensure that no transactions are lost or overlooked.

HP Storage Products

OpenVMS offers two storage-related products to help with ILM:

- Hierarchical Storage Management System (HSM)
- Archive Backup System (ABS) applications

HSM provides manageability of data that has a high access value while ABS assists with managing the restore and retention values of data. ABS as a backup application also provides protection of all data no matter where it resides in its lifecycle. Both products serve their own purpose but can work together to provide a complete ILM solution.

Data that is needed for immediate access needs to be backed up on a regular basis. A proven backup strategy should be in place during this time to ensure that any failure -- whether minor or catastrophic -- can be recovered from easily. Hardware functionality such as RAID or shadowing may be looked at to deepen the level of protection of this data. Once the access value of data begins to drop, HSM can be used to shelve data to near-line storage and free up disk space while allowing reasonable access times. This strategy is most powerful when large data files are required to be accessed online but only randomly. From the user point of view, the data is still on the disk and visible via a directory command. In reality though, only the file header is left behind with the bulk of the data located either on another storage disk or on an easily accessible tape within a library. HSM catalogs then keep track of the location of the data for easy retrieval. Database applications may also be included in the scheme with the use of export functions to move data from the main application to a backup location. In turn, this file is shelved to near-line storage for access when the need arises to import the data back to the application.

Once the access value is no longer an issue, archiving of the data via ABS can take place. At this point, you need to prioritize and categorize how long data must be retained overall. In order to save space and reduce backup windows, low access data may be archived from disk. Archiving typically refers to the backing up of the data and a subsequent deletion from disk. These tapes may be stored in a library but, for safety's sake, should be taken out of a library and shipped to a secure offsite location.

HSM Settings

In HSM, policies can be set up to manage how long data is kept and maintained before being shelved to a near-line device. These policies can do the following:

- Specify which files to move between primary storage and shelf storage.
- Specify which files are not to be moved from primary storage.
- Set a "high-water mark" on primary storage to automatically trigger shelving on dormant data to shelf storage. A high-water mark is a defined percentage of disk space used that, when exceeded, causes shelving to begin.
- Set a "low-water mark" as a space-recovered goal to limit the number of files that are moved to shelf storage. A low-water mark is a defined percentage of disk space used that, when reached, causes policy-defined shelving to stop.

You will want to set your policies so that data with a medium access value becomes a candidate for shelving. However, data with high access values or that for any other reason should not be moved, should be set with the "no-shelve" bit set. Databases are prime candidates for this type of setting. Current data will be located in the database and archived out of the database via an export-type command. This exported data may then become eligible to be shelved.

ABS Settings:

ABS has two policies that directly affect ILM: the SAVE and the ARCHIVE.

The SAVE policy identifies what files or disks are to be backed up, the schedule to do so, and specific information about the backup. This is typically unique data that only occurs once within the configuration. The ARCHIVE policy defines information about the backup usually found to be redundant with other backups. The ARCHIVE policy is then associated with one or more SAVE policies. For example, the catalog where data about each backup is to be stored is written to a field on the ARCHIVE record. There may be multiple SAVE policies storing their data to the same catalog. In this case, the one ARCHIVE policy will be associated with all these SAVE policies.

Between the ARCHIVE and SAVE there are seven policy settings that may affect ILM.

1. Retention – Found on the ARCHIVE, retention is the length of time in days to keep data in catalogs and available for lookup. For example, retention might be set to 7, 30, or 365 days. The longer the data is kept, the larger the catalog will grow. It is important to create the

correct system of catalogs to ensure efficient disk usage. For example, yearly backups should be kept in a catalog of their own. This data will need to be retained for a long time period and there is no use in having daily or monthly backups work around this data. The retention setting is the key in the ILM environment to ensure that metadata about backups is available as long as needed.

2. Scratch Date – The scratch date is the length of time to keep data on tape. Within ABS/MDMS, a separate volume database tracks information about tapes. The scratch date is stored on the volume record and is initially derived from the retention value. It is possible to manually change the scratch date found on the volume causing a tape to be retained longer than the information kept by retention in the catalog. This may be useful for low-priority-retention-valued volumes where some system of manually tracking these tapes has been put into place. This can help reduce the size of the catalogs. ILM will use the scratch date to preserve the data on tape for its appropriate lifetime. Once the scratch date is reached, the volume may be set to a free state and the data at risk to be overwritten.

3. Offsite Date – Recorded on the volume record as well, this field defines when to take the tape to an offsite location. Vaulting is a process that needs to be set up at each site to ensure that backup data is safe from physical harm. Moving backup volumes also reduces the threat of a single point of failure by having backups at multiple locations. The offsite date is the key to its implementation.

4. Onsite Date – Also stored on the volume record, this field defines when to bring the tape back onsite. Vault management settings, onsite, and offsite dates, play an important role in ILM and are set after a backup is completed. This may be done via an epilogue command or by using some manual process.

5. Consolidation – Stored on the ARCHIVE policy, ABS/MDMS uses this field to determine how long to make or keep a volume set. A volume set is one or more tapes tied together using the previous and next pointers. There are three different modifiers available for consolidation: interval, savesets, and volumes. The most commonly used is the interval qualifier that tells the number of days that volume set should be active. Once this date is reached, ABS/MDMS will retire the volume set and start a new one. ILM needs this setting to ensure that data is movable offsite in a timely manner and does not tie up large amounts of data in a volume set.

6. Expiration date – Expiration is also on the ARCHIVE and is the date the saved data expires. Expiration can be used as an alternative to retention.

7. Catalog – When implementing ABS/MDMS it is advisable to develop a system of catalogs to ensure time-efficient cleanups, ability to backup and restore, and proper amount of disk storage available to store the catalogs. Depending on the amount of data backed up and the length of the retention, catalogs can become very large and need to be managed carefully. Catalog is found on the ARCHIVE and may be used for multiple SAVEs. Catalog become key in initiating faster restores by allowing for the easy location of data as well as volumes.

ILM Data Zoning

To get started on setting up your site-specific Information Lifecycle Management strategy, it is a good practice to spend time reviewing the data on your systems, their security requirements, and current locations. With large arrays of disk with gigabytes and terabytes of data, it is even more important to get a handle on what files are on your system.

1. Define a list of policies and rules that affect the data on your system such as:

- Governmental Policy's (e.g., HIPAA, Sarbanes-Oxley)
- Corporate Mandates
- Application requirements
- Commonsense rules

With this information, a corporate backup policy can be initiated that includes how long data should be retained and the process for handling the data. Security issues should be addressed in the policy as well as requirements for scrubbing a tape after use.

2. Divide data farms into zones:

- System Files - System files may change with one of the following conditions.

  ECO installed which could affect drivers or other images on the system data files such as:

  – vms$audit_server.dat
  – vmsimages.dat
  – sysuaf.dat
  – rightslist.dat
  – qman$master.dat

  Configurations to the system such as new users, security changes, adding of queues, and DECNET and TCP/IP changes can affect these files.

- Application Files – Upgrades, updates, and ECOs for specific application may affect the images belonging to applications. These images may also affect who has access to what data.
- Data Files – The data for an application that may be contained in text, binary, database, and other type of files. Define whether they are high, medium, or low access.
- User Files – Those files kept by individual users for managing their day-to-day work. These files will vary from system to system but they should not fall into one of the above categories.

3. Apply the policies to each data zone:

Write into the backup policy and apply findings from step one to those in step two. For example, if dealing with patient information within the medical community, a retention value of seven years may need to be applied. Accounting data will have Sarbanes-Oxley requirements applied should any type of governmental audit take place.

The policy should also include a "bare metal restore" section in case of catastrophic outages. This will address how often application and system files need to be backed up. Perhaps ECOs or application updates will only be permitted after a system backup. Backups of original distribution binaries should also be covered in the policy to ensure their quick location.

4. Test recovery procedures:.

As mentioned earlier, an actual disaster recovery is not the time to test your procedures. Schedule specific times to restore tapes and rebuild systems to ensure they are valid and accessible.

5. Review on a regular basis:

Ensure that processes are kept up to date. When architectural changes take place review your backup policy to make sure that everything is covered. If using HSM and/or ABS, you may need to review your policies for them as well. Make sure all your data is being backed up appropriately and at the expected time. A formal change process that updates the backup policy during any software or hardware modification is also a good idea.

Information Life Cycle Stages

The following chart suggests the stages of life your data may travel through. Time intervals and depth to which backups must be retained may vary from site to site.

State	**Retention Requirement**	**Access Value**	**Description**
Data creation	High	High	Data is being created and may be subject to many changes. Getting a good backup may be difficult and the use of journaling may be useful.
Current	Regular backup	High	Data is accessed and read consistently. Modifications may be made during this point of time.
Not current – high probability access	Regular backup	High	The age of the data is getting older but it is being accessed regularly.
Not current – low probability access	Regular backup	Medium	Data is now being used less often but archiving to tape would not be efficient. A near-line solution may fit well at this point.
Stored	Send to tape – archive	Low	Data has been moved to tape and deleted off disk. Access requirements are low but policies will determine the retention.
Transit	Tape going offsite	Low	Data is moving to a secure location and unavailable during this timeframe.
Offsite	Tape secure	Low	Data is now located at a secure offsite location.
DR – transit	Tape coming onsite	High	An outage has occurred requiring disaster recovery procedures to be initiated. Data is now on its way back to the datacenter.
Transit – normal retire	Low	Low	In this case, rather than the DR Transit, the data is on its way back to the datacenter for its normal retirement. This date has been set by backup policy.
Volatile – tape in transition	Low	Low	Backup application may put the data in this state as a last chance to recover effort. Application would need to be updated should recovery be done at this point.
Overwritten	Tape reused	Data no longer	Tape is reinitialized and possibly reused at this point.

		available	

Summary

In the early years of computing, data was used to accomplish an organization's work on the computer and, in theory, make operations more efficient. In today's world, though, security and privacy issues along with the many levels of governmental control have made data a liability to the organization. It is imperative that it be controlled and protected. Managing the information lifecycle can be a time-consuming and tedious process but must remain a priority. Any file created and stored on a computer whether online, near-line, or off-line should be in its location by design with its movement carefully monitored.

For more information

Please contact the author with any questions or comments regarding this article. To get to the latest issue of the OpenVMS Technical Journal, go to:
http://www.hp.com/go/openvms/journal.

OpenVMS Technical Journal V8

System Service Interception

Ruth Goldenberg

Overview

Many of the actions that the OpenVMS operating system takes on behalf of a user are implemented as procedures called system services. A user application requests system services directly, and components such as the file system also request system services on an application's behalf. For example, a program calls the Get Job/Process Information ($GETJPI) system service to find out specific details about another process or job and the Queue I/O Request system service to perform input from or output to a specific device.

System service interception (SSI) is a mechanism that enables system services to be intercepted and user-specified code to run before, after, or instead of the intercepted service. An image can declare routines to perform pre-service processing, post-service processing, system service replacement routines, or any combination thereof.

The implementation of the mechanism limits interception to system services in executive images; that is, system services in privileged shareable images cannot be intercepted.

SSI is used by the Debugger and OpenVMS tools such as the Heap Analyzer and Performance Coverage Analyzer (PCA). The Debugger, for example, uses it to implement global section watchpoints and speed up static watchpoints. The Heap Analyzer needs to follow changes to memory layout. It intercepts memory-changing services ($EXPREG, $CRMPSC, and so on) to track and display these changes as they occur. At one customer's site, SSI has been used to intercept file opens and closes done by a third-party application to minimize unnecessary operations: $CLOSE is intercepted to keep a file open and $OPEN to refrain from re-opening a still-open file.

SSI has been available, but not previously documented, on OpenVMS Alpha since Version 6.1. It is available for OpenVMS I64 with Version 8.3 and later versions.

This article first describes how OpenVMS dispatches to executive system services on both Alpha and I64 platforms and then how these services are intercepted.

System Service Dispatching

Most system service procedures are contained in executive images and reside in system space; others are contained in privileged shareable images.

System services typically execute in kernel or executive access mode so that they can read and write data structures protected from access by outer modes.

The implementation of inner mode system services is based on a controlled change of access mode. Although an unprivileged process can enter an inner access mode to execute code in that mode, it can execute only procedures that are part of the executive or that have been specifically installed by the system manager.

When a program requests an inner mode system service, it executes a system service transfer routine that serves as a bridge between the requestor's mode and the inner mode in which the service procedure executes. System service transfer routine names are resolved using the same mechanisms as externally visible names in a shareable image.

The following sections describe how executive system service names are resolved and how control is transferred to an executive system service procedure on both OpenVMS Alpha and I64 systems.

System Service Dispatching on OpenVMS Alpha

Every OpenVMS Alpha procedure is described by a data structure called a procedure descriptor (PD), which contains the address of the procedure's code entry point and information about its type and characteristics. This is true for every procedure, whether it is a procedure in an executable, shareable, or executive image, or whether it is an ordinary procedure, a system service transfer routine, or a system service procedure.

A compiler generates a PD for each procedure in a module and places them together in a program section called a linkage section.

A linkage section also contains information about calls to external procedures. A call to an external procedure is represented as a two-quadword data structure called a linkage pair. By the time a call using a linkage pair is executed, the first quadword of the linkage pair must contain the external procedure's code entry address, and the second quadword must contain the address of its PD.

The global symbols from object modules in a shareable image that are to be visible externally are called universal symbols. Each universal symbol in an image is represented within two image structures: the global symbol table and the symbol vector. A global symbol table entry gives the offset of the corresponding symbol vector entry. A symbol vector entry consists of two quadwords. For a universal symbol that is the name of a procedure, the two quadwords hold addresses of the procedure's code entry point and its PD. That is, they form replacement contents for a linkage pair.

When the linker links an object module containing a call to an external procedure in a shareable image, the linker cannot resolve the contents of the linkage pair because a shareable image is typically not assigned address space until it is activated. Instead, the linker records in the image it is building the need for the image activator to resolve the contents.

When an image and a shareable image with which it is linked are activated, each procedure symbol vector entry in the shareable image is updated to contain the actual addresses of the PD and code entry point for that procedure. The image activator resolves the linkage pair for the call into the shareable image by replacing the linkage pair's contents with the corresponding contents from the shareable image symbol vector entry.

To call the procedure in the shareable image, compiler-generated code in the main image loads from the linkage pair the target procedure's PD and code entry addresses. It transfers control to the code entry address, saving the return address in a register.

In general, transferring control to an executive system service resembles transferring to a procedure in another image. It differs in the following ways:

- There is one symbol vector, built as part of the SYS$PUBLIC_VECTORS.EXE image, for all system services in all executive images. The image also contains a global symbol table listing all the system service transfer routine names and their offsets in the image's symbol vector.
- The PDs for all executive services initially describe system service transfer routines within SYS$PUBLIC_VECTORS.EXE. Its symbol vector initially contains linkage pairs with the addresses of these PDs and transfer routines.

When an executive image containing a system service is loaded during system initialization, information about the service in SYS$PUBLIC_VECTORS.EXE is modified.

- If the service is a mode of caller service, the symbol vector entry contents are modified to reflect the location of the service PD and entry point in the executive image. In other words, the system service call goes directly from the caller's image to the service procedure.
- If the service is an inner mode service, the transfer routine is modified to identify the service and specify its mode.

Figure 1 shows a user image that calls both an inner mode (SYS$k) and a mode of caller (SYS$moc) system service. When an image that requests a system service is linked, the linker resolves the system service transfer routine name by searching the SYS$PUBLIC_VECTORS.EXE global symbol table. It stores the symbol vector offset corresponding to the system service transfer routine in the linkage section of the image making the service request. It stores information about the need to update that linkage pair in the fixup section of the image.

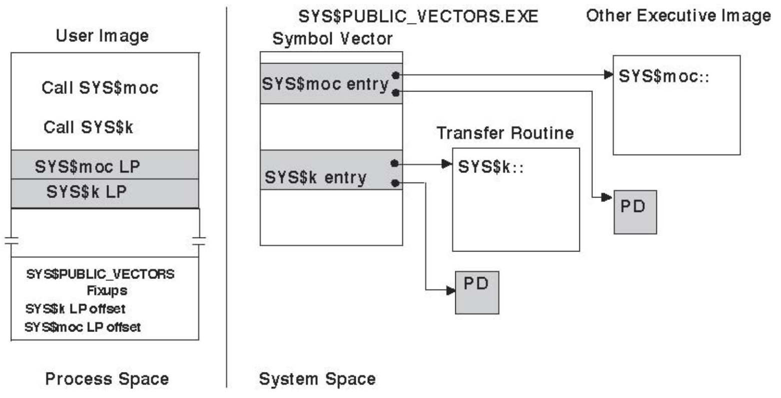


Figure 1 - Resolving OpenVMS Alpha System Services Names

In the figure, the linkage section has a shaded linkage pair for each of the two services. Its fixup section lists the two corresponding fixups.

Unlike other shareable images, SYS$PUBLIC_VECTORS.EXE is not activated with an executable image. Instead, as part of the executive, it is loaded into system space during system initialization. Because all processes map system space, SYS$PUBLIC_VECTORS.EXE is shared.

In the SYS$PUBLIC_VECTORS.EXE symbol vector in Figure 1, the two entries for these services are shaded. The entry for SYS$moc points to a PD and an entry point within the executive image that contains that service. The entry for SYS$k points to the PD and a transfer routine within SYS$PUBLIC_VECTORS.

When an executive image with an inner mode service is loaded, the corresponding transfer routine is modified to contain an instruction that loads a service-specific value into R0 and either a CALL_PAL CHMK or a CALL_PAL CHME instruction.

Figure 2 shows a flow to and from the example kernel mode service SYS$k.

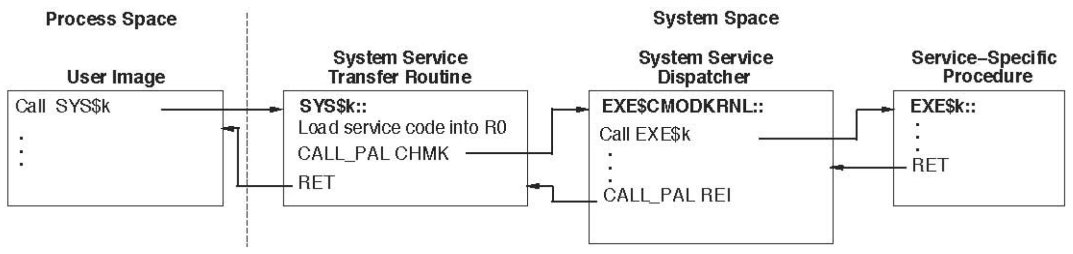


Figure 2 - Transferring to Inner Mode OpenVMS Alpha System Services

The user image calls the system service, passing control to the transfer routine in SYS$PUBLIC_VECTORS.EXE.

Its CALL_PAL CHMK instruction causes an exception and a mode change to kernel. The change mode exception is handled by the system service dispatcher, which uses the value in R0 to determine which system service procedure to call.

System Service Dispatching on OpenVMS I64

An OpenVMS I64 procedure can be described by a data structure called a function descriptor (FD), which contains the address of the procedure's entry point and the address that should be loaded into its global pointer (gp) register. The gp is a base address for references to the short data segment, which includes small writable data, pointers to external data, and procedure linkages.

An FD roughly corresponds to an Alpha linkage pair and is accessed in transferring control from one procedure to another. Most FDs are created by the linker. Each externally visible procedure has one so-called “official” FD, but other procedures do not necessarily have any.

To describe universal symbols in a shareable image, the OpenVMS I64 linker creates a symbol vector and the equivalent of a global symbol table. The global symbol table lists the shareable image's universal symbols, their symbol vector indices, and their types. A symbol vector entry contains one quadword for each universal symbol; in the case of a universal procedure it contains an FD address.

A linkage to a procedure in another image or another image segment in the same image is represented in the calling procedure's short data by a local FD or a pointer to an official FD. By the time a call using the FD is executed, it must contain the target procedure's entry point and the value for its gp.

If the caller and target procedure are in different images, the linker generates an FD, a procedure linkage table (PLT) routine in the same program section as the call site, and a br.call to the PLT routine.The addresses in the local FD are fixup targets. The PLT routine uses the FD to load the gp register and branches to the target procedure's entry point.

In general, transferring control to an executive system service resembles transferring to a routine in another image. It differs in that SYS$PUBLIC_VECTORS.EXE contains the symbol vector for system services in all executive images and the FDs for all such services. Its FDs point either to executive images or to service-specific routines in a system space promote area: FDs for mode of caller services point to service-specific routines in executive images; FDs for inner mode services point to service-specific routines in a part of system space called the promote area.

Figure 3 shows an I64 image that calls both an inner mode (SYS$k) and a mode of caller (SYS$moc) system service. Note that the figure is somewhat simplified; Section *System Service Interception on OpenVMS I64* contains a more complete description.

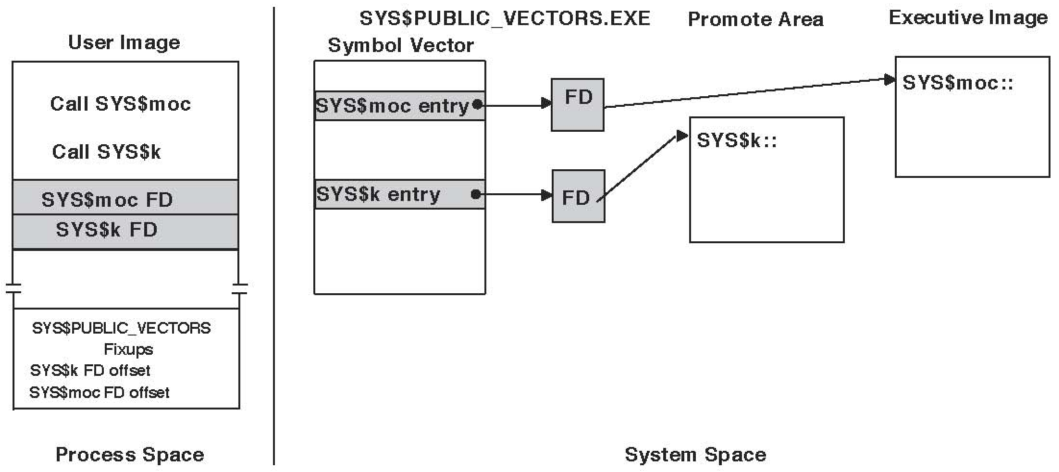


Figure 3 - Resolving OpenVMS I64 System Services Names Against the Original SYS$PUBLIC_VECTORS.EXE

On I64 platforms, the mechanism for changing to an inner access mode is to execute an epc instruction on a page with a special protection code. Such a page is called a promote page. A set of virtually adjacent promote pages is called a promote area. Routines in the promote area change access mode and transfer control to the system service dispatcher.

When an executive image is loaded that contains an inner mode service, a transfer routine specific to that service is created in the promote area. The SYS$PUBLIC_VECTORS.EXE FD for the inner mode service is modified to contain the address of the promote area transfer routine.

Figure 4 shows a simplified flow to and from the example kernel mode service, SYS$k.

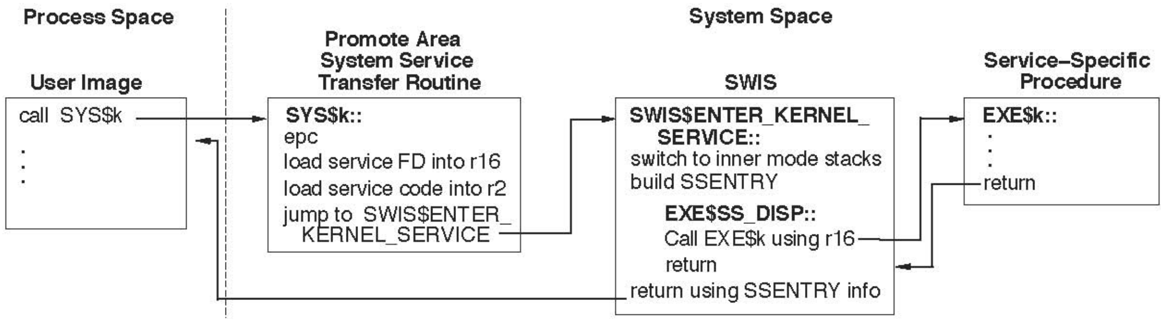

Process Space
System Space
User Image
call SYS$k
Promote Area System Service Transfer Routine
SYS$k::
epc
load service FD into r16
load service code into r2
jump to SWIS$ENTER_KERNEL_SERVICE
SWIS
SWIS$ENTER_KERNEL_SERVICE::
switch to inner mode stacks
build SSENTRY
EXE$SS_DISP::
Call EXE$k using r16
return
return using SSENTRY info
Service–Specific Procedure
EXE$k::
return

Figure 4 - Transferring to Inner Mode OpenVMS I64 System Services

The user image calls the system service, passing control to the transfer routine in the promote area.

The promote area routine executes an epc instruction to change mode to kernel, loads the service's FD address into a register, loads a code identifying the service into another register, and branches to an executive component called software interrupt support (SWIS).

SWIS switches to the inner mode stacks (memory and register). It builds a stack data structure called an SSENTRY that records the state of the thread of execution at the time of the mode switch, including the address to which control should be returned. It then transfers to the system service dispatcher, which uses the service FD to load the gp for the system service procedure and to call its entry point.

When the service is done, the system service dispatcher returns to SWIS. SWIS restores state from the SSENTRY, switches back to the outer mode stacks, and returns to the system service caller.

System Service Interception (SSI)

SSI enables system services to be intercepted and user-specified code to run before, after, or instead of an intercepted service. When a system service request is intercepted, SSI code saves the complete system service argument list in a VAX-style argument list format and calls any pre-processing routine. It then calls the real service or the replacement routine using the saved argument list. When the service replacement routine returns, SSI code calls any post-processing routine with status from the service or replacement routine.

SSI is local to the process that has enabled it.

Although the same SSI application programming interfaces (APIs) are provided on both Alpha and I64 platforms, the underlying SSI implementation is quite different. SSI requires alterations in either the system service name resolution or the dispatch to system services. On OpenVMS Alpha, the SSI implementation depends on altering service name resolution. On OpenVMS I64, the implementation depends on altering transfer routines. The following sections describe these two different implementations.

System Service Interception on OpenVMS Alpha

On OpenVMS Alpha, the SSI mechanism is implemented in a privileged shareable image called SYS$SSISHR.EXE. Its symbol vector is laid out in the same order as the symbol vector for SYS$PUBLIC_VECTORS.EXE. SYS$SSISHR.EXE essentially provides a set of jacket routines for all the SYS$PUBLIC_VECTOR services as well as a set of APIs to declare pre- and post-processing routines and replacement routines.

If the main image was linked with SYS$SSISHR.EXE or linked /DEBUG, the image activator activates SYS$SSISHR.EXE before doing any fixups. Once it has been activated, the image activator fixes up references to SYS$PUBLIC_VECTORS.EXE symbols using the SYS$SSISHR.EXE symbol vector instead. This means that services can be intercepted only from images activated with and after SYS$SSISHR.EXE.

If the main image was not linked with SYS$SSISHR.EXE and was not linked /DEBUG, its references to system service names are fixed up against SYS$PUBLIC_VECTORS.EXE. A subsequent dynamic activation of DEBUG with the DCL CTRL/Y $DEBUG sequence or with connect/disconnect cannot cause SYS$SSISHR.EXE to become activated in time to intercept system service calls from images already activated because they have already been fixed up.

Figure 5 shows the image of Figure 1 with references to SYS$PUBLIC_VECTORS.EXE symbols fixed up against SYS$SSISHR.EXE: the shaded LPs get the contents of the related SYS$SSISHR.EXE symbol vector entries and thus cause transfer to SYS$SSISHR.EXE. All of the symbol vector entries transfer to the same place, a routine called SSI_TRANSFER. SSI_TRANSFER is, however, entered with the address of the PD unique to that system service and thus can determine which service was requested.

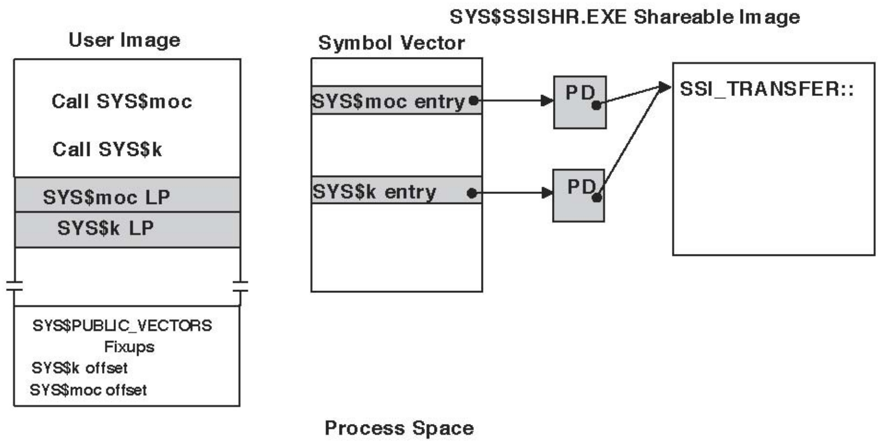


Figure 5 - Intercepting OpenVMS Alpha System Services

SSI_TRANSFER is written in assembly language to enable direct access to registers involved in the service call. It saves the complete system service argument list on the stack in VAX-style argument list format. It then calls the main SSI routine, SSI_MAIN_TRANSFER, with the following arguments:

- The address of the VAX-style argument list
- The service caller's return address
- The address of the service PD within SYS$SSISHR.EXE

SSI_MAIN_TRANSFER transforms the service PD address to the address of the service PD within SYS$PUBLIC_VECTORS. It determines whether any pre-processing, post-processing, or replacement routines have been declared and calls them before, after, and/or instead of the actual system service. It takes the following steps:

1. It determines whether the service was requested from an inner mode. If so, it calls the real system service with the saved arguments.
2. It determines whether the service caller was SYS$SSISHR.EXE itself. If so, it calls the real system service with the saved arguments and returns.

3. It copies the lists of pre-processing and post-processing routines in case any pre-processing or post-processing routine should declare or cancel a pre-processing or post-processing routine. Any such call will not take effect until the next system service request.
4. If any pre-processing routines have been declared, it calls them in the reverse order in which they were declared.
5. If a replacement routine was declared for this service, SSI_MAIN_TRANSFER calls it. Otherwise, it calls the actual system service.
6. When the replacement routine or actual system service returns, R0 contains the status value. SSI_MAIN_TRANSFER saves R0 to pass it as an argument to post-processing routines.
7. If any post-processing routines have been declared, it calls them in the order in which they were declared.
8. It restores R0 as the status value and returns to the caller.

System Service Interception on OpenVMS I64

On I64, the SSI mechanism is implemented in a shareable image called SYS$SSISHR.EXE and a privileged shareable image called SYS$SSISHRP.EXE. SYS$SSISHR.EXE provides the APIs to declare and cancel pre- and post-processing routines and replacement routines. SYS$SSISHRP.EXE contains inner mode support routines for SYS$SSISHR.EXE.

A system space copy of SYS$PUBLIC_VECTORS.EXE created during system initialization provides hooks for interception. References to system services from within executive images are fixed up against the original version of SYS$PUBLIC_VECTORS.EXE. By default, references to system services from all other images are fixed up against the copy of SYS$PUBLIC_VECTORS.EXE.

When a process is created, the physical pages that make up the system-space promote area are double-mapped into its P2 space. Every FD in the copy of SYS$PUBLIC_VECTORS.EXE, even one for a mode of caller service, points to a service-specific routine in the P2 space mapping of the promote area. Figure 6 shows an image fixed up typically, namely against the copy of SYS$PUBLIC_VECTORS.EXE.

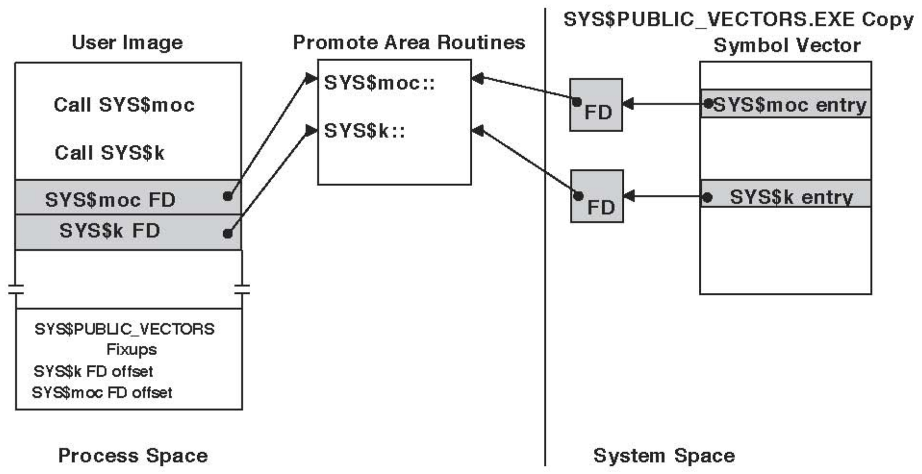


Figure 6 - Resolving OpenVMS I64 System Services Names Against the Copy of SYS$PUBLIC_VECTORS.EXE

When an image calls an SSI API to request system service interception, a process-private copy of the promote area is created, and promote area routines are modified to transfer control to SYS$SSISHR.EXE.

Figure 7 shows an image set up to intercept SYS$k on OpenVMS I64.

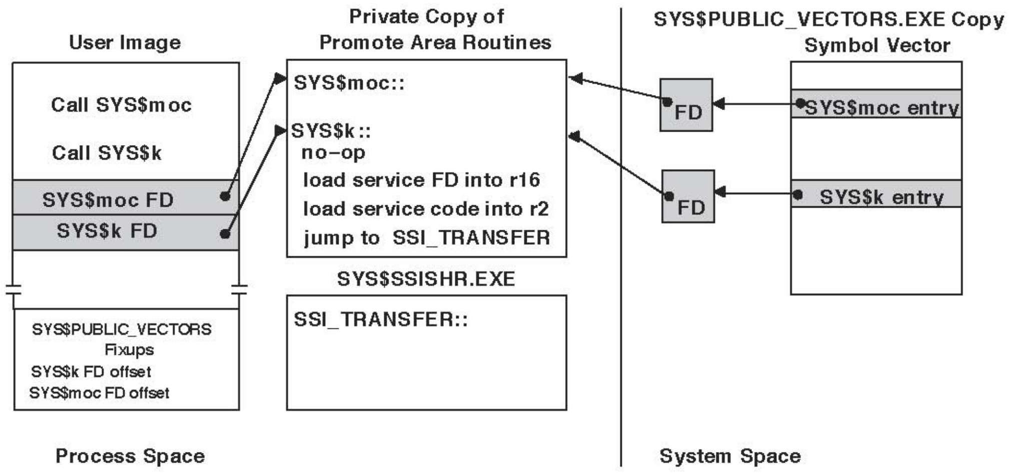


Figure 7 - Intercepting OpenVMS I64 System Services

The modified transfer routines for both mode of caller and inner mode services record the address of the system service FD and a code that identifies the service and transfer to SSI_TRANSFER, in SYS$SSISHR.EXE.

As on Alpha, the I64 version of SSI_TRANSFER is written in assembly language to enable direct access to registers involved in the service call. It saves the complete system service argument list on the stack in VAX-style argument list format. It then calls the main SSI routine, SSI_MAIN_TRANSFER, which is common code with the OpenVMS Alpha version.

As a result of this implementation, services in a process can be intercepted regardless of when SYS$SSISHR.EXE and SYS$SSISHRP.EXE are activated. Once system service interception has been enabled in a process, any SYS$PUBLIC_VECTORS service requests made from user mode can be intercepted. This includes service requests made from shareable images linked with the main image.

For more information

For a detailed description of image activation, see OpenVMS *Alpha Internals and Data Structures: Scheduling and Process Control*, available from Digital Press.

For a detailed description of system service dispatching on OpenVMS Alpha, see *OpenVMS AXP Internals and Data Structures Version 1.5,* Digital Press

For a detailed description of the OpenVMS Alpha and I64 calling standards (e.g., PDs, linkage pairs, and FDs), see the *HP OpenVMS Calling Standard*, available at http://h71000.www7.hp.com/doc/82final/5973/5973PRO.HTML

For information on the use of SSI, please check the HP OpenVMS Systems Documentation home page for forthcoming documentation.

For information on a mechanism to intercept privileged shareable image services, see *Faking it with OpenVMS Shareable Images* in Volume 7 of the OpenVMS Technical Journal http://h71000.www7.hp.com/openvms/journal/v7/faking_it_with_openvms_shareable_images.html.

OpenVMS Technical Journal V8

What's in your AlphaServer console?

Kostas G. Gavrielidis (GSE-MSE) Master Technologist HP Services

Overview

An HP Services engineer at a customer production environment must find out how a given AlphaServer has booted up and identify some of the console parameter settings.

This article explores how to get to the Alpha console and how to get the settings for the console environment variables from the operating system via implemented console callbacks.

In HP customer production environments, are we very rarely allowed to shutdown or reboot a production system, let alone try to connect a device to the console of the system. Our customers will allow only a secure connection to their production systems via the SSH[1] protocol. This will get us only to the operating system level. So how do we get the environmental variables and their settings from the boot loader[2] without taking the system down?

What is the SRM?

All HP Alpha systems running the OpenVMS, Tru64 UNIX, and the Linux operating systems come with the System Reference Manual (SRM) console subsystem in their firmware, perhaps along with other consoles such as the ARC/AlphaBIOS console. The SRM has control of the system hardware and provides the following features (please see the Alpha Architecture Reference Manual for more details):

- Initializes, tests, and prepares the hardware platform for system software

[1] `ssh` (secure shell) is a program for logging into a remote system and for executing commands on the remote system. It is intended to replace `rlogin` and `rsh`, and provide secure encrypted communications between two un-trusted hosts over an insecure network.

[2] A boot loader or boot manager is a program that loads the image of the operating system to be run on a particular hardware platform. Other boot loaders include the GNU's Grand Unified Boot Loader (GRUB), the Linux Loader (LiLo), the Windows NT loader (NTLDR), the aboot loader, the Alpha miniloader (MILO), Initial System Loader (ISL) for PA-RISC systems, Extensible Firmware Interface (EFI) for Itanium systems.

- Loads in memory and starts the execution of the system software
- Controls and monitors the state changes of all processors
- Provides services to system software that simplify system software control of, and access to platform hardware
- Provides a monitor and control of the system

The Privileged Architecture Library (PALcode) provides a mechanism to implement a range of functions, provided by the SRM, consistently without microcode. These functions range from the binary encoding of the instruction and data to the execution mechanisms and synchronization primitives.

Some of the more useful SRM commands (at the ">>>" prompt), include: `show device`, `show config | more`, `e pc, e sp, e ps, e r26, crash, halt`, `init`, `boot`, `memtest`, `memexer_mp`, `show fru`, `sys_exer`, `test`, and `wwidmgr`.

Here is the SRM command to show all devices.

```
>>> show device
Resetting I/O buses...
ewa0: link up : Negotiated  100BaseTX: full duplex
ewb0: link up : Negotiated  100BaseTX: full duplex
dka0.0.0.14.0              DKA0              COMPAQ BD036745A4  B010
dqb0.0.1.13.0              DQB0             Compaq   CRD-8402B  1.03
dva0.0.0.0.0               DVA0
ewa0.0.0.9.0               EWA0              00-10-64-30-D3-E0
ewb0.0.0.11.0              EWB0              00-10-64-30-D3-DF
pga0.0.0.16.0              PGA0        WWN 1000-0000-c92a-7195
pgb0.0.0.17.0              PGB0        WWN 1000-0000-c92a-71f5
pka0.7.0.14.0              PKA0                  SCSI Bus ID 7
>>>
```

This is the SRM command to show hardware configuration.

```
P00>>> show config | more
                        Compaq Computer Corporation
                      Compaq AlphaServer ES45 Model 2

Firmware
SRM Console:    V6.1-3
PALcode:        OpenVMS PALcode V1.93-37, Tru64 UNIX PALcode V1.88-28
Serial ROM:     V2.18-F
RMC ROM:        V1.0
RMC Flash ROM:  V1.9

Processors
CPU 0           Alpha EV68CB pass 2.4 1000 MHz  8MB Bcache
CPU 1           Alpha EV68CB pass 2.4 1000 MHz  8MB Bcache
CPU 2           Alpha EV68CB pass 2.4 1000 MHz  8MB Bcache
CPU 3           Alpha EV68CB pass 2.4 1000 MHz  8MB Bcache

Core Logic
Cchip           Rev 17
Dchip           Rev 17
PPchip 0        Rev 17
PPchip 1        Rev 17
TIG             Rev 2.6

--More-- (SPACE - next page, ENTER - next line, Q - quit)
```

Here is the command for examining the pc, sp, ps and r26 registers.

```
P00>>> e pc
PC psr:                0 (    PC) FFFFFFFF0048C1F0
P00>>> e sp
gpr:                  1E (   R30) FFFFFE06863BF3D0
P00>>> e ps
ipr:                  17 (    PS) 0000000000001F00
P00>>> e r26
gpr:                  1A (   R26) FFFFFFFF004A4660
```

What are the console environment variables?

An environment variable is a name and value association that is maintained by the console program. There are two types of environmental variables: volatile and nonvolatile. The volatile variables are initialized to their default by a system reset, and the nonvolatiles remain as set across system power cycles. Alpha systems have a variety of variables with values set up within the SRM system console. These environment variables control the particular behavior of the console program and the system hardware, the particular console interface presented to the operating system, various default values for the operating system bootstrap, and related control mechanisms. In other words, ''the environment variables provide an easily extensible mechanism for managing complex console state.''

Most users will never see or have a need to use the console system on the Alpha platforms. Technologists who provide proactive consulting and support to the customer production environments, however, must know the settings of certain important console variables such as: `auto_action`, `boot_dev`, `booted_dev`, `boot_file`, `booted_file`, `boot_osflags`, `booted_osflags`, `console`, `os_type`, `pal`, and `sys_serial_num`. The specific number and names of these variables vary from platform to platform and by firmware version.

How to get to the Alpha console

The console prompt varies from ">>>" to "`Pnn>>>`"

where nn is the `00` | `01` | `02` | … | nn CPU processor.

Several methods are available to get to the console:

- After an orderly shutting down of the system

  On Tru64 UNIX:

  ```
  # /usr/sbin/shutdown –h now "system is going down…"
  …
  # /usr/sbin/halt
  >>>
  ```

  On OpenVMS:

  ```
  $ @SYS$SYSTEM:SHUTDOWN.COM
  …
  >>>
  ```

- Pressing the HALT button on the system
- From a direct serial connection to the Alpha console, issue the <Ctrl>P command to get to the console prompt ">>>" (or "`P00>>>`"), as shown in the example below.

```
Welcome to OpenVMS (TM) Alpha Operating System, Version V7.3-2

Username:

<Ctrl>P

halted CPU 0
CPU 1 is not halted
CPU 2 is not halted
CPU 3 is not halted

halt code = 1
operator initiated halt
PC = ffffffff88670438
P00>>>
```

To control the behavior of the system hardware, SRM provides a variety of environment variables as a means of managing the system states. For a complete list of all the console environment variables and their values issue the command "`show *`" from the console prompt "`P00>>>`".

```
P00>>>show *
auto_action             RESTART
boot_dev                dgb2001.1003.0.10.0 dgb2001.1004.0.10.0
dga2001.1001.0.8.0 dga2001.1002.0.8.0
boot_file
boot_osflags            3,0
boot_reset              OFF
bootbios
bootdef_dev             dgb2001.1003.0.10.0 dgb2001.1004.0.10.0
dga2001.1001.0.8.0 dga2001.1002.0.8.0
booted_dev              dgb2001.1004.0.10.0
booted_file
booted_osflags          3,0
char_set                0
com1_baud               9600
com1_flow               SOFTWARE
com1_mode               THROUGH
com1_modem              OFF
com2_baud               9600
com2_flow               SOFTWARE
com2_modem              OFF
console                 serial
controlp                ON
...
pal                     OpenVMS PALcode V1.98-43, Tru64 UNIX PALcode V1.92-33
...
sys_serial_num          4228KSTZA000
tt_allow_login          1
tty_dev                 0
version                 V6.9-2 Nov 18 2004 09:57:20
wwid0                   2001 1 WWID:01000010:6005-08b4-0001-278f-0000-7000-
052f-0000
wwid1
wwid2
wwid3
P00>>>
```

One of the tools that allows console access among other things is ConsoleWorks[3] by TECSys Development Inc. (TDi). Figure 1 shows HTTP access to ConsoleWorks. The port used by ConsoleWorks is 5176, and normally the path to it will be similar to the URL: http://hostname.domain.com:5176/.

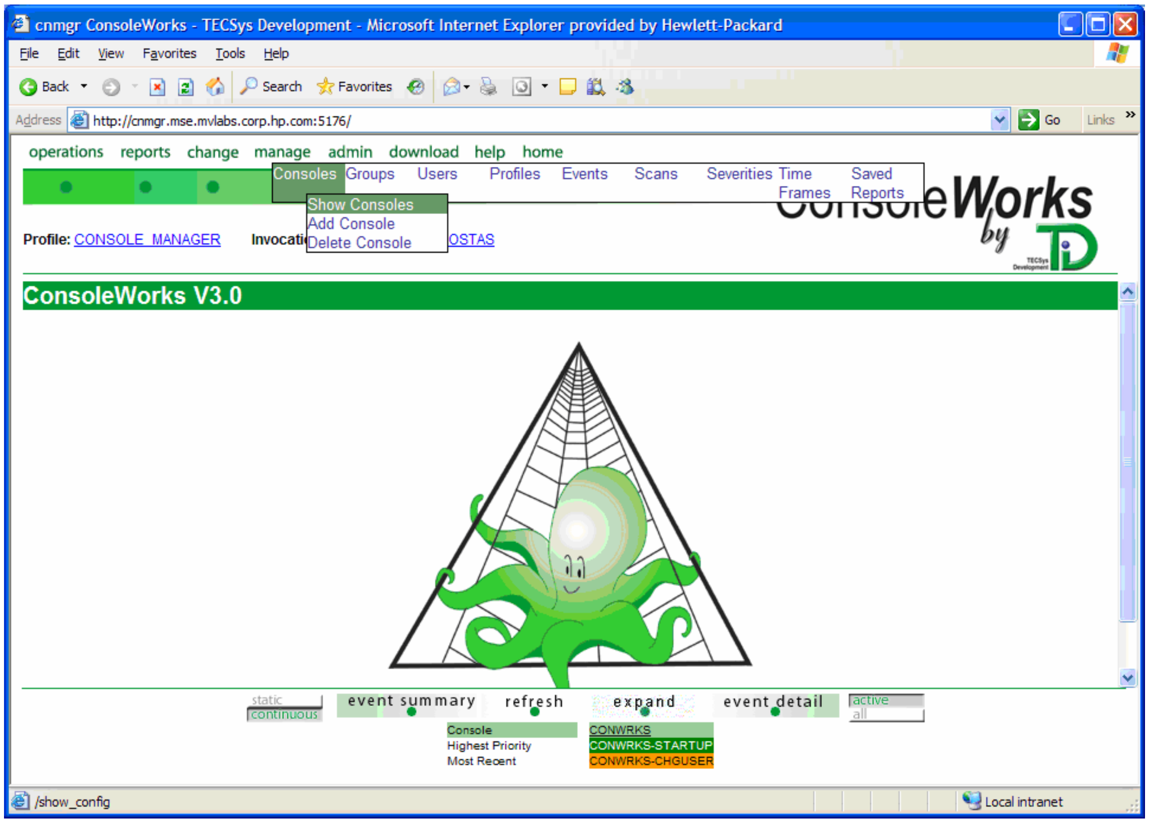
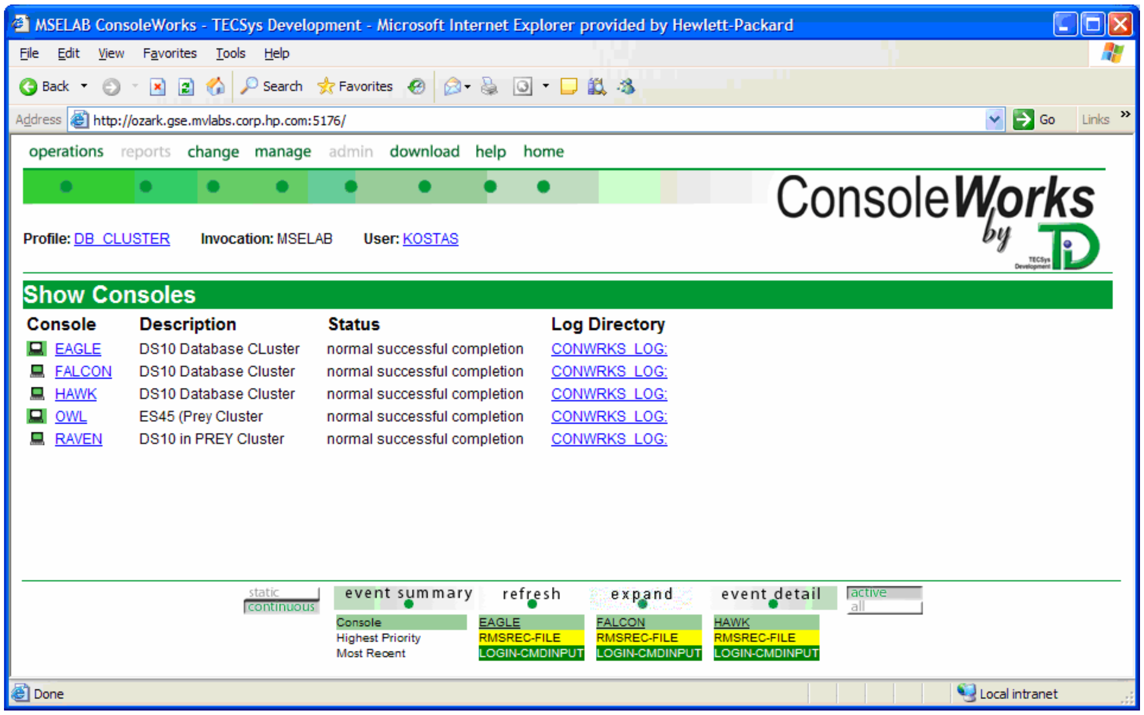

cnmgr ConsoleWorks - TECSys Development - Microsoft Internet Explorer provided by Hewlett-Packard
File Edit View Favorites Tools Help
Back Search Favorites
Address http://cnmgr.mse.mvlabs.corp.hp.com:5176/
Go Links
operations reports change manage admin download help home
Consoles Groups Users Profiles Events Scans Severities Time Frames Saved Reports
Show Consoles
Add Console
Delete Console
Profile: CONSOLE_MANAGER
Invocati
OSTAS
ConsoleWorks V3.0
static continuous
event summary refresh expand event detail
active all
Console
Highest Priority
Most Recent
CONWRKS
CONWRKS-STARTUP
CONWRKS-CHGUSER
/show_config
Local intranet

Figure 1 - ConsoleWorks Web interface

Once connected, choose the top menu option *manage* → *Consoles* → *Show Consoles* to get a list of all configured and available consoles to connect to. The resulting screen will be similar to Figure 2 below.

[3] ConsoleWorks is developed by TECSys Development Inc. (TDi), and it is a Web-based enterprise event monitoring, event management and regulatory compliance software solution. It provides secure remote monitoring and management for enterprise networks, server, devices and applications. Additionally captures, audits and logs console data from these devices. For more details see the URL: http://www.tditx.com/consoleworks.html


MSELAB ConsoleWorks - TECSys Development - Microsoft Internet Explorer provided by Hewlett-Packard
File Edit View Favorites Tools Help
Back Search Favorites
Address http://ozark.gse.mvlabs.corp.hp.com:5176/
Go Links
operations reports change manage admin download help home
ConsoleWorks by TD
static continuous
event summary refresh expand event detail
active all
Done
Local intranet

Profile: DB_CLUSTER Invocation: MSELAB User: KOSTAS

Show Consoles

Console	Description	Status	Log Directory
EAGLE	DS10 Database CLuster	normal successful completion	CONWRKS_LOG:
FALCON	DS10 Database Cluster	normal successful completion	CONWRKS_LOG:
HAWK	DS10 Database Cluster	normal successful completion	CONWRKS_LOG:
OWL	ES45 (Prey Cluster	normal successful completion	CONWRKS_LOG:
RAVEN	DS10 in PREY Cluster	normal successful completion	CONWRKS_LOG:

Console	EAGLE	FALCON	HAWK
Highest Priority	RMSREC-FILE	RMSREC-FILE	RMSREC-FILE
Most Recent	LOGIN-CMDINPUT	LOGIN-CMDINPUT	LOGIN-CMDINPUT

Figure 2 - Available consoles

HP Tru64 UNIX Alpha Console

In HP Tru64 UNIX, two system functions, `getsysinfo(2)` and `setsysinfo(2)`, are available for use by C programs to get and set the console environment variable values. The `getsysinfo(2)` function with the `GSI_PROM_ENV` operation can be used to obtain the current value of the PROM console environment variables. The following small C program shown below – `getcvar.c` – illustrates this capability.

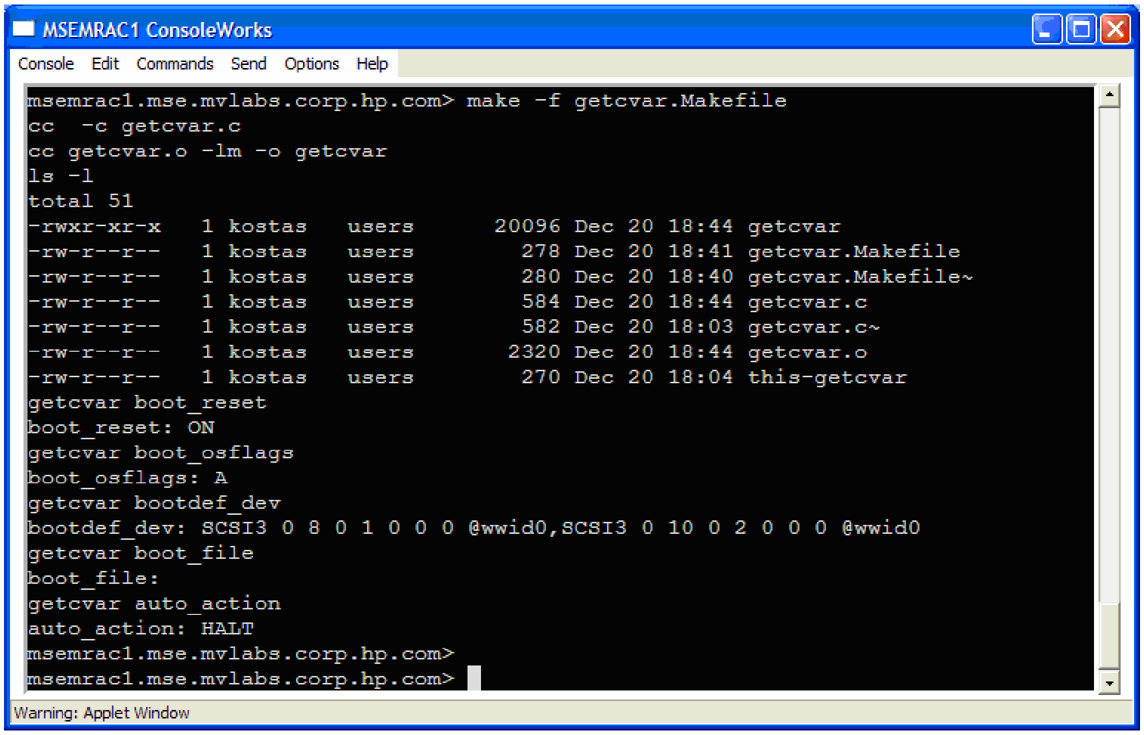
```
msemrac1.mse.mvlabs.corp.hp.com> cat getcvar.c
   /*
    * Module: getcvar.c  Version: X0.0-000 January 31st, 1996 /KGG
    */
   #include <sys/types.h>
   #include <machine/hal_sysinfo.h>
   main(int argc, char **argv)
   {
           char env[132];
           int ret;
           if (argc == 1)  {
                   puts("Usage: getcvar string");
                   exit(1);
           }
           if ((ret = getsysinfo(GSI_PROM_ENV, env, 132, 0, argv[1])) == -1) {
                   perror("getsysinfo");
                   exit(1);
           }
           printf("%s: %s\n", argv[1], env);
           exit(0);
    }
```

For more information on both the `getsysinfo()` and `setsysinfo()` functions please refer to the man pages.

Makefile for the getcvar C program

```
msemrac1.mse.mvlabs.corp.hp.com> cat getcvar.Makefile
#++
# Module: getcvar.Makefile
# Version: X0.0-000 January 31st, 1996 /KGG
#--
OBJS = getcvar.o
getcvar: $(OBJS)
        cc $(OBJS) -lm -o getcvar
        ls -l
        getcvar boot_reset
        getcvar boot_osflags
        getcvar bootdef_dev
        getcvar boot_file
        getcvar auto_action
getcvar.o:
        cc -c getcvar.c
```

Compiling and running the getcvar C program



```
msemrac1.mse.mvlabs.corp.hp.com> make -f getcvar.Makefile
cc  -c getcvar.c
cc getcvar.o -lm -o getcvar
ls -l
total 51
-rwxr-xr-x   1 kostas   users      20096 Dec 20 18:44 getcvar
-rw-r--r--   1 kostas   users        278 Dec 20 18:41 getcvar.Makefile
-rw-r--r--   1 kostas   users        280 Dec 20 18:40 getcvar.Makefile~
-rw-r--r--   1 kostas   users        584 Dec 20 18:44 getcvar.c
-rw-r--r--   1 kostas   users        582 Dec 20 18:03 getcvar.c~
-rw-r--r--   1 kostas   users       2320 Dec 20 18:44 getcvar.o
-rw-r--r--   1 kostas   users        270 Dec 20 18:04 this-getcvar
getcvar boot_reset
boot_reset: ON
getcvar boot_osflags
boot_osflags: A
getcvar bootdef_dev
bootdef_dev: SCSI3 0 8 0 1 0 0 0 @wwid0,SCSI3 0 10 0 2 0 0 0 @wwid0
getcvar boot_file
boot_file:
getcvar auto_action
auto_action: HALT
msemrac1.mse.mvlabs.corp.hp.com>
msemrac1.mse.mvlabs.corp.hp.com>
```

Figure 3 - Compiling and running the getcvar.c program

As with the `getcvar` program, one can write a `setcvar` program that would take a variable name and variable value pair and set it, as in this example:

```
# setcvar boot_reset OFF
```

And in general:

```
# setcvar <console_var_name> <console_var_value>
```

by making the following modifications to the `getcvar` program.

Include the `<machine/prom.h>` header file in addition to the others:

```
#include <machine/prom.h>
```

Define two string variables, `env_name` and `env_value`, and assign the environment variable name and its value to each one, respectively, from the command line:

```
            char *my_name;
            char *env_name, *env_value;

            my_name = argv[0];
            env_name = argv[1];
            env_value = argv[2];
```

Call the system function `setsysinfo()` with the correct parameters:

```
            if ((ret = setsysinfo(SSI_PROM_ENV, env_value, strlen(env_value)+1,
                                  env_name, PROM_CONVERT_TYPE)) == -1) {
               fprintf(stderr,
                     "%s: setsysinfo() failed setting %s to '%s': %s\n",
                     my_name,
                     env_name, env_value,
                     strerror(errno));
               return(-1);
            }
```

For assistance with writing your version of the `setcvar` program contact the author for a working version of the `setcvar` program.

/sbin/consvar Tru64 UNIX program

Another method of getting and setting the values of the console variables involves the `/sbin/consvar` program.

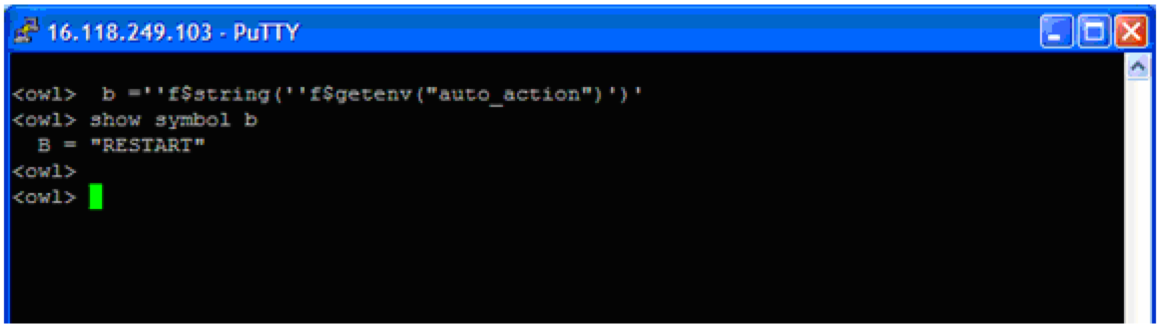
```
kamlia.zk3.dec.com> /sbin/consvar help
Firmware Console Environment Variable Manager
 consvar [-v] [-nc] -g -s -a -l -d [variable] [value]
 -v                      Verbose
 -g variable             Get the value of variable
 -s variable value       Set the value of variable
 -l                      List all variables with their values
 -a                      Save all variables to NV storage
 -d                      Print out exception database data
 -nc                     Do NOT perform conversion
```

Obtain a verbose (`-v` option) of all variables with their values (`-l` option):

```
kamlia.zk3.dec.com> /sbin/consvar -v -l
Firmware Rev: 6.8-13
system fam:34  cpu:8  smm:1814
auto_action = HALT
boot_dev = dsk6(4 paths)
bootdef_dev = dsk6(4 paths)
booted_dev = dsk6
boot_file =
booted_file =
boot_osflags = s
booted_osflags = s
boot_reset = ON
Failed to get dump_dev
enable_audit = ON
license = MU
char_set =
language = 0x36
tty_dev = 0
Failed to get scsiid
Failed to get scsifast
com1_baud = 9600
com1_modem = OFF
com1_flow = SOFTWARE
Failed to get com1_misc
com2_baud = 9600
com2_modem = OFF
com2_flow = SOFTWARE
Failed to get com2_misc
Failed to get password
secure = off
logfail = 0
srm2dev_id =
```

HP OpenVMS Alpha Console

The HP OpenVMS operating system provides two methods of getting the console environment variables: the lexical function `f$getenv()` and the `sys$getenv()` system service library call. Figure 4 illustrates how to use the lexical function `f$getenv()` from the operating system prompt to get the value of the `auto_action` console variable.



```
<owl>  b =''f$string(''f$getenv("auto_action")')'
<owl> show symbol b
  B = "RESTART"
<owl>
<owl>
```

Figure 4 - Using the f$getenv() lexical function

The next example, illustrates how to obtain the value of the console variable `auto_action` from the console using a direct serial connection to the system console:

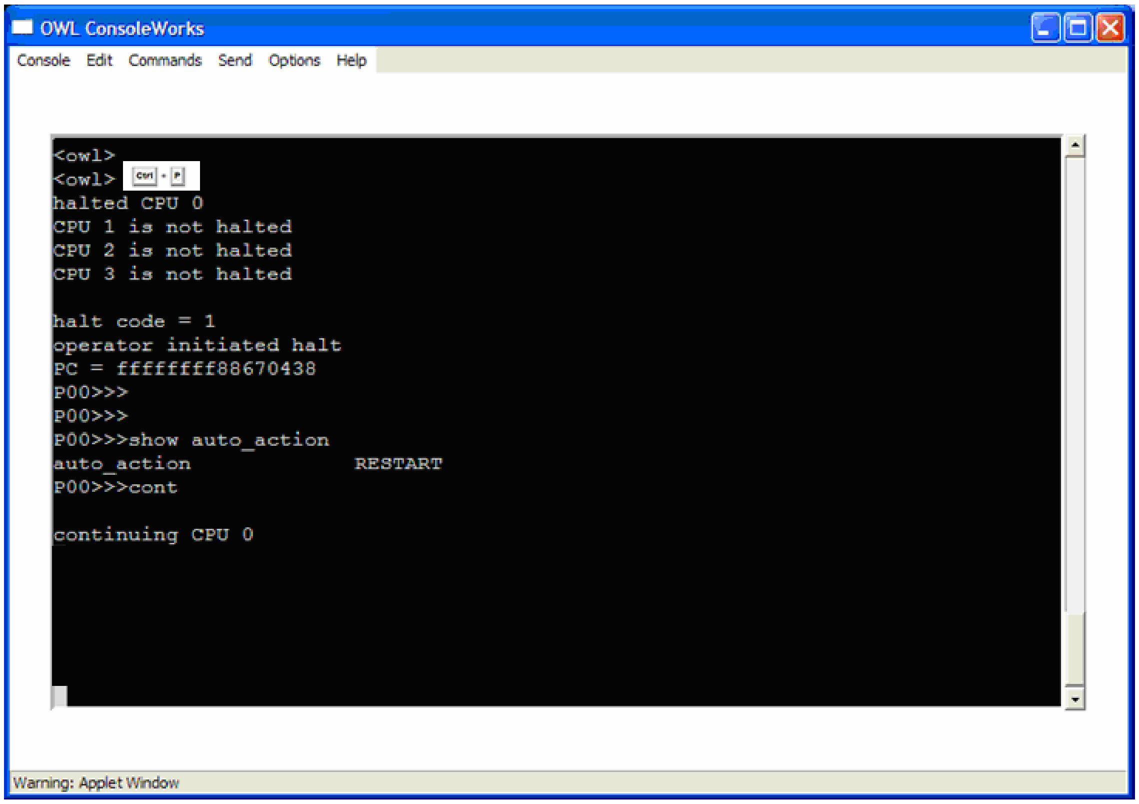
```
Welcome to OpenVMS (TM) Alpha Operating System, Version V7.3-2

Username:    <Ctrl>P

halted CPU 0
CPU 1 is not halted
CPU 2 is not halted
CPU 3 is not halted

halt code = 1
operator initiated halt
PC = ffffffff88670438
P00>>> show auto_action
auto_action             RESTART
P00>>> cont

continuing CPU 0
```

Note the **`<Ctrl>P`** is not the username but the two keyboard keys:

Ctrl + **P**

To enter the SRM console, hold down the Ctrl key and press the P key.

Figure 5 illustrates how to get the value of the Alpha console variable `auto_action` from the console using the ConsoleWorks tool.



```
<owl>
<owl> Ctrl + P
halted CPU 0
CPU 1 is not halted
CPU 2 is not halted
CPU 3 is not halted

halt code = 1
operator initiated halt
PC = ffffffff88670438
P00>>>
P00>>>
P00>>>show auto_action
auto_action             RESTART
P00>>>cont

continuing CPU 0
```

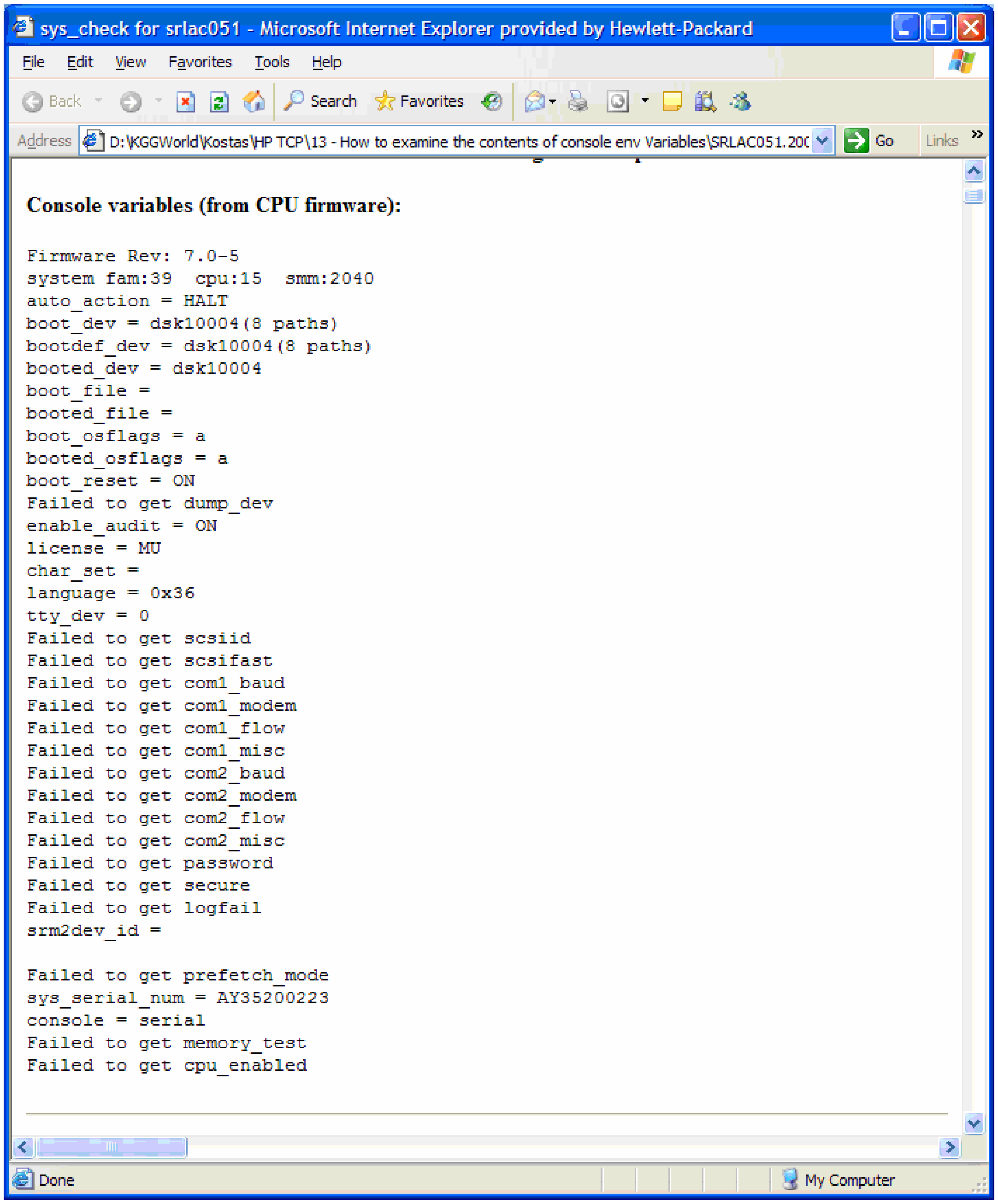
Figure 5 - ConsoleWorks access to Alpha console

Existing HP tools that get the console variables and their values

The two tools used by HP Services professionals to collect AlphaServer system configuration information are the `sys_check` for HP Tru64 UNIX and the `VMS_Check`[4] for HP OpenVMS platforms.

sys_check

The `sys_check` tool gets the console variables on HP Tru64 UNIX systems. The following window (fig. 6) represents a typical report of the information collected by `sys_check`.


sys_check for srlac051 - Microsoft Internet Explorer provided by Hewlett-Packard
File Edit View Favorites Tools Help
Back Search Favorites
Address D:\KGGWorld\Kostas\HP TCP\13 - How to examine the contents of console env Variables\SRLAC051.20(
Go Links
Done
My Computer

Console variables (from CPU firmware):

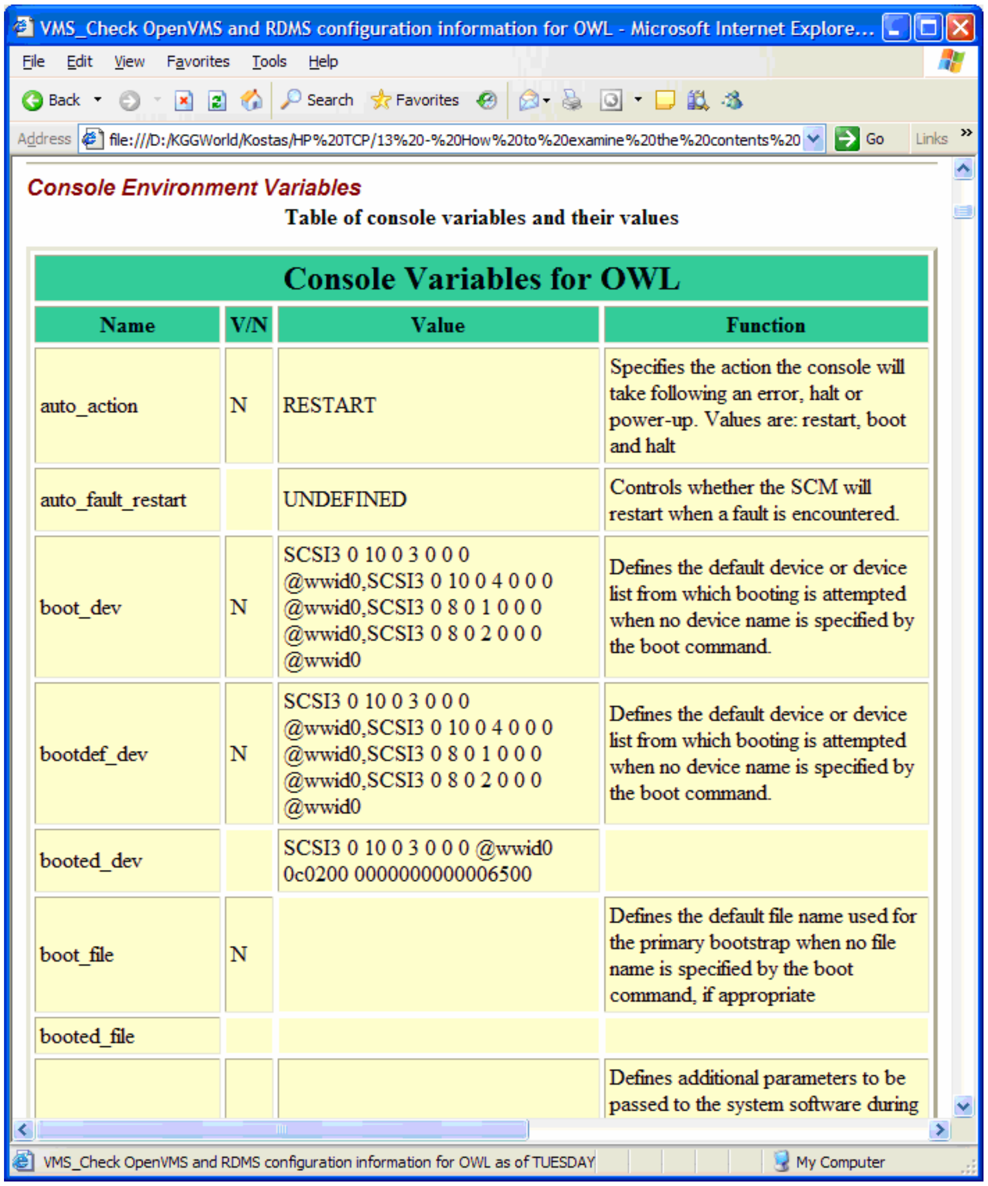
```
Firmware Rev: 7.0-5
system fam:39  cpu:15  smm:2040
auto_action = HALT
boot_dev = dsk10004(8 paths)
bootdef_dev = dsk10004(8 paths)
booted_dev = dsk10004
boot_file =
booted_file =
boot_osflags = a
booted_osflags = a
boot_reset = ON
Failed to get dump_dev
enable_audit = ON
license = MU
char_set =
language = 0x36
tty_dev = 0
Failed to get scsiid
Failed to get scsifast
Failed to get com1_baud
Failed to get com1_modem
Failed to get com1_flow
Failed to get com1_misc
Failed to get com2_baud
Failed to get com2_modem
Failed to get com2_flow
Failed to get com2_misc
Failed to get password
Failed to get secure
Failed to get logfail
srm2dev_id =

Failed to get prefetch_mode
sys_serial_num = AY35200223
console = serial
Failed to get memory_test
Failed to get cpu_enabled
```

Figure 6 - sys_check report on the console variables

[4] For more information on VMS_Check please reference the OpenVMS Technical Journal article http://h71000.www7.hp.com/openvms/journal/v7/vms_check_tool.html or contact the author.

VMS_Check

The `VMS_Check` tool gets the console variables on HP OpenVMS systems. The following window (fig. 7) represents a portion of the information provided by `VMS_Check` on the HTML generated code as it is viewed in Microsoft Internet Explorer.


VMS_Check OpenVMS and RDMS configuration information for OWL - Microsoft Internet Explore...
File Edit View Favorites Tools Help
Back Search Favorites
Address file:///D:/KGGWorld/Kostas/HP%20TCP/13%20-%20How%20to%20examine%20the%20contents%20
Go Links
VMS_Check OpenVMS and RDMS configuration information for OWL as of TUESDAY
My Computer

Console Environment Variables

Table of console variables and their values

Console Variables for OWL

Name	V/N	Value	Function
auto_action	N	RESTART	Specifies the action the console will take following an error, halt or power-up. Values are: restart, boot and halt
auto_fault_restart		UNDEFINED	Controls whether the SCM will restart when a fault is encountered.
boot_dev	N	SCSI3 0 10 0 3 0 0 0 @wwid0,SCSI3 0 10 0 4 0 0 0 @wwid0,SCSI3 0 8 0 1 0 0 0 @wwid0,SCSI3 0 8 0 2 0 0 0 @wwid0	Defines the default device or device list from which booting is attempted when no device name is specified by the boot command.
bootdef_dev	N	SCSI3 0 10 0 3 0 0 0 @wwid0,SCSI3 0 10 0 4 0 0 0 @wwid0,SCSI3 0 8 0 1 0 0 0 @wwid0,SCSI3 0 8 0 2 0 0 0 @wwid0	Defines the default device or device list from which booting is attempted when no device name is specified by the boot command.
booted_dev		SCSI3 0 10 0 3 0 0 0 @wwid0 0c0200 0000000000006500	
boot_file	N		Defines the default file name used for the primary bootstrap when no file name is specified by the boot command, if appropriate
booted_file			
			Defines additional parameters to be passed to the system software during

Figure 7 - VMS_Check report of the console variables

How to update the SRM console

There are several methods available to update the SRM firmware, depending of the Operating System. One method, that is common to all, involves creating a bootable firmware update CD-ROM from an ISO image available from the URL: ftp://ftp.digital.com/pub/DEC/Alpha/firmware/index.html, and then booting the system from the created bootable CD-ROM.

Summary

This article explored the Alpha SRM console and the different ways to access it and to set and retrieve the values of the console environment variables.

References

GRUB – ftp://alpha.gnu.org/pub/gnu/grub/

LILO – http://www.tldp.org/HOWTO/LILO.html

MILO - http://www.tldp.org/HOWTO/MILO-HOWTO/index.html

EFI - http://www.intel.com/technology/efi/

TECSys Development Inc (TDi) - http://www.tditx.com/

Sys_check - http://h30097.www3.hp.com/sys_check/

VMS_Check – To learn more about VMS_Check please see the OpenVMS Technical Journal article at the URL: http://h71000.www7.hp.com/openvms/journal/v7/vms_check_tool.html and to get a copy of the VMS_CHECK tool please download it from the URL: http://h71000.www7.hp.com/openvms/journal/v7/vms_check.zip . Please email the author with any questions or comments regarding VMS_Check.

hp AlphaServer ES47/Es80/GS1280 Server Management SRM Console Reference Guide V1.0 - http://h18002.www1.hp.com/alphaserver/download/srm_reference.pdf

Alpha Systems Firmware Updates - ftp://ftp.digital.com/pub/DEC/Alpha/firmware/index.html

Alpha Architecture Reference Manual – http://lab46.corning-cc.edu/haas/spring2005/asm/public/doc/Alpha_Architecture_Reference_Manual.pdf

Acknowledgements

The author wishes to acknowledge his GSE-MSE colleagues Laurence Oberman and Joe Bowker for providing access to the ConsoleWorks configurations on Tru64 Unix and OpenVMS systems in the MSE interoperability lab. In addition, I would like to thank Ken W. Campbell, Bayard W. Wenzel, and Ken Sullivan, also from GSE-MSE, for their review of this article and for their valuable feedback and contributions.

About the Author

Kostas G. Gavrielidis works in GSE-MSE which is part of HP Services Customer Support and has been at HP for more than 22 years. Currently, and for the last 12 years, he is involved with the MSE proactive consulting projects for our customer production Database Management systems, and he works on the analysis and performance improvements for SAP R/3, Oracle, Rdb, Ingres, SYBASE, SQL Server on UNIX, Linux, OpenVMS, and Windows platforms.

For more information

Please contact the author with any questions or comments regarding this article or the VMS_Check tool.

To get to the latest issue of the OpenVMS Technical Journal, go to: http://www.hp.com/go/openvms/journal.

OpenVMS Technical Journal V8

CHARON-VAX Performance Benchmarks

Bruce Claremont, Software Migration and OpenVMS Consultant

Overview

Migration Specialties International recently conducted independent benchmark and stress testing of the CHARON-VAX emulator. This article contains our findings.

Introduction

Migration Specialties is a small, privately held corporation that specializes in HP OpenVMS porting, software migration, and complementary services. We became involved with the CHARON-VAX emulation product last year. CHARON-VAX is a software-based VAX hardware emulator produced by Software Resources International that runs on Windows-based servers like the HP ProLiant. More product information is available on the HP DSPP site under VAX Replacement.

After working with the emulator, I became curious as to how it would perform against native OpenVMS hardware. I also wanted to stress the product and see how it held up. Migration Specialties has VAX, Alpha, and Integrity servers in-house. We also have a comprehensive, automated DTM test suite that supports our Migration RPG product. Thus, the tools were at hand to benchmark and stress test the CHARON-VAX emulator.

Baseline

I used our MicroVAX 3100-80 as the baseline for these tests. All tests compare performance against the MicroVAX. I ported the MicroVAX environment to the CHARON-VAX environment using a set of image backups and restores.

Equipment

Native OpenVMS environment testing was conducted on the MicroVAX 3100, DECsystem 3000-300X, Digital AlphaServer 1000As, and HP Integrity RX2600. Simulated VAX testing using CHARON-VAX was conducted on an in-house Compaq Presario R3000 equipped with an Intel Pentium 4 Hyper-threaded processor and a SuperMicro server equipped with dual Xeon EM64T processors. More information on the hardware configurations is provided in the Performance Results and Testing Details sections.

Performance Tests

I was interested in two simple criteria: how fast was each system as measured by process elapsed time and CPU utilization. Testing was conducted using our Migration RPG quality assurance suite. The suite consists of eight interactive test sets that primarily check compiler functionality and accuracy. Consequently, the tests are compile-intensive, meaning they use more CPU resources than most production applications. By the same token, they generate less I/O than most production applications. More information on the test sets is provided in the Testing Details section.

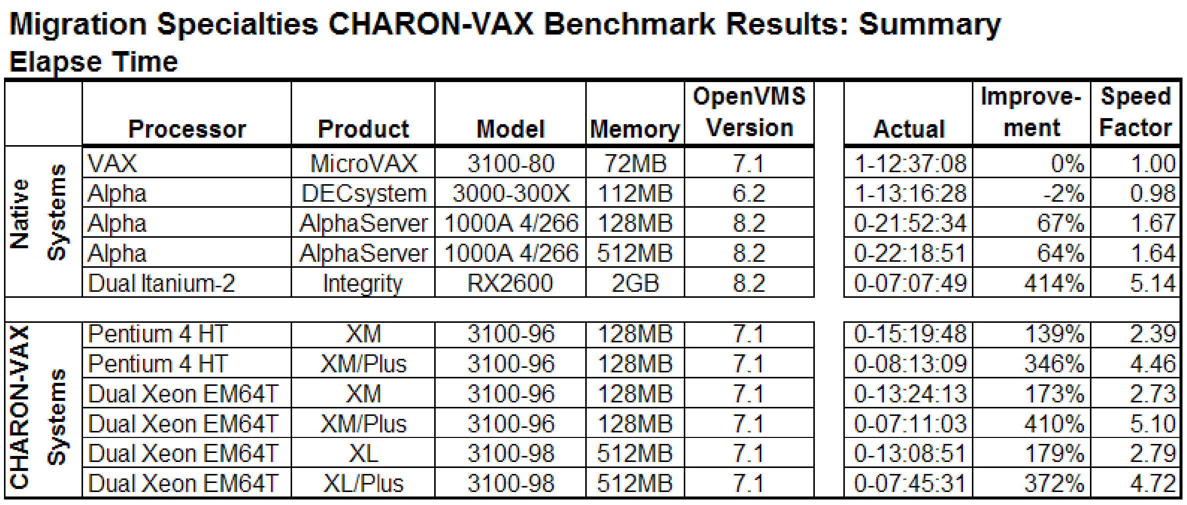
Performance Results

The performance results of the VAX emulator are impressive. I experienced no problems with the emulator during all of the testing.

I tested two versions of CHARON-VAX, XM and XL. I tested both the base and enhanced modes within each version. Enhanced mode is indicated by the ***Plus*** designation in the Product column. Enhanced mode deploys an accelerator that optimizes OpenVMS instruction processing.

The following two tables summarize the results. Each table leads off with the native OpenVMS processors, then follows with the simulated VAX results. The first set of columns in the table summarizes the hardware and OpenVMS version used. The second set of columns summarizes either elapsed time or CPU utilization. Performance and speed measurements use the MicroVAX 3100-80 as a baseline.

Important Note: The memory listed under the emulated VAX systems is the emulated VAX memory with which the simulated VAX was configured. It is not the physical memory present in the server. The Compaq system was equipped with 1.28 GB of memory and the Intel system was equipped with 2 GB of memory. Likewise, the models listed in the emulated VAX results are the emulated VAX model,

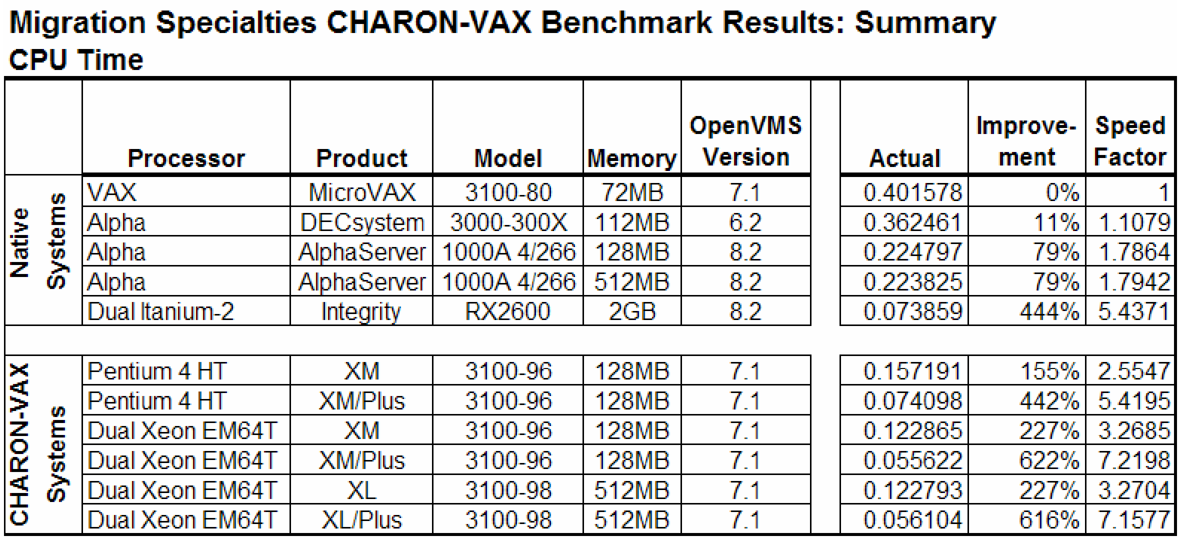


Migration Specialties CHARON-VAX Benchmark Results: Summary
Elapse Time

	Processor	Product	Model	Memory	OpenVMS Version	Actual	Improve-ment	Speed Factor
Native Systems	VAX	MicroVAX	3100-80	72MB	7.1	1-12:37:08	0%	1.00
	Alpha	DECsystem	3000-300X	112MB	6.2	1-13:16:28	-2%	0.98
	Alpha	AlphaServer	1000A 4/266	128MB	8.2	0-21:52:34	67%	1.67
	Alpha	AlphaServer	1000A 4/266	512MB	8.2	0-22:18:51	64%	1.64
	Dual Itanium-2	Integrity	RX2600	2GB	8.2	0-07:07:49	414%	5.14
CHARON-VAX Systems	Pentium 4 HT	XM	3100-96	128MB	7.1	0-15:19:48	139%	2.39
	Pentium 4 HT	XM/Plus	3100-96	128MB	7.1	0-08:13:09	346%	4.46
	Dual Xeon EM64T	XM	3100-96	128MB	7.1	0-13:24:13	173%	2.73
	Dual Xeon EM64T	XM/Plus	3100-96	128MB	7.1	0-07:11:03	410%	5.10
	Dual Xeon EM64T	XL	3100-98	512MB	7.1	0-13:08:51	179%	2.79
	Dual Xeon EM64T	XL/Plus	3100-98	512MB	7.1	0-07:45:31	372%	4.72

Figure 1 - Summarized elapsed times for native OpenVMS systems and emulated VAX systems.

not the underlying Windows servers.



Migration Specialties CHARON-VAX Benchmark Results: Summary
CPU Time

	Processor	Product	Model	Memory	OpenVMS Version	Actual	Improve-ment	Speed Factor
Native Systems	VAX	MicroVAX	3100-80	72MB	7.1	0.401578	0%	1
	Alpha	DECsystem	3000-300X	112MB	6.2	0.362461	11%	1.1079
	Alpha	AlphaServer	1000A 4/266	128MB	8.2	0.224797	79%	1.7864
	Alpha	AlphaServer	1000A 4/266	512MB	8.2	0.223825	79%	1.7942
	Dual Itanium-2	Integrity	RX2600	2GB	8.2	0.073859	444%	5.4371
CHARON-VAX Systems	Pentium 4 HT	XM	3100-96	128MB	7.1	0.157191	155%	2.5547
	Pentium 4 HT	XM/Plus	3100-96	128MB	7.1	0.074098	442%	5.4195
	Dual Xeon EM64T	XM	3100-96	128MB	7.1	0.122865	227%	3.2685
	Dual Xeon EM64T	XM/Plus	3100-96	128MB	7.1	0.055622	622%	7.2198
	Dual Xeon EM64T	XL	3100-98	512MB	7.1	0.122793	227%	3.2704
	Dual Xeon EM64T	XL/Plus	3100-98	512MB	7.1	0.056104	616%	7.1577

Figure 2: Summarized CPU times for native OpenVMS systems and emulated VAX systems.

Acknowledgements
I would like to thank Reynolds Technical Services and Quayle Consulting for loaning us the CHARON-VAX licenses used in the benchmark process. Intel Corporation generously provided the SuperMicro server hardware using in testing.

Testing Details

The following sections provide more details concerning the test hardware, system configurations, test sets, and testing protocols.

Test Hardware

- DEC MicroVAX 3100-80, VAX Mariah processor (12 VUP), 72 MB memory, two RZ29L-AS disk drives.
- DECsystem 3000-300X, Alpha 21064 175 Mhz processor, 112 MB memory, two RZ28M disk drives.
- Digital AlphaServer 1000A 4/266, Alpha 21064 266 Mhz processor, 128 MB memory, two RZ29B disk drives.
- Digital AlphaServer 1000A 4/266, Alpha 21064 266 Mhz processor, 512 MB memory, two RZ29B disk drives.
- HP Integrity RX2600, dual Itanium2 1.4 Ghz processors, 2 GB memory, two Ultra320 15,000rpm disk drives.
- Compaq Presario R3000 Laptop, Intel Pentium 4HT 3.0 Ghz processor, 1.28 GB memory, single 4200 rpm disk drive.
- Supermicro SuperServer 6024H-82R, dual Intel Xeon EM64T processors, 2GB memory, single Ultra320 15,000 rpm disk drive.

Hardware Note: SRI does not officially support CHARON-VAX on single CPU systems or 64-bit Windows systems. Hence, our test beds would not be appropriate production systems.

System Configurations

No special tuning was done on any of the systems used in testing. The native OpenVMS systems are maintained using AUTOGEN with feedback. No special adjustments were made to system parameters for these tests.

The two Windows servers were new, each running a fresh install of the Windows O/S. The systems were configured per published CHARON-VAX recommendations, with no anti-virus, automatic update, or anti-spyware processes running.

The CHARON-VAX configuration on each of the Windows servers was an image copy of our MicroVAX 3100-80. All OpenVMS system parameters were identical to the MicroVAX 3100-80 from which the software was sourced.

Each of the native OpenVMS systems uses a dedicated system drive that supports the O/S, page file, and swap file. The DTM test suite was located on a secondary drive to reduce I/O loads. The VAX emulation systems each contained a single disk drive. The Windows O/S, VAX emulation software, and VAX disk container files were all loaded on this one drive. In the case of the VAX emulator, this is not a recommended configuration; it is simply what I had available for testing.

- MicroVAX 3100, OpenVMS 7.1, DTM version 4.0-4, dedicated system drive, DTM test suite run from second drive.
- DECsystem 3000-300X: OpenVMS 6.2, DTM version 4.0-4, dedicated system drive, DTM test suite run from second drive
- AlphaServer 1000As: OpenVMS 8.2, DTM version 4.1, dedicated system drive, DTM test suite run from second drive
- Integrity RX2600: OpenVMS 8.2, DTM version 4.2, dedicated system drive, DTM test suite run from second drive.
- Compaq Presario: Windows XP SP2. VAX emulation software and VAX disk container files supported on single hard drive. CHARON-VAX version 3.0.39, OpenVMS 7.1, DTM version 4.0-4, system drive emulated as an RZ74 container file, DTM drive emulated as an RZ73 container file.
- Supermicro SuperServer: Windows Server 2003. VAX emulation software and VAX disk container files supported on single hard drive. CHARON-VAX version 3.0.39, OpenVMS 7.1, DTM version 4.0-4, system drive emulated as an RZ74 container file, DTM drive emulated as an RZ73 container file.

Test Sets

Our Migration RPG DTM test suite consists of eight automated test sets. The test sets comprise both batch and interactive test processes. The test suites primarily exercise the Migration RPG compiler by compiling and linking programs. Thus, the test suites tend to be CPU intensive while placing little demand on memory or I/O resources.

The two Alpha 1000 systems and CHARON-VAX XL tests support these observations. One Alpha 1000 is equipped with 128 MB of memory while the other is equipped with 512 MB of memory. This had little impact on the benchmarks. Likewise, CHARON-VAX XL permits an emulated VAX to be configured with 512 MB of memory versus the 128 MB available under XM. Again, the difference in performance is negligible.

About 10% of the tests run production style interactive and batch processes. Test set 3 is the most realistic from a production perspective. It consists of a series of interactive processes that acquire, process, update, and display information from general ledger, payroll, inventory, and similar applications.

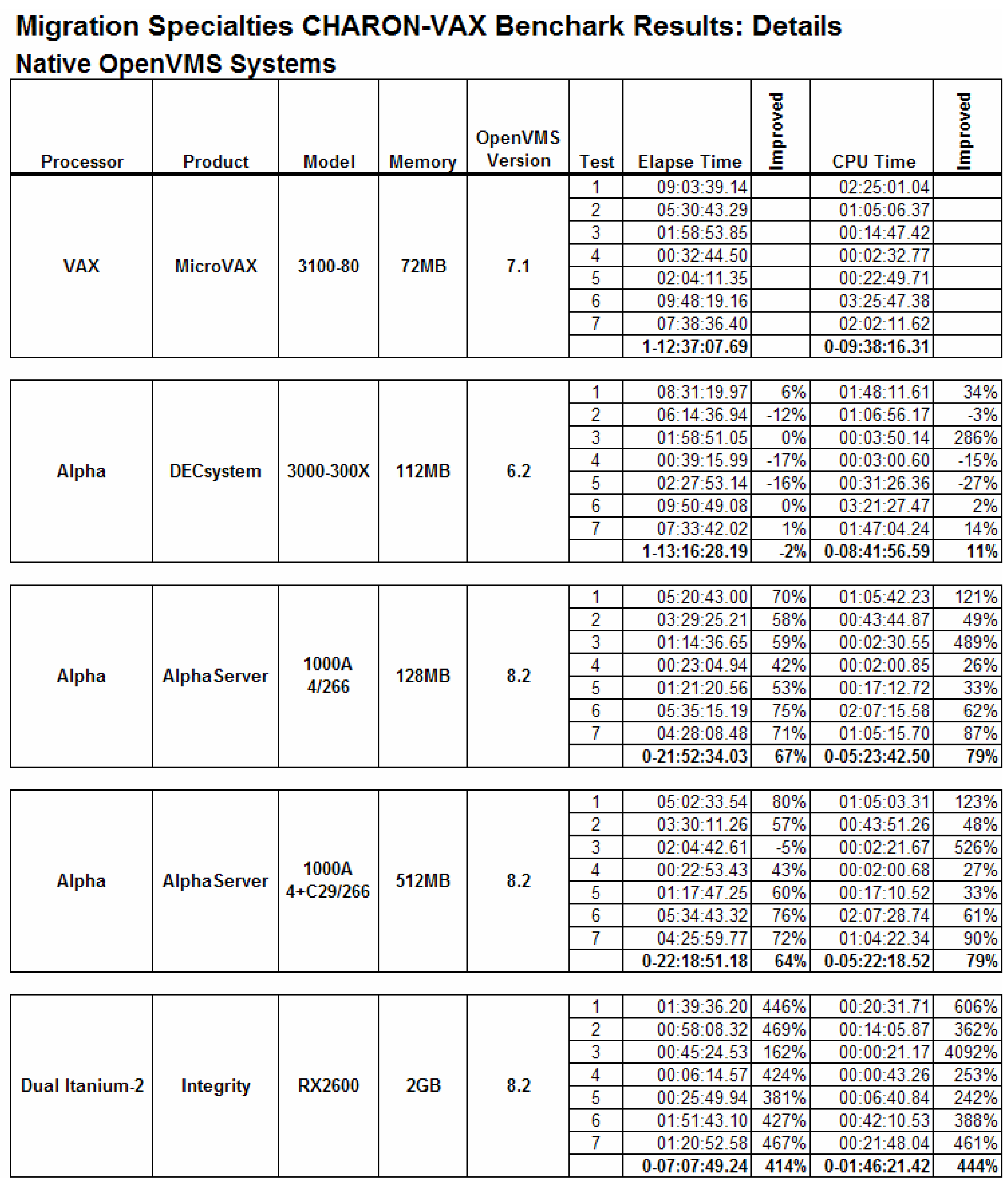
The results from each test set are listed in the Test Results Details section.

Testing Protocol

The testing protocol was submission of all eight test sets to a batch queue capable of executing six jobs simultaneously. No other user processes were run on the system during the benchmark tests.

Test Results Details

The following tables provide the elapsed time and CPU time used by each test set in the DTM test suite.

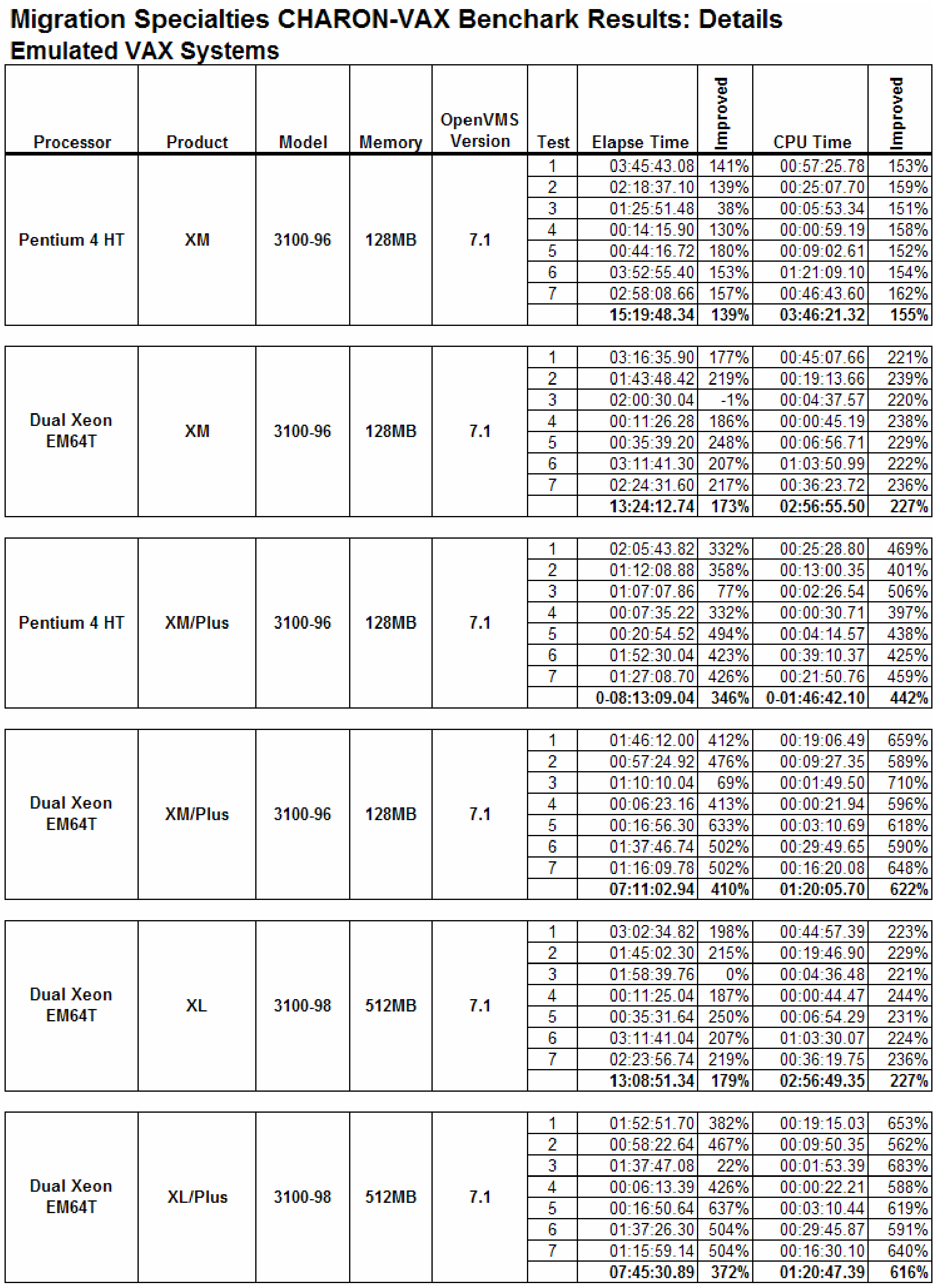


Migration Specialties CHARON-VAX Benchark Results: Details

Native OpenVMS Systems

Processor	Product	Model	Memory	OpenVMS Version	Test	Elapse Time	Improved	CPU Time	Improved
VAX	MicroVAX	3100-80	72MB	7.1	1	09:03:39.14		02:25:01.04	
					2	05:30:43.29		01:05:06.37	
					3	01:58:53.85		00:14:47.42	
					4	00:32:44.50		00:02:32.77	
					5	02:04:11.35		00:22:49.71	
					6	09:48:19.16		03:25:47.38	
					7	07:38:36.40		02:02:11.62	
						1-12:37:07.69		**0-09:38:16.31**	
Alpha	DECsystem	3000-300X	112MB	6.2	1	08:31:19.97	6%	01:48:11.61	34%
					2	06:14:36.94	-12%	01:06:56.17	-3%
					3	01:58:51.05	0%	00:03:50.14	286%
					4	00:39:15.99	-17%	00:03:00.60	-15%
					5	02:27:53.14	-16%	00:31:26.36	-27%
					6	09:50:49.08	0%	03:21:27.47	2%
					7	07:33:42.02	1%	01:47:04.24	14%
						1-13:16:28.19	**-2%**	**0-08:41:56.59**	**11%**
Alpha	AlphaServer	1000A 4/266	128MB	8.2	1	05:20:43.00	70%	01:05:42.23	121%
					2	03:29:25.21	58%	00:43:44.87	49%
					3	01:14:36.65	59%	00:02:30.55	489%
					4	00:23:04.94	42%	00:02:00.85	26%
					5	01:21:20.56	53%	00:17:12.72	33%
					6	05:35:15.19	75%	02:07:15.58	62%
					7	04:28:08.48	71%	01:05:15.70	87%
						0-21:52:34.03	**67%**	**0-05:23:42.50**	**79%**
Alpha	AlphaServer	1000A 4+C29/266	512MB	8.2	1	05:02:33.54	80%	01:05:03.31	123%
					2	03:30:11.26	57%	00:43:51.26	48%
					3	02:04:42.61	-5%	00:02:21.67	526%
					4	00:22:53.43	43%	00:02:00.68	27%
					5	01:17:47.25	60%	00:17:10.52	33%
					6	05:34:43.32	76%	02:07:28.74	61%
					7	04:25:59.77	72%	01:04:22.34	90%
						0-22:18:51.18	**64%**	**0-05:22:18.52**	**79%**
Dual Itanium-2	Integrity	RX2600	2GB	8.2	1	01:39:36.20	446%	00:20:31.71	606%
					2	00:58:08.32	469%	00:14:05.87	362%
					3	00:45:24.53	162%	00:00:21.17	4092%
					4	00:06:14.57	424%	00:00:43.26	253%
					5	00:25:49.94	381%	00:06:40.84	242%
					6	01:51:43.10	427%	00:42:10.53	388%
					7	01:20:52.58	467%	00:21:48.04	461%
						0-07:07:49.24	**414%**	**0-01:46:21.42**	**444%**

Figure 3: Native OpenVMS system test results.



Migration Specialties CHARON-VAX Benchark Results: Details
Emulated VAX Systems

Processor	Product	Model	Memory	OpenVMS Version	Test	Elapse Time	Improved	CPU Time	Improved
Pentium 4 HT	XM	3100-96	128MB	7.1	1	03:45:43.08	141%	00:57:25.78	153%
					2	02:18:37.10	139%	00:25:07.70	159%
					3	01:25:51.48	38%	00:05:53.34	151%
					4	00:14:15.90	130%	00:00:59.19	158%
					5	00:44:16.72	180%	00:09:02.61	152%
					6	03:52:55.40	153%	01:21:09.10	154%
					7	02:58:08.66	157%	00:46:43.60	162%
						15:19:48.34	**139%**	**03:46:21.32**	**155%**
Dual Xeon EM64T	XM	3100-96	128MB	7.1	1	03:16:35.90	177%	00:45:07.66	221%
					2	01:43:48.42	219%	00:19:13.66	239%
					3	02:00:30.04	-1%	00:04:37.57	220%
					4	00:11:26.28	186%	00:00:45.19	238%
					5	00:35:39.20	248%	00:06:56.71	229%
					6	03:11:41.30	207%	01:03:50.99	222%
					7	02:24:31.60	217%	00:36:23.72	236%
						13:24:12.74	**173%**	**02:56:55.50**	**227%**
Pentium 4 HT	XM/Plus	3100-96	128MB	7.1	1	02:05:43.82	332%	00:25:28.80	469%
					2	01:12:08.88	358%	00:13:00.35	401%
					3	01:07:07.86	77%	00:02:26.54	506%
					4	00:07:35.22	332%	00:00:30.71	397%
					5	00:20:54.52	494%	00:04:14.57	438%
					6	01:52:30.04	423%	00:39:10.37	425%
					7	01:27:08.70	426%	00:21:50.76	459%
						0-08:13:09.04	**346%**	**0-01:46:42.10**	**442%**
Dual Xeon EM64T	XM/Plus	3100-96	128MB	7.1	1	01:46:12.00	412%	00:19:06.49	659%
					2	00:57:24.92	476%	00:09:27.35	589%
					3	01:10:10.04	69%	00:01:49.50	710%
					4	00:06:23.16	413%	00:00:21.94	596%
					5	00:16:56.30	633%	00:03:10.69	618%
					6	01:37:46.74	502%	00:29:49.65	590%
					7	01:16:09.78	502%	00:16:20.08	648%
						07:11:02.94	**410%**	**01:20:05.70**	**622%**
Dual Xeon EM64T	XL	3100-98	512MB	7.1	1	03:02:34.82	198%	00:44:57.39	223%
					2	01:45:02.30	215%	00:19:46.90	229%
					3	01:58:39.76	0%	00:04:36.48	221%
					4	00:11:25.04	187%	00:00:44.47	244%
					5	00:35:31.64	250%	00:06:54.29	231%
					6	03:11:41.04	207%	01:03:30.07	224%
					7	02:23:56.74	219%	00:36:19.75	236%
						13:08:51.34	**179%**	**02:56:49.35**	**227%**
Dual Xeon EM64T	XL/Plus	3100-98	512MB	7.1	1	01:52:51.70	382%	00:19:15.03	653%
					2	00:58:22.64	467%	00:09:50.35	562%
					3	01:37:47.08	22%	00:01:53.39	683%
					4	00:06:13.39	426%	00:00:22.21	588%
					5	00:16:50.64	637%	00:03:10.44	619%
					6	01:37:26.30	504%	00:29:45.87	591%
					7	01:15:59.14	504%	00:16:30.10	640%
						07:45:30.89	**372%**	**01:20:47.39**	**616%**

Figure 4: Emulated VAX system test results.

For more information

More information about VAX emulation, porting OpenVMS systems, and Migration Specialties services can be found at www.MigrationSpecialties.com.

About the Author

Bruce Claremont has a degree in Computer Science and is a certifiable VMS bigot, having worked with OpenVMS since 1983. In addition to OpenVMS skills, Bruce understands legacy programming languages like RPG and DIBOL and knows how to code in Macro-32. He also knows a thing or two about software migration. He has worked all sides of the fence, as a customer, software engineer, system manager, support specialist, and project manager. He founded Migration Specialties International, Inc. in 1992 and, when not dealing with all the distractions that come with owning a business and being married, continues to deliver OpenVMS and software migration related services.

OpenVMS Technical Journal V8

HP PERFDAT: A New Performance Solution for OpenVMS

Wolfgang Burger, Technical Consultant HP Service

Ewald Pieber, Solution Architect HP Service & OpenVMS Ambassador

Manfred Kaser, Technical Consultant HP Service & OpenVMS Ambassador

John Dite, Technical Consultant Compinia Gmbh & Co. KG

Overview

This paper presents the OpenVMS performance and capacity planning solution called HP PERFDAT. HP PERFDAT performance solution for OpenVMS provides an unprecedented level of insight into multi-system performance. A complete suite of highly automated collection, filtering, charting and trend analysis capabilities provide the user with accurate and complete performance information for effective performance lifecycle management. The user interface was developed in close cooperation with customers in order to keep performance data analysis simple and intuitive and to enable the user to pinpoint performance problems and to identify their cause without OpenVMS internals knowledge. This article describes the basic concepts, main components and highlights the most important features of HP PERFDAT.

The Performance Management Process

Long term measurement and observation of your system is the key to understanding how well the system performs and is invaluable in identifying potential performance problems before they become so serious that the system grinds to a halt so that it negatively affects your business. Thus, performance measurement should become a routine process of monitoring and measuring your systems to assure good performance through deliberate planning and resource management.

Performance management involves:

- *Systematically collecting system data*
- *Gathering and analyzing the data*
- *Evaluating trends*

- *Archiving data to maintain a performance history*

You will often observe trends and thus be able to address performance issues before they become serious and adversely affect your business operations. Should an unforeseen problem occur, your historical data will likely prove invaluable for pinpointing the cause and rapidly and effectively resolving the problem. Without past data from your formerly well-performing system, you may have no basis upon which to judge the value of the metrics you collect on your poorly performing system. Without historical data you are guessing; resolution will take much longer and cost far more.

The preceding is the key initial statement in the *HP OpenVMS Performance Management* manual. Similar statements can be found in any document seriously dealing with performance and capacity management (e.g., ITIL process).

A prerequisite for effective OpenVMS performance management is to gather accurate and complete performance information of all OpenVMS subsystems. If performance measurements are inaccurate or incomplete, it is very likely that performance root-cause analysis will fail or lead to wrong conclusions.

Accurate and complete performance data is a prerequisite but it is not sufficient for effective performance management. Trend evaluation and data archiving to maintain a performance history have to be highly automated. Even if just one data file per day and system gets created by any OpenVMS performance data collector and your environment consists of only three nodes this will result in more than 1000 data files per year. If data management and trend evaluation is not highly automated, this has to be done manually, which may involve importing the performance data file to a utility such as Excel for charting. In case the amount of data in the performance data file exceeds the amount of data that can be processed by the target utility, which is very likely, the data file has to be pre-processed. You will probably have to copy data files from where the performance data is collected to the node where you analyze the data and to the node where you finally archive the data. Such manual tasks are time consuming and, consequently, costly activities. In addition, it may well happen that these manual trend and data management activities get postponed due to higher priority system management tasks. This can lead to situations where performance history is not immediately available when required, or huge amounts of data have to be processed manually in advance or, the worst case of all, historical data is lost.

The Birth of HP PERFDAT

"High available and high-performance IT services are critical to our business. Thus, we need a performance solution that supports the performance management process as well as the root-cause analysis of performance incidents without explicit expert knowledge time and be cost efficient . . . Such a performance solution has to be "plug and play" and should perform performance management tasks like trending highly automated, reliable and without any need of system management intervention . . . Performance data of our systems have to be available immediately when they are needed to be analyzed for any reason without any preceding data management or data pre-processing activities."

This was the key statement from the head of IT-operations of Austrian Lotteries – Thomas Müller-Guttenbrunn – when HP and Austrian Lotteries first discussed the key requirements that a performance solution for OpenVMS should fulfill in order to provide added value for their system management.

This discussion was triggered by Austrian Lotteries in 2003 when they started evaluating new performance solutions for OpenVMS since they found that their existing performance solution did not fulfill the basic prerequisite to collect accurate and complete performance information of all sub-systems of the most current OpenVMS version (OpenVMS V7.3-1 AXP) at that time. They complained especially about missing XFC statistics and that the performance data provided for the I/O sub-system were sometimes questionable which in turn made performance root-cause analysis difficult and sometimes impossible.

The result of that discussion was the requirements list shown below. After cross-checking the list with the ideas related to performance management of several other OpenVMS customers it was used to evaluate alternative performance solutions:

- High resolution performance data collection for easy root-cause analysis: Especially when analyzing performance issues of the I/O sub-system it is very often of special interest to know which process causes heavy I/O load on a device or a specific file. None of the available performance data collectors available in 2003 provided this kind of information.

- Completeness of data: The data collector has to provide sufficient performance information about all sub-systems of OpenVMS including XFC and LAN and network protocol support.
- Plug and play: Once the performance solution is installed data has to be collected and all performance management related tasks like trending and data archiving have to be performed automatically to maintain a performance history based on predefined profiles, unattended, and without any need of additional customization work.
- Easy to manage and control.
- Online rule based performance alerting: Online performance alerting has to support system management to detect performance anomalies even though their impact does not slow down the overall system performance significantly so that this remains transparent to the end-user.
- Automated data management without any system management intervention.
- The ability to manage huge amounts of data (> 1 terabyte).
- Single point and transparent performance data access regardless of where the performance data is stored within the whole environment via a single common interface.
- Best practice workflow support based on a variety of statistical functions for any kind of performance analysis task in order to:
  - Reduce analysis time.
  - Receive feedback about what is going on without expert knowledge.
- Analysis tool that does not depend on the source data format -- adhering to the principle of "Analyze what you get."
- Data analysis without data pre-processing.
- Automatic trend and capacity reporting.
- Archive and housekeeping functionality.
- Open interface to map/import data from additional data sources (e.g., database, application, storage controllers and so forth) to guarantee collaboration with other performance data collection utilities.
- Performance data export capability to CSV files to guarantee collaboration with existing performance analysis utilities and charting tools. The format of a CSV export file (date/time format, list separator and decimal symbol) will be freely definable to avoid re-formatting the CSV export file before it can be used as input for a dedicated utility (e.g. Excel – CSV input format accepted depends on the regional settings).
- Data analysis will not depend explicitly or implicitly on the start time nor on the sample interval of any data collection.
- Easy data transfer of the performance database, or parts of it, for offline analysis.
- Up- and backward data compatibility.
- State of the art graphical GUI for data analysis:
  - Easy to handle.
  - Intuitive.
  - Easy data navigation.
  - Online descriptions for all statistics available.
  - State-of-the-art graphical features like:
    - Stack/unstack functionality.
    - Zoom in/out.
    - Shift left/right.
    - Data scanning.
    - Ability to scale graphs separately.
    - Auto, native, and manual scaling capability.
    - Data overlay capability (graphs of different time periods can be overlapped to allow visual comparison).
  - Correlation- and deviation analysis capability.
  - Multi window support for multi screen systems.
  - Export capability to Excel.
- Full cluster analysis capability.
- No dependency on any layered product except those available on the OpenVMS installation media.
- No dependency on any third-party product or any kind of shareware/freeware.

Austrian Lotteries tested T4 and ECP as alternatives to their current solution. None of these OpenVMS performance solutions fulfilled all the customer's requirements to the full extend. Thus, HP Austria decided to develop a new performance solution for OpenVMS in close cooperation with the customer.

This was the birth of HP PERFDAT performance solution for OpenVMS.

HP PERFDAT Design Considerations and Challenges

When we look at the performance management process and at the requirements listed above, it is obvious that any performance solution consists of two major categories of components – those that collect performance data and those that process performance data (data trending, performance history maintenance, and so on) in a highly automated fashion. From the software design point of view, the key requirements are different for the data collecting and data processing components.

Performance data collectors have to provide complete and high-resolution performance data consuming as little system resources as possible. There is nothing worse in the context of performance data collection than that the data collecting process becomes the top consumer of system resources but does not provide the level of insight required.

When we designed HP PERFDAT, the top requirement for the data processing components was reusability. Reusability in this context means that a service provided by any of the data processing components can be applied to any data source. This can only be achieved if performance data is provided in a standardized format to the data processing service. Thus, data processing services have to be completely decoupled from the file and record structure of the data files provided by the performance data collectors. It was clear to us, that if we were able to decouple the data processing services from the source data format data access, problems related to version incompatibilities would never arise and the set of data processing components would be easily extendable.

Thus, during the initial design phase we were focussed on:

- Developing a high-resolution OpenVMS performance data collector
- Designing a generic data access model and, based on that, developing a common query interface

This section describes the major challenges we faced during the development of these two core components of HP PERFDAT and provides some insight into their design. Features have been added to both components over time but the base design remains unchanged.

HP PERFDAT OpenVMS Performance Data Collector

During the design phase we found that there were additional requirements to those already stated:

- High-resolution data collector that provides an unprecedented level of details.
- Completeness of data.
- Low system resource consumption.

These requirements are:

- The ability to handle several performance data collections in parallel:

  This requirement derives from best practice considerations on how to use HP PERFDAT during the design phase. In most cases, a single performance data collection will be active on a system. Typically, this base collection will not be configured to collect all performance data that can be provided by the data collector but only a subset. Under normal condition this would be sufficient to perform all required performance management tasks and, in most cases, the performance data provided will also be sufficient for root-cause performance analysis as well. Sometimes, however, it may be the case that more detailed information has to be gathered for a defined period of time due to performance problems or due to just testing new application software. If the data collector were not able to handle several performance collections in parallel, the base collection would have to be stopped. As no other data collection is active during normal operations except the base collection, it will be the source for all automated performance management related tasks. If the base collection has to be stopped, data will be missing in its data file exactly during a critical time period where performance problems were encountered or where software tests were performed. Thus, this missing data would have to be fetched from a different source for long-term analysis. This cannot be handled without manual intervention.

- No sample time drift when collecting performance data:

  As long as you analyze performance data from a dedicated data collection for a short period of time it does not matter if the sample time drifts slowly over time. If you want to compare this data to performance data from another period of time or to data collected on other nodes with the same sample interval but a different time drift, you may get into trouble comparing them and as a result reach no, or wrong conclusions. Thus, archiving zero sample time drift was a prerequisite to us (if the selected sample interval is 60 seconds a performance data sample has to be taken exactly each 60 seconds and not 60.6 or 61.2 seconds).

- Performance data has to be collected simultaneously from all OpenVMS sub-systems:

  If the time spent to gather all performance data from all OpenVMS subsystems is not negligible compared to the sample interval of the data collection (e.g., 0.5 seconds elapsed time to gather the performance data of a 2-second sample interval) you may get into trouble trying to analyze the data since the system state may have changed during the collection period.

- Files are referenced by their file ID and thus file IDs are available from OpenVMS data structures but no file names. To analyze I/O performance data of files one would rather request the file name than a file ID. To provide file name information the data collector has to maintain its own file name cache.

- Manageability.

Performance data can be collected by using the MONITOR utility or by the use of system services such as $GETRMI, $GETSYI, $GETJPI, $GETSPI (undocumented) and so on. The MONITOR utility as well as system services provide detailed performance information for particular subsystems. The problem is that this is not the case for all OpenVMS subsystems such as XFC, LAN, and network protocols and no interface exists that provides device I/O performance information on a per-file and/or per-process basis. One has the option to use the MONITOR utility and wrap some additional tools around it as T4 does, but in this case the requirements of:

- No sample time drift
- Simultaneous data collection
- Manageability
- Running several collections in parallel

can hardly be fulfilled. Thus, we decided to develop a single-process data collector and not a set of collector tools.

At first glance, extensive use of system services to collect performance data seems to be a good idea. System services are stable and are easy to use. On the other hand, the HP PERFDAT OpenVMS data collector has to provide highly detailed performance information but, at the same time, consume as few system resources as possible. Each system service call causes additional overhead compared to fetching the same information directly from OpenVMS data structures. Thousands of system service calls may be necessary whenever a performance data sample is taken. For example, if you want to collect process performance information for all processes active on a huge system with 4000 concurrent processes, 4000 $GETJPI calls are required. This increases CPU load and the elapsed time to collect performance data. In this case, system services are used significantly compared to fetching the same performance data directly from the OpenVMS data structures.

This was the main reason we decided not to use system services but to fetch performance data directly from the data structures. In addition (almost) simultaneous data collection is more likely to occur compared to using system services since the run-time to collect data is much shorter. Most of the data fetches are performed at elevated IPL to prevent being re-scheduled during that period of time.

Not all performance data can be collected by actively gathering data from the data structures of OpenVMS. Device I/O performance data can be collected by using the call-back mechanism provided by the OpenVMS PMS subsystem. This feature tracks information about each I/O performed on the system. Special data structures and buffers have to be provided to OpenVMS to trigger the call-back mechanism. Once the call-back mechanism is enabled OpenVMS inserts information about each I/O performed on the system into these buffers. Up to 4 records are inserted per I/O, each at a different stage of the I/O flow. All information is provided to break down the I/O to devices, files, and process. Once a memory buffer is full, the collecting process is triggered to read and evaluate the entries in that buffer and OpenVMS switches to the next free buffer, if available, and proceeds to write I/O information. If OpenVMS finds no free buffer, the I/O information is dismissed. Thus, it is important that the calling process starts reading and evaluating the I/O information provided in the buffers immediately whenever it is triggered in order to guarantee that at least one free buffer is available to OpenVMS.

If the collecting process were single threaded, evaluating the I/O information buffers could be delayed due to other activities of the process, such as gathering other performance data or writing the data to the file.

The HP PERFDAT OpenVMS data collector was designed from the very beginning as a multi-threaded process. It consists of seven detached threads (see Figure 1):

- Active data sampling thread polling raw performance data but I/O information periodically.
- Reactive I/O sampling thread that exclusively evaluates the I/O information provided by the OpenVMS PMS sub-system.
- Three data collection worker threads processing the raw data provided by the active and reactive data sampling thread and writing this data to a file.
- Management thread that provides information to a management utility via mailboxes. It provides information about the actual status of all active collections, triggers collection startups, coordinates all other threads when a collection is requested to be stopped and is responsible for synchronized thread termination when shutting down the whole performance data collection process.
- File name resolving thread.

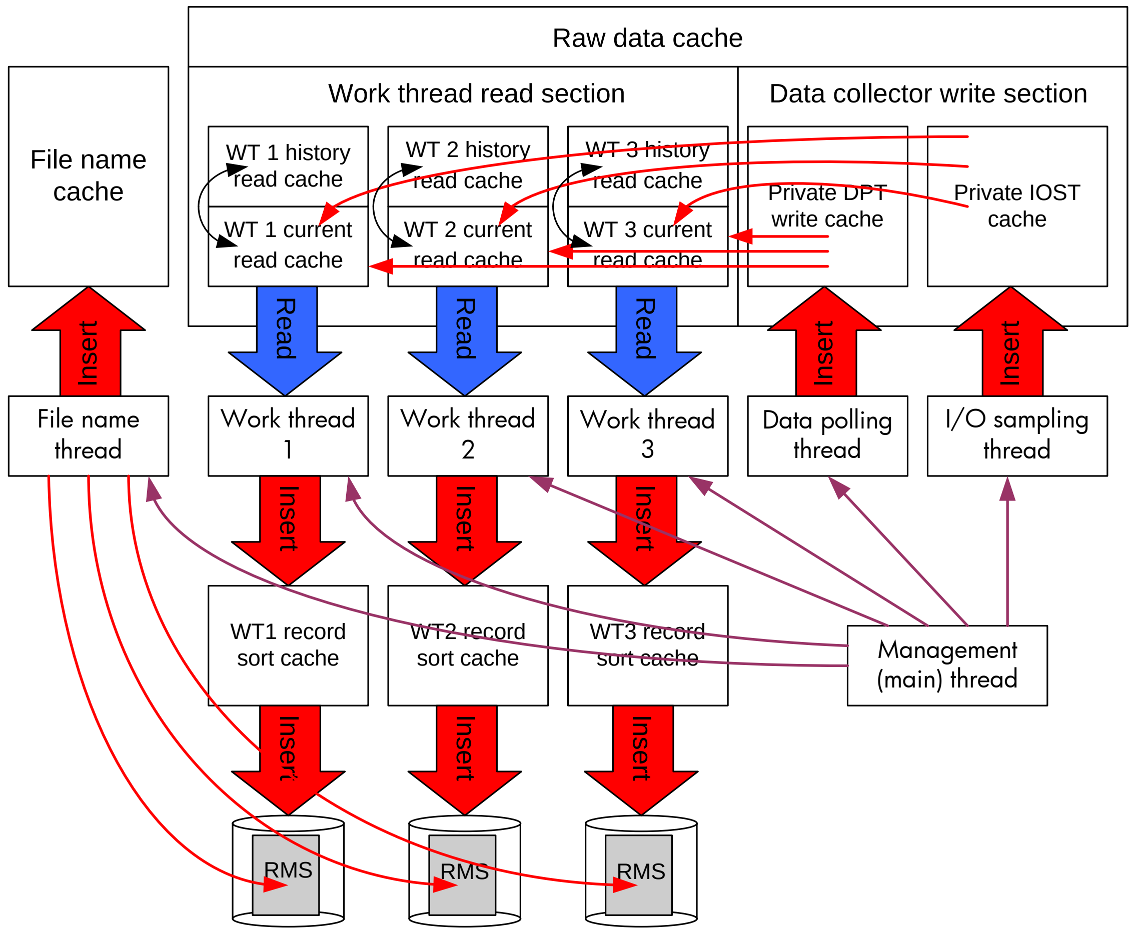


Figure 1 - HP PERFDAT OpenVMS data collector thread and cache layout

If a data collector process handles more than one data collection, you have the option to gather all required performance raw data separately for each collection. This is not efficient, however, since it is very likely that data sampling for all active performance data collections are periodically triggered at the same time. This means that the same kind of data is requested by each data collection and data is gathered unnecessarily. It is more efficient to have the raw data sampling threads separated from the "worker" threads as we call them. The sampling threads are triggered whenever the sample interval of an active data collection expires. If the sample interval of different collections expires at the same time, all performance raw data requested by each performance data collection is collected just once and inserted into the global raw data cache for post-processing.

With this design, the number of collections that can be active in parallel is just limited by the number of worker threads started. The current version of the performance data collector starts three worker threads. Thus, up to three data collections can be active in parallel.

The raw data cache is one of the most important components of the OpenVMS data collector and it is the key to the high efficiency of the collection process. To avoid synchronization delays when accessing data in the raw performance data cache, the cache is separated into a read and a write section. The read section consists of thread-specific regions each exclusively accessed by each of the worker threads. The write section consists of a region that is exclusively accessed by the active data sampling thread and one that is exclusively accessed by the reactive-I/O sampling thread. This means that thread run-time delays due to raw data cache synchronization can be avoided.

In most cases, data gathered from OpenVMS data structures are absolute counter values. When analyzing performance, one is interested in the change of these counters rather in their absolute values. Consequently, the raw data cache has to maintain not only raw data actually collected but also the raw performance data of the last sample interval for each data collection. Thus, each worker thread region in the read section is divided into a current region containing the most recent data collected, and a historical region that contains performance raw data of the last sample interval collection.

Once data is requested from a worker thread due to sample interval expiration the active data sampling thread is triggered. It fetches data from the OpenVMS data structures and inserts it into its write-cache section region. In addition, it triggers the reactive-I/O sampling thread to dump the I/O data collected into its private region of the write-cache section. As both collection threads have signaled completion of their operation the content of both write regions is moved into the current regions of the read section of the calling threads, which are then triggered for post-processing. If the current region of a worker-thread read cache is not clear because the worker thread is still busy processing data of the last sample interval, then it is not replaced by the content of the write cache section and the worker thread is not triggered for post-processing. In this case, performance data will be lost for the actual sample interval in the data collection processed by that worker thread. All the signaling is done by event flags. Once a worker thread has completed all of its tasks, it clears the historical region and swaps its current and historical region.

It is obvious that the delay in addressing raw cache entries is critical to the overall performance of the OpenVMS data collector. A mixture of highly optimized hashing and block-cache algorithms is used to guarantee that all cache entries can be directly addressed or addressed via a minimum of hops. Optimizing all of the caching algorithms so that the data collector -- even on huge, heavily loaded systems -- provides in-depth, high-resolution performance information without harming system performance was the real challenge (For example, one of our customers uses PERFDAT on a CPU and I/O test system with approximately 3,000 concurrent processes. With 2 GB of performance data collected per day, the CPU load caused by the data collector is typically less than 6% of a single CPU.)

The worker threads are responsible for processing raw performance data and converting it into valuable data for performance analysis while making the data persistent by writing it to a dedicated file. Performance data files are stored in indexed sequential files. Each worker thread has to process and store all performance data within its sample time. Insert speed to the data files is also critical to the usability of the data collector. For this reason, performance data records are not written directly to RMS when they are created, but are cached in advance, ordered by their primary key. Only then are these records inserted in order to guarantee optimal RMS performance.

The file-name-resolving thread is responsible for file-ID-to-file-name resolution. As stated previously, OpenVMS provides file IDs when collecting file-related I/O performance counters. To resolve file IDs to file names synchronously is not an option since this would slow down the data insertion to such an extent that the whole data collector would become unusable. Thus, file-ID-to-file-name resolution is out-tasked to the file-name-resolving thread. Whenever a file-related data record is written to a file, the file ID is passed to the file-name-resolving thread. If the file name related to the ID received does not exist in the file name cache, it passes the file name asynchronously to all other threads using the LIB$FID_TO_NAME run-time routine and adds it to its cache. Once the dedicated worker thread has inserted all its records to the data file, the file-name-resolving thread starts updating the file-name reference region in the appropriate data file of the worker thread. File names that refer file IDs that have been received from a worker thread and that are not marked as inserted into the worker thread's data file will be written. File IDs that have not been resolved up to that time will be inserted to the data file as soon they are available. It is guaranteed that the file-name-resolving thread blocks all data file inserts when the owner worker thread regains activity (the priority of performance-data inserts is higher than file-name inserts).

The higher the hit rate of the file-name cache when collecting file-related performance information, the less I/O load is caused by the data collector during the data collection phase. Thus, when the data collector is started all files currently known to OpenVMS are pre-cached before any performance data collection is triggered.

Generic Data Access Model and Common Query Interface.

The second core requirement of the HP PERFDAT solution for OpenVMS was the development of a generic data access model and a common query interface. This would guarantee that data from any source is provided to any component in a standardized format. Thus, with this approach we would achieve a high level of reusability of all performance processing services. In addition, we would avoid any data access problems for all performance processing services caused by version incompatibilities. Under the envisioned design, any data can be accessed transparently regardless of which tool created the file or if, for example, the record format, size, or content changed from one HP PERFDAT release to the next. Data will stay accessible to any utility that accesses performance data via such an interface.

A common query interface that provides generic data access to any kind of data file has to contain a data abstraction layer that reads the records from the files in native format and converts them to a standardized format. This is obvious. The fundamental question, however, is which component of the overall HP PERFDAT environment should maintain the knowledge of the native record structure. Knowledge of the native record structure is the key for any data conversion.

One option is to bundle all required adapters, which, in fact, are software components, with the data abstraction layer. Typically, one would provide one adapter per data file type. The more data files of different types (different layout and record structures) that exist, the more adapters that have to be available. That is not the key point, however, and we were of the opinion that such an approach was not sufficient.

In principle, an approach such as this just displaces the version-dependency problems one would face if no data abstraction layer existed from the performance-processing services into the data abstraction layer. If the record format of a source data file changes, the appropriate adapter has to be changed too. In this case, data files containing the new data structure are accessible, but you will not be able to read data in a previous data format. In that case, the data abstraction layer has to provide both the old and the new adapter. Thus, for any change of data and/or file format new adapters have to be provided without removing the old ones. The number of adapter versions will continuously increase over time and we would end up with a hard-to-maintain data abstraction layer.

The other option is that the data abstraction layer of a common query interface does not have explicit knowledge of the record and file structure, but that the information is placed directly in the source file. In fact, nothing except the source that created the data files knows the data and record structure better. Thus, we adhere to the principle of letting the source tell all other components how to access the data.

All meta-data (field and record descriptors, data link descriptors, index reference table descriptor s, and so on) required to access the data is stored in the header of each data file. The first record in the file is the link descriptor that points to where to find the descriptors in the file. Those are required for reading specific data from that file. This is the only structure that has to be common to the data abstraction layer and the sources of the data files. As long as the structure of the link descriptor does not change, the data abstraction layer can access data of any kind. Thus, record structures and data formats of the data file can be changed at any time without having any impact on the accessibility of data via the data abstraction layer.

Providing such a data abstraction layer fulfills all the requirements that are related to up- and backward data compatibility, but it does not fulfill all the requirements related to single-point access.

Data files are created periodically on each node within your environment running HP PERFDAT as shown in Figure 3. Transparent single-point data access has the following meaning to us. It is the ability to access all the data files from one single node regardless on which node the data files are located. This is exactly the challenge we faced when we talked about transparent single-point data access – where are the data files located?

Databases like Oracle Rdb maintain a root file that refers to the storage areas of the database. To create such a persistent root file for the sum of all data files (we call it the HP PERFDAT distributed performance database) within your environment is not possible since all nodes that run HP PERFDAT -- even those that are not members of a cluster -- would need access to a common disk. The only thing possible is to create and maintain a root file on each node that refers to the files stored locally. That root file contains references only to the data files that are locally accessible which is, in most cases, a small part of the whole HP PERFDAT distributed database.

Our approach to that challenge was to design a network abstraction layer that services data queries similar to the way the data abstraction layer services data file content. The basic functionality of the network abstraction layer is comparable to routers.

When the networking abstraction layer starts up on a node (let's call this node the access server) it broadcasts a root file query to all nodes within the HP PERFDAT community. All remote nodes that have data files stored locally return their local root file information. The access server caches the data file information received and creates a virtual root file (non persistent – memory resident) of the whole HP PERFDAT distributed performance database (you can also call this a routing table).

Both the data and the network abstraction layer were the keys to fulfilling all the requirements related to data access as described in the pervious section. These two layers represent the kernel of the common query interface that is used by all HP PERFDAT data processing services. Thus, all HP PERFDAT data processing services can transparently access any data independent of the source data format. The data file may be stored locally or on a server located hundreds of miles away – it makes no difference to the HP PERFDAT data processing services.

The common query interface has been extended over time, but the design of these two layers is still valid and unchanged. A brief description of all components that comprise the common query interface of the current HP PERFDAT version is provided in the next section.

HP PERFDAT Architecture

This section provides an overview of the architecture, the components, and features available of the current HP PERFDAT version V3.2. HP PERFDAT V3.2 is supported on:

- OpenVMS V7.2-2 Alpha.
- OpenVMS V7.3 Alpha.
- OpenVMS V7.3-1 Alpha.
- OpenVMS V7.3-2 Alpha.
- OpenVMS V8.2 Alpha.
- OpenVMS V8.2 for Integrity servers.
- OpenVMS V8.2-1 for Integrity servers.

It consists of the following software components:

- HP PERFDAT OpenVMS Data Collector (up to 3 collections can be run simultaneously).
- HP PERFDAT SNMP extension – with the SNMP extension, any single OpenVMS node can collect performance data from up to 64 non-OpenVMS systems providing performance data via SNMP.
- Distributed performance database.
- PERFDAT configuration database.
- PERFDAT cluster view database.
- Data Query Interface/Data Query Language (DQL).
- Performance database file name cache service.
- Online performance alerting.
- Statistics package.
- HP PERFDAT Auto-archiving service and housekeeping.
- HP PERFDAT Auto-trend engine.
- Graphical User Interface.
- Management Interface (PERFDAT_MGR).
- Tools and utilities.

Figure 2 shows the overall software architecture of the HP PERFDAT performance solution for OpenVMS.

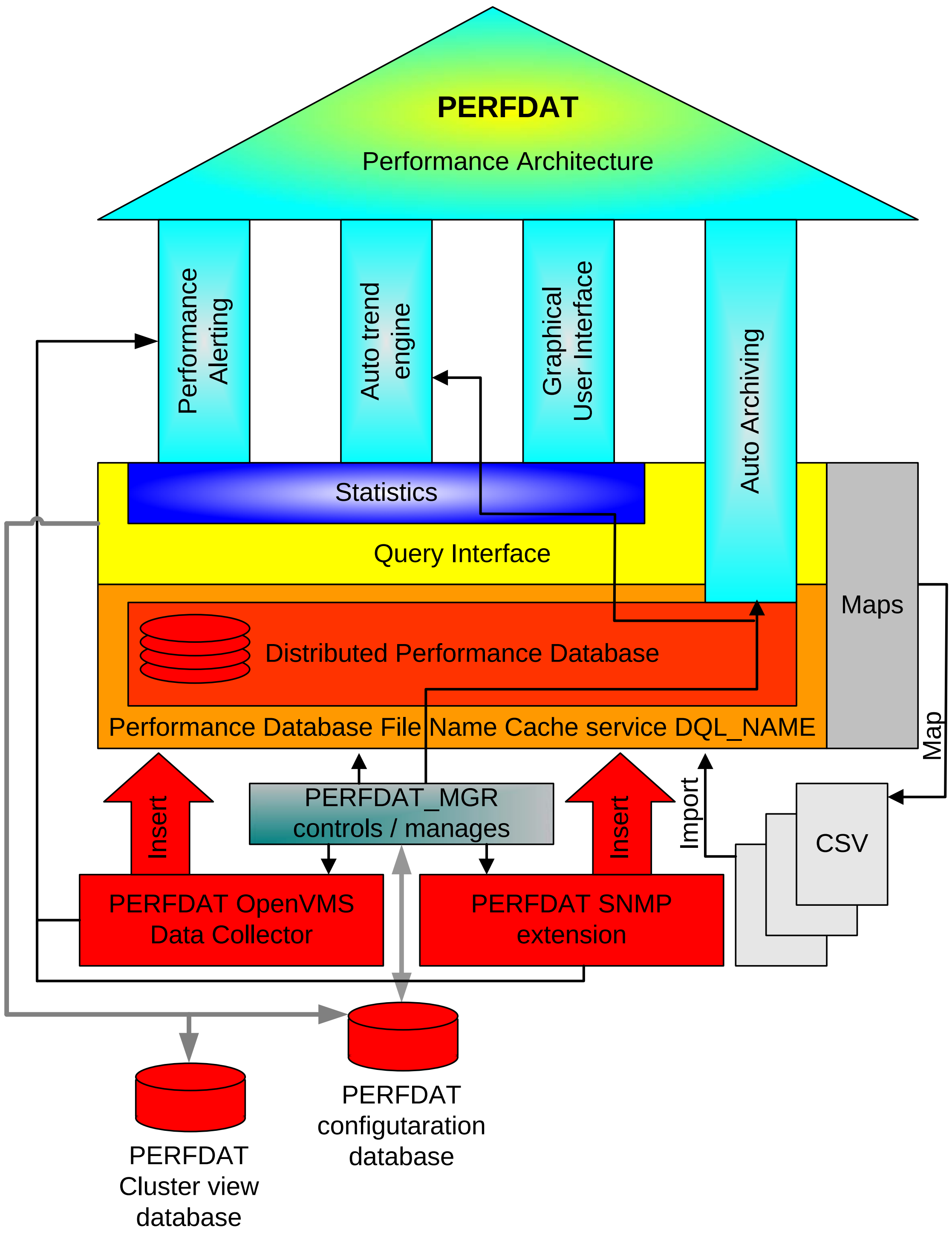

PERFDAT
Performance Architecture
Performance Alerting
Auto trend engine
Graphical User Interface
Auto Archiving
Statistics
Query Interface
Maps
Distributed Performance Database
Performance Database File Name Cache service DQL_NAME
Map
Insert
PERFDAT_MGR controls / manages
Insert
Import
CSV
PERFDAT OpenVMS Data Collector
PERFDAT SNMP extension
PERFDAT configutaration database
PERFDAT Cluster view database

Figure 2 - HP PERFDAT software architecture

Although HP PERFDAT consists of several components it is truly plug and play. It is automatically configured during the installation process. The default configurations have been developed with OpenVMS customers and will fulfill customer needs in most cases. Once HP PERFDAT is installed, it is automatically started. All the important performance management activities such as creating trend reports and data archiving to maintain a performance history of your system are performed automatically and without any need for system management intervention. The only important post-installation activity is to add the HP PERFDAT startup and shutdown procedures to the site-specific OpenVMS startup and shutdown routine to guarantee that HP PERFDAT is automatically started when the system is (re)booted.

HP PERFDAT OpenVMS Data Collector

The prerequisite for effective performance management is to gather accurate and complete performance information of any OpenVMS subsystem. If performance measurements are inaccurate or incomplete, it is very likely that the root-cause cannot be evaluated when analyzing a performance incidence.

HP PERFDAT provides today's most comprehensive OpenVMS data collector on the market. It gathers performance information of any OpenVMS subsystem with an unprecedented level of detail. It provides, for instance, full XFC support down to I/O-size statistics per file and LAN and network protocol statistics. It also gives you the ability to breakdown device I/Os to process and file level.

The main features of the HP PERFDAT OpenVMS data collector include:

- With the multi-threaded HP PERFDAT OpenVMS data collector up to 3 performance data collections can be performed in parallel – each with a different level of detail and metrics enabled.
- Performance data collections are profile controlled.
- These profiles can be defined offline before starting a collection.
- An auto-start collection profile can be defined to auto-start a performance collection whenever the HP PERFDAT OpenVMS data collector is (re)started.
- Performance data collections can be started and stopped automatically, manually, or scheduled to run for a predefined period of time.
- Performance data collections and their profiles are managed and controlled by the common management utility PERFDAT_MGR.
- It collects more than 660 statistics organized in 23 metrics.
- The sample interval for gathering performance information of the OpenVMS sub-systems is freely definable (minimum = 1 sec)
- Each of the metrics can be enabled/disabled independently.
- Performance data collected can be restricted to single/multiple devices, processes, users, images, and volumes.
- I/O performance data (device metric) can be collected with unprecedented resolution. I/O performance data is not only collected per device, but also, for easy root-cause analysis, it can be collected per process and device; per file and device; and per process, file, and device on demand. Thus, hot file statistics as well as the perpetrator of hot files can be identified.
- I/O performance data can be collected for any type of class device (the device metric is not restricted to disk, tape, or mailbox devices).
- Files in the device and XFC metric are not only resolved to file IDs but also to their actual file names.
- Complete XFC integration.
- Full LAN and network protocol support.
- Dynamic resource trimming:

  The HP PERFDAT OpenVMS data collector was designed to use the least amount of system resources as possible since the task of a performance data collector is to provide all the data required to analyze and resolve performance problems -- not to cause them. Nevertheless, if data collections are started with the lowest sample interval possible without any device filters and all device and XFC options enabled, the OpenVMS data collector may become one of the top resource-consuming processes on the system. In order to avoid performance problems due to running HP PERFDAT, the OpenVMS data collector watches its own resource consumption, and if CPU load and/or I/O load exceeds definable thresholds, HP PERFDAT automatically increases the collection sample intervals and/or dismisses metrics according to internal rules.

- New performance data files are created daily for each active collection or whenever a performance data collection is (re)started. The daily data flush time can be freely defined.
- Permits online monitoring.

HP PERFDAT SNMP Extension

The data from the HP PERFDAT OpenVMS data collector provides an unprecedented level of insight into system performance. However, this sometimes may not be sufficient for root-cause analysis, especially if your OpenVMS systems are attached to components that are not exclusively accessed by the OpenVMS systems like a SAN. In this case, the activity of other non-OpenVMS system may have a direct impact on the performance (especially I/O performance) of your OpenVMS systems. Thus, HP PERFDAT provides a SNMP extension to monitor any non-OpenVMS system that provides performance data via SNMP. Up to 64 remote systems can be monitored per OpenVMS node.

The HP PERFDAT SNMP extension support Brocade switches and Tru64 UNIX systems out of the box. If you want to monitor any other system that provides performance information via SNMP, you can customize the appropriate configuration tables.

The main features of the HP PERFDAT SNMP extension at a glance:

- Up to 64 remote nodes can be monitored in parallel.
- Performance data collections are profile controlled and these profiles can be defined offline before starting a collection.
- An auto-start collection profile can be defined to auto-start a performance collection for each remote system that shall be monitored whenever the HP PERFDAT SNMP extension is (re)started.
- Performance data collections can be started and stopped automatically, manually, or scheduled to run for a predefined period of time.
- Performance data collections of the HP PERFDAT SNMP extension and their profiles are managed and controlled by the common management utility PERFDAT_MGR.
- Metrics and statistics are predefined for Tru64 UNIX systems and Brocade switches.
- Sample interval is freely definable (minimum = 1 minute).
- Each metric can be enabled/disabled independently.
- Permits online monitoring.
- Online performance alerting can be enabled dynamically.
- New performance data files are created daily for each active SNMP collection or whenever a performance data collection is (re)started. The daily data flush time can be freely defined.

Distributed Performance Database

All data collected by the HP PERFDAT OpenVMS data collector and the HP PERFDAT SNMP extension is stored in index-sequential RMS files. As described in the previous sections, each data collector creates a new file daily or whenever a collection is started (or restarted). Thus, 1 to n data files can exist per day and collection. A single data file is called a physical storage area. All physical storage areas that are created on the same day and that belong to the same collection (collection profile and node) are called a logical storage area. Figure 3 presents a graphical overview of the database organization of the HP PERFDAT distributed database.

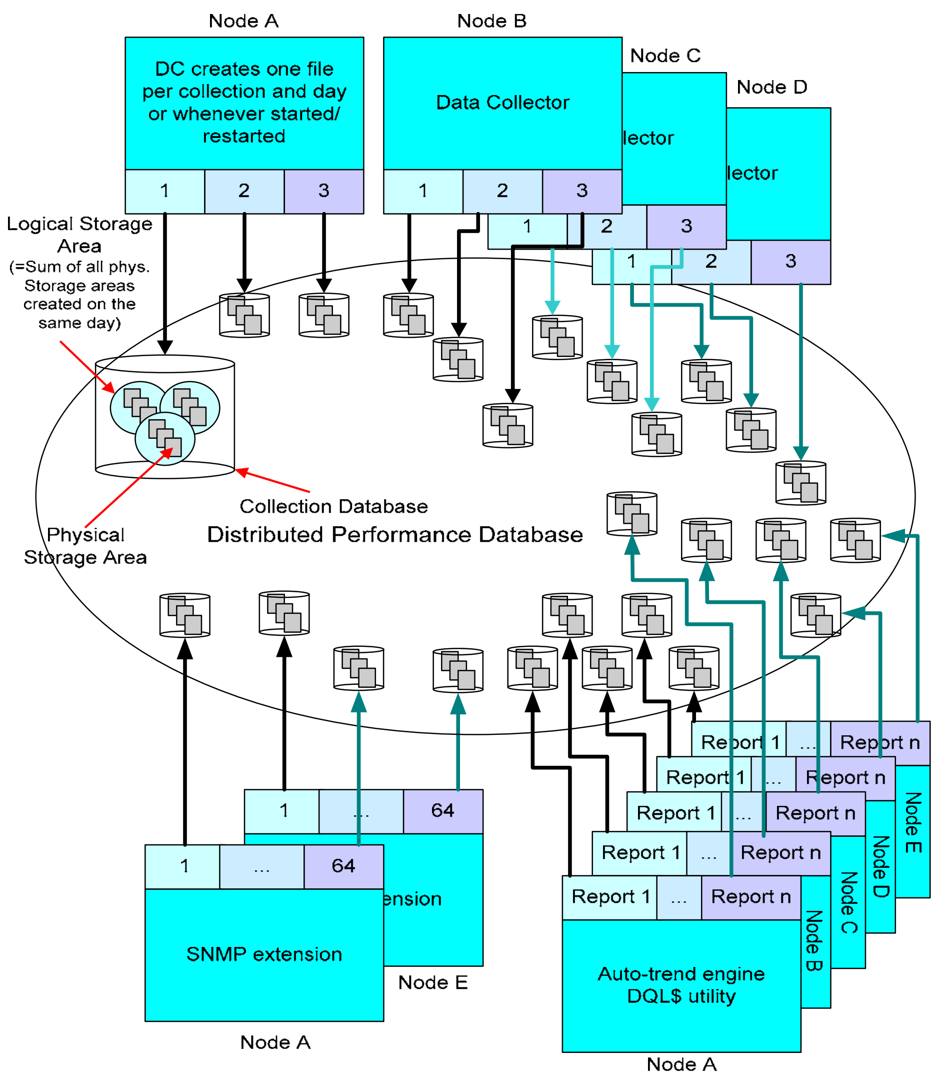


Figure 3 - Database organization of the HP PERFDAT distributed database

In addition to the HP PERFDAT OpenVMS data collector and the HP PERFDAT SNMP extension, performance data files are created by the auto-trend engine responsible for providing automated trend and capacity reports. Trend and capacity report data files contain data of a particular time period – called a report period (day, week, month, quarter, year) – defined by the report profile used to create these reports. At the end of such a predefined time period -- or whenever the definitions of the report profile change -- a new report data file is created. As with the data collector, a single report data file is also called a physical storage area. The sum of all physical storage areas that are created during a report period is called a logical storage. If the report period is WEEK, for example, all report data files created during a week make up the logical storage area for this report.

All logical storage areas created by the same data collection or the auto-trend engine using the same report profile make up a collection database. The sum of all collection databases available within your environment is called the HP PERFDAT distributed performance database (Fig. 3).

The data files of the distributed performance database can be stored on any node within your environment. HP PERFDAT distributed database data is accessed via the common DQL interface.

HP PERFDAT DQL Interface

HP PERFDAT DQL (Data Query Language) interface provides a common interface for transparent access to the distributed performance database. DQL is similar to SQL. All basic query statements such as SELECT, INSERT, CREATE, and DROP are supported except UPDATE and DELETE to prevent after-image data manipulation.

DQL provides single-point access to all HP PERFDAT performance data files regardless of where the data files are stored within your environment. Even if data files are literally spread all over the world there is no need for any manual data transfer or preprocessing to access and analyze performance data. The relocation or renaming of performance data files has no effect on the accessibility of the data.

All meta-data (field and record descriptors, data link descriptors, index reference table descriptor, and so on) necessary to access performance data in a data file is stored in the header of each physical storage area. Due to the fact that the DQL interface needs no implicit knowledge about the internal structure of the data files, there exists no version dependency when accessing performance data. Performance data accessed via the DQL interface is always readable independent of the OpenVMS and/or HP PERFDAT version with which the performance data was collected; the OpenVMS and/or HP PERFDAT version of the system where the performance data resides; or the version of the HP PERFDAT GUI used for data analysis.

Additional features of the DQL interface are as listed below:

- Ability to map/import CSV formatted performance data from additional data sources (for example, databases like Oracle Rdb, applications, and storage controllers) to guarantee collaboration with other performance data collection utilities. If a CSV file is mapped, the content of the CSV file can be accessed via the DQL interface as if it were part of the distributed performance database. In contrast to mapping a CSV data file, if you import the content of a CSV file to the distributed performance database it becomes part of the distributed performance data and is automatically handled by the HP PERFDAT data archiving service. Mapped CSV files are not handled by the HP PERFDAT archiving service. In addition, only a limited number of statistical methods and features to analyze performance data can be applied to mapped CSV content.
- Advanced performance data export capability to CSV files to guarantee collaboration with existing performance analysis utilities (e.g., TLViz) and charting tools (e.g., Excel). The format of a CSV export file (date/time format, list separator, and decimal symbol) is freely definable to avoid re-formatting the CSV export file before it can be used as input for a dedicated utility (e.g., Excel – CSV input format accepted depends on the regional settings).
- Multi file version support.
- Ability to define cluster views.
- User defined statistics:

  User defined statistics are calculated values that can, once defined, be accessed as if they are part of the collection databases. There are several reasons for defining stored procedures. The most common are:

  – You want to normalize data.
  – You are interested in special statistics that are not directly collected by the HP PERFDAT OpenVMS data collector or the HP PERFDAT SNMP extension but all input parameters to compute these are available. For example, the average I/O size of disk devices is not directly collected by the OpenVMS data collector but throughput and I/O requests are. Thus, the average I/O size can be defined as a user-defined statistic:

  Avg. I/O size = Throughput/I/O requests

- Statistics package fully integrated in data query interface. The query interface is not a monolithic layer but consists of six components as shown in Figure 4.
- DQL$SRV (DQL server).
- Cluster view engine.
- Stored procedure engine.
- Statistics Package.
- DQL$ command line utility.
- PDBC$SRV (Performance database connectivity server).

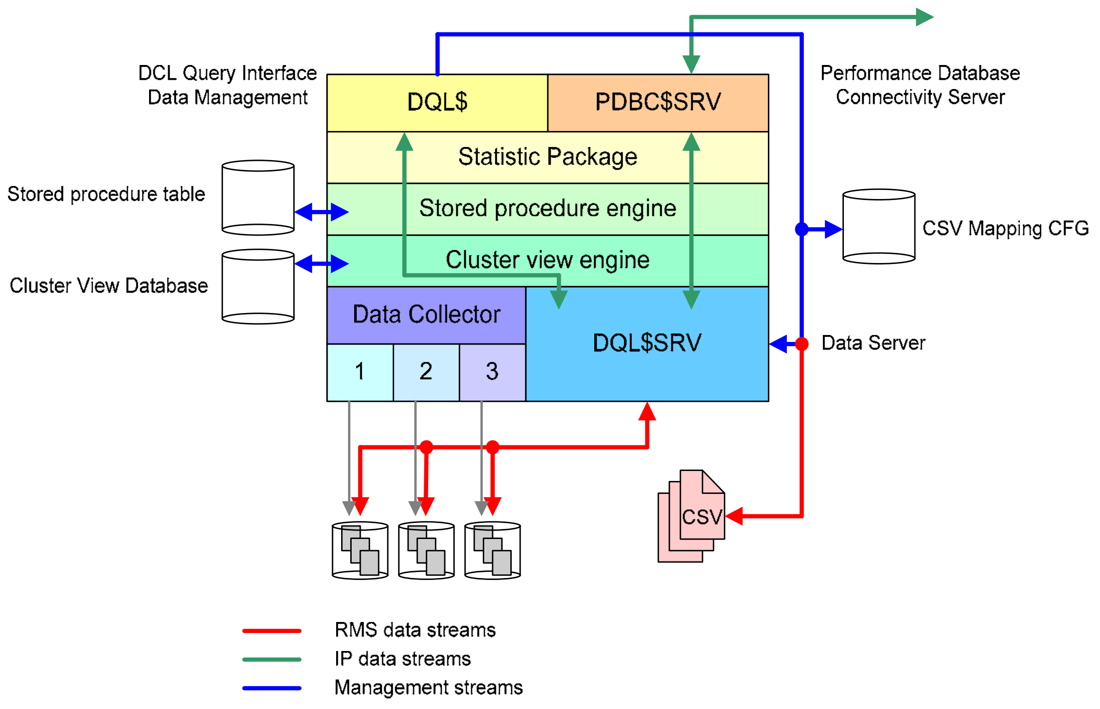


Figure 4 - Components of the DQL interface

DQL$SRV (DQL Server)

The DQL$SRV (DQL server) represents the data abstraction layer of the DQL interface. This component directly accesses the data of the performance data stored locally according to the definitions in the header of the data files. Its main task is to map the data query command received from the cluster view engine to RMS calls. Data that is read from the data files is converted into type-independent format and returned compressed to the calling layer. It handles locally stored data from the collection databases as well as locally mapped CSV files.

The DQL server is implemented as an IP service. The listener port number of this IP service is 3879. Thus, any node within your environment can request data from the DQL$SRV. Up to 99 DQL$SRV connections are allowed per node. Each DQL$SRV process can access up to 2048 data files concurrently.

Cluster View Engine

The cluster view engine provides the feature of mapping performance data from different nodes for cluster-wide performance analysis. Once a cluster view is created, a virtual collection database is accessible that maps the data of the cluster view members. The advantage is that such a virtual cluster view collection database can be accessed as if it were created by the OpenVMS data collector or the SNMP extension. Thus, all methods and features to analyze single-node performance data are available for cluster views, too. Consequently, the workflow to analyze cluster view performance data does not differ from the workflow to analyze single-node performance data.

In most cases, cluster views will be created for cluster-wide performance data analysis of OpenVMS clusters. There are no restrictions, though, that state that only performance collection databases of OpenVMS cluster members can be members of a cluster view. In fact, any collection database of any node available can be added to a cluster view. The only restriction is that all collection databases of a cluster view must be created with the same sample interval.

Any data query is passed to the cluster view engine. If the data query requests cluster view data, appropriate data queries are created for all members (collection databases) of the cluster view. These queries are sent to DQL$SRV. The data streams received from DQL$SRV are merged and the merged data stream is returned to the calling layer. If the data query received contains no cluster view data requests the query is directly bypassed to DQL$SRV.

Stored Procedure Engine

The stored procedure engine enables you to define site-specific measures (statistics). User-defined statistics are calculated values that can, once defined, be accessed as if they are part of the collection databases.

Any data query is passed to the stored procedure engine. If the data query requests user-defined statistics, the data query is modified to request all base statistics necessary to calculate the user-defined statistics. The modified query is passed to the cluster view engine. Once the stored procedure engine receives data from the cluster view engine the user defined statistics are calculated according to the assigned function (procedure) and the result is returned to the caller.

Statistics Package

Any query is passed to the statistics layer. The query is analyzed if it contains a statistics request. If this is the case, appropriate data queries are sent to the stored procedure engine. The data received from the stored procedure engine are decompressed, cached locally, processed according to the statistics request, and the final result is returned to the caller. If the query is a data query, it is sent directly to the stored procedure engine.

PDBC$SRV (Performance Data Connectivity Server)

The performance data connectivity server services data and statistics queries sent by the HP PERFDAT GUI. PDBC$SRV and the DQL$ command line utility represent the network abstraction layer of the DQL interface. Its main tasks are:

- Creating a virtual root file (memory resident) whenever a user connects to the distributed performance database using HP PERFDAT GUI. The PDBC$SRV retains the knowledge of where the data files are located and how to access them.
- Passing the data and statistics queries to the appropriate nodes that host the data files. If the query refers to data files that are stored on different nodes, the performance data connectivity server disassembles the query, forwards appropriate queries to the nodes, consolidates the data received, and returns the result to the caller.
- The performance data connectivity server is implemented in a similar manner to the DQL$SRV - as an IP service listening on port 5245. Up to 99 concurrent PDBC$SRV (PC-client) connections are allowed per node.

DQL$ Command Line Utility

The DQL$ command line utility, like the performance data connectivity server, is responsible for transparent access to the data files within the defined community (network abstraction). The DQL$ utility services interactive DQL requests from the DCL command line interface. In addition the DQL$ utility provides data content capabilities and can be used to extract trend and capacity reports from the distributed performance database manually. The DQL$ utility is scriptable so you can automate data analysis tasks directly on the OpenVMS system.

Performance Database File-name Cache Service

The performance database file-name cache service, DQL_NAME, is a valid cache containing file name and particular file header information of all data files stored in the distributed performance database throughout the whole environment. It provides this file name and file header information to the components of HP PERFDAT that access performance data via the DQL interface.

The file name cache guarantees low database connectivity time when a user initially connects to the distributed performance database even though thousands of data files may be spread over all the systems of your environment.

HP PERFDAT Statistics Package

Effectively managing system performance starts with understanding what is normal and abnormal for your environment.

Based on the historical performance data automatically maintained by HP PERFDAT, the statistics package provides various statistical methods to distinguish between normal and exceptional system behavior. These methods enable you to spot deviations quickly and easily. Deviations may indicate problem areas or just identify opportunities for performance improvements. In addition, the statistical methods available support the efficient filtering of the statistics and parameters that characterize your system most accurately.

The statistics package is a part of the DQL interface. Thus, it is available from the GUI as well as from the command line interface (DQL$).

The advantage of having the statistics package as a server-based implementation is that no massive data transfer between the access server and the PC-client running the GUI is necessary to receive the results. Thus, network load does not increase due to statistic queries. In addition, it is guaranteed that the runtime of a statistic query is (almost) independent of the server location (it does not really matter if the server is located locally or 100 miles away) and the bandwidth of the network.

The statistical functions implemented are:

- Min/max calculations.
- Mean value calculations.
- Standard deviation.
- Correlation.
- Integral and mean value based deviation calculation.

Elements can be ordered by any statistics of the metric. This means that the elements are displayed in ascending or descending order based on the percentage of the overall load defined by the statistics caused by each element. The time range is freely definable.

HP PERFDAT Online Alerting

The online performance alerting subsystem provides real-time alerting capabilities. It supports system management to detect performance anomalies in real time. Because it does not slow down the overall system performance significantly, it is transparent to the end-user. Online performance alerting can be enabled for any active performance data collection independent of whether the data collection is performed by the OpenVMS data collector or the SNMP extension.

Once online alerting has been enabled for an active performance data collection, the alerting subsystem tracks the actual values of specific statistics collected by the OpenVMS data collector and the SNMP extension and triggers alerts if any alert condition is found to be true.

The statistics to monitor the alert conditions and the alert method are defined by alert blocks within an alert definition file. The prerequisite for enabling online alerting for an active performance data collection is that a valid alert definition file exists. An alert definition file is an easily customizable text file.

During HP PERFDAT installation, predefined alert definition files are provided. They can be used as template configuration files to create your own alert definition file or used directly to enable online alerting for appropriate performance data collections.

HP PERFDAT Auto-archiving Service and Housekeeping

HP PERFDAT provides automatic data management capabilities. The HP PERFDAT archiving service carries out the archiving and housekeeping tasks reliably and unattended. All tasks performed by the service on any node within your environment are listed in order below:

- If an archive node (see HP PERFDAT environment section) is defined within your environment, all closed performance data files stored locally are moved to the archive node.
- Data collection file cleanup:
  - All data collection files stored locally with a creation date that has exceeded the *keep-time* are deleted in order to save disk space. Data collection file cleanup only affects performance data collected by the OpenVMS data collector and the SNMP extension.
  - Trend and capacity reports are not processed by the archiving service. Trend and capacity reports are kept as long as the data files are not manually deleted by system management or HP PERFDAT is uninstalled. Thus, it is guaranteed that the performance history of a system is available whenever historical performance data is required for analysis.

  - If you want to prevent performance data files collected by the OpenVMS data collector or by the SNMP extension from being deleted by the archiving service you can move these files manually to the predefined directory PERFDAT$DB_SAVE before the *keep-time* expires. All data files stored in that directory stay accessible to the DQL interface but will not be processed by the HP PERFDAT archiving service.

- Log-file and temp file cleanup
- All software components of the HP PERFDAT environment create log-files when started. All Log-files created by HP PERFDAT components are purged with /KEEP=5. Thus, the last five versions will always be available for examination, if necessary.
- After all data management activities listed above have completed the auto-trend engine is triggered to perform automated trend and capacity reports.
- Calculating the next time to run the archiving and cleanup jobs.

The archiving process performs these tasks once a day. The user can configure the following:

- Enable/disable archive processing.
- Time of day that the archive processing starts.
- Number of days that old performance data files will be kept (*keep-time)*.

HP PERFDAT Auto-trend Engine

One of the most important tasks of performance management is trend evaluation. Trend performance data is valuable for pinpointing an unforeseen performance problem and it is a prerequisite for performance prediction while planning for the future. In order to stay ahead of the curve of how your systems will perform in the future, you have to be aware of historical performance trends.

With the HP PERFDAT automated reporting capability which is easy and flexible to configure, trend and capacity reports are created and continuously updated for any set of statistics and parameters that characterize your systems -- without any necessary user action - for the lifetime of your systems.

Trend report data files created by the auto-trend engine are typically much smaller (1:100) than the performance raw-data files created by the HP PERFDAT OpenVMS data collector or the HP PERFDAT SNMP extension. This is because only a subset of performance data that most characterizes your system is extracted from the raw-data files. These trend data files are not processed by the HP PERFDAT auto-archiving service in order to guarantee the availability of the performance history for the lifetime of the OpenVMS systems. The only way to clean up trend data files is to delete them manually or to uninstall HP PERFDAT.

Four types of reports are configurable:

- Trend report.
- Capacity report.
- Day-to-day deviation report.
- Baseline deviation report

Trend Report

A trend report compresses the selected statistics that most characterize your system. This means that the selected performance data is averaged according to the time compression. The time compression is freely definable, but has to be greater than the sample interval of the collection that created the collection database. Thus, if the time compression is set to 30 min you get 48 values per day and defined statistics in the report profile. The source data has to be collected with a sample interval smaller than 30 min.

Trend reports are helpful if the user is interested in detecting changes in system characteristics over time. If a trend report is created on a weekly basis, trend report data of week 2 and week 25 can easily be compared with the HP PERFDAT GUI. From the graphs created by the GUI you can, for example, directly identify if the level of CPU load on Monday of week 2 is still the same as on Monday of week 25, or if it has changed. You can then easily examine the way it has changed. In addition, you can directly identify if the change in the course of the workload is just limited to a specific day of the week, due to any system problem on a specific day, or if a workload change can be identified on each day of week

Capacity Report

Capacity reports are the basis for capacity planning and forecast analysis. A capacity report contains a daily value per defined statistics. It is very easy, therefore, to identify if the workload on a system increases, decreases, or remains stable over a long period of time. From the performance point of view not all data is of the same interest. It is common to most systems that there are times it is idle and times that it is busy. For capacity planning purposes the busy times are of interest. Thus, up to 5 different time ranges can be defined for a capacity report to cover these busy periods. Only the data within these time ranges is used for calculating the average values.

Day-to-day Deviation Report

The day to day deviation report is derived in a similar manner as the capacity report (averaging the statistics over the time period defined) but does not store the average values. It compares the actual average values to the average values for the previous day and stores the deviation of these average values (%). Such a report directly highlights on a daily basis whether or not there are significant load changes within the time periods of interest.

Baseline Deviation Report

The baseline deviation report performs, in principle, the same calculations as the day-to-day deviation report. It does not, however, compare the actual average values to the previous day's values. Instead, it calculates and stores the deviation between the actual average values and the average values of the same day of the week from the baseline. Baseline data is a set of performance data files collected for a system covering a full week. The baseline represents a typical week where the system performance was considered "normal" based on the user's knowledge and experience. The baseline has to be defined by the system manager by moving the appropriate logical storage areas of a collection database to the predefined directory PERFDAT$DB_SAVE. It does not matter if the data is moved to this directory on the local node or on the archive node (see HP PERFDAT environment section). If the baseline does not cover a full week, only the days of the week are processed that are contained in the baseline. Compared to the other reports, the time period covered by a baseline deviation report file is endless. "Endless" means that the report is extended as long as the baseline data is valid. The system manager can invalidate the baseline data by simply copying a new set of data to the predefined PERFDAT$DB_SAVE directory and deleting the old baseline data. In this case, the next time the baseline report is triggered, the auto-trend engine detects that the baseline data has changed and a new baseline deviation report file is created.

PERFDAT_MGR Management Utility

The OpenVMS command line utility PERFDAT_MGR is the common management interface for the HP PERFDAT components.

The main tasks of PERFDAT_MGR are:

- Startup/shutdown of the HP PERFDAT environment.
- Control/monitor the status of HP PERFDAT OpenVMS performance data collections.
- Control/monitor the status of remote performance data collections via the HP PERFDAT SNMP extension.
- Management/control of the performance data archiving service.
- Management/maintenance of the HP PERFDAT configuration database.
- Online performance alert management.
- Management/control of the performance database file name cache service DQL_NAME.

HP PERFDAT Graphical User Interface

With the state-of-the-art HP PERFDAT Graphical User Interface (GUI) you can access performance data from anywhere on the network. Due to the analysis capabilities provided by the HP PERFDAT GUI, your IT-staff will be able to pinpoint problems and identify their causes without expert knowledge. The HP PERFDAT GUI helps you to focus on the most critical performance information, saving valuable time.

HP PERFDAT GUI supports a consistent analysis methodology for any performance data. Independent of which data and how this data was collected, the HP PERFDAT GUI simplifies the performance management process, reducing the need for training and improving the productivity of your IT-staff.

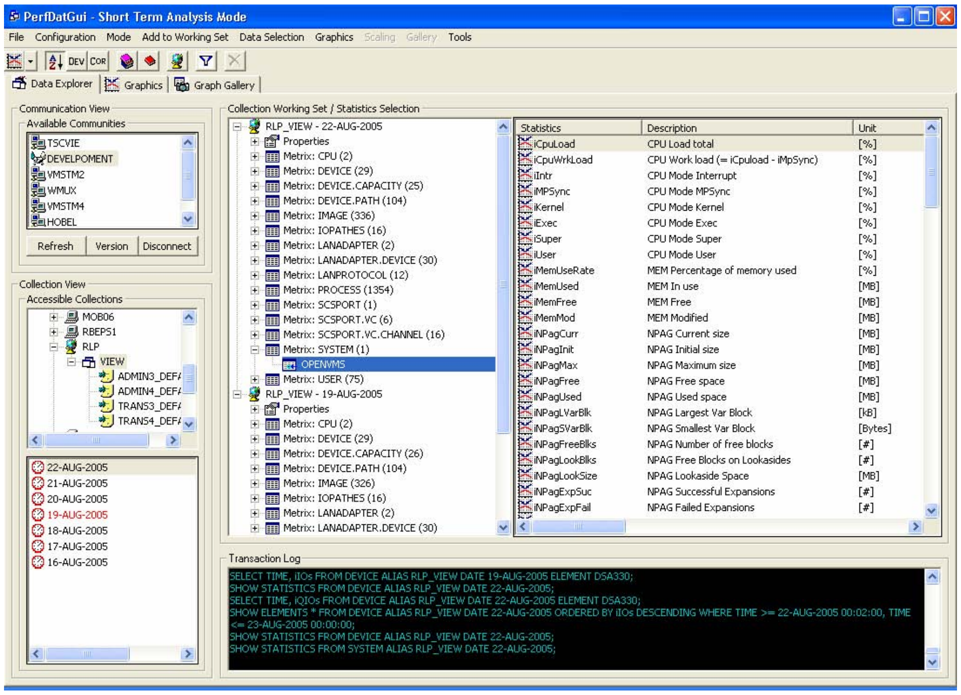
The HP PERFDAT GUI is a Windows tool supported on Windows 2000/2003/XP. After installing the tool, the only thing you have to do before starting performance analysis work is to define an OpenVMS access server that is part of the HP PERFDAT environment (see HP PERFDAT environment section).

The HP PERFDAT GUI accesses performance data via the common DQL interface. No data file is transferred to the PC where the GUI is installed when analyzing data. Only data that is actually selected for graphing or the result of statistical queries is transferred. Since no massive data transfer occurs between the OpenVMS access server and the PC-client running the GUI, it is guaranteed that the network load does not increase significantly due to data analysis.

The main features of the HP PERFDAT GUI are:

- Easy to handle.
- Intuitive.
- Data explorer - easy data navigation (see Figure 5).
- The data explorer of the GUI provides a brief online description and the unit of each of the statistics available (see Figure 5).
- Cluster analysis capability:

  As described in the previous sections, cluster views are provided by the DQL interface. Thus, all methods and features to analyze single-node performance data are available for cluster views too. Consequently, the workflow to analyze cluster view performance data does not differ from the workflow to analyze single-node performance data. Cluster views are only marked with a special icon in order to distinguish between cluster views and performance data collections of single nodes. (All the following HP PERFDAT GUI screen shots were taken during a cluster performance analysis session.)



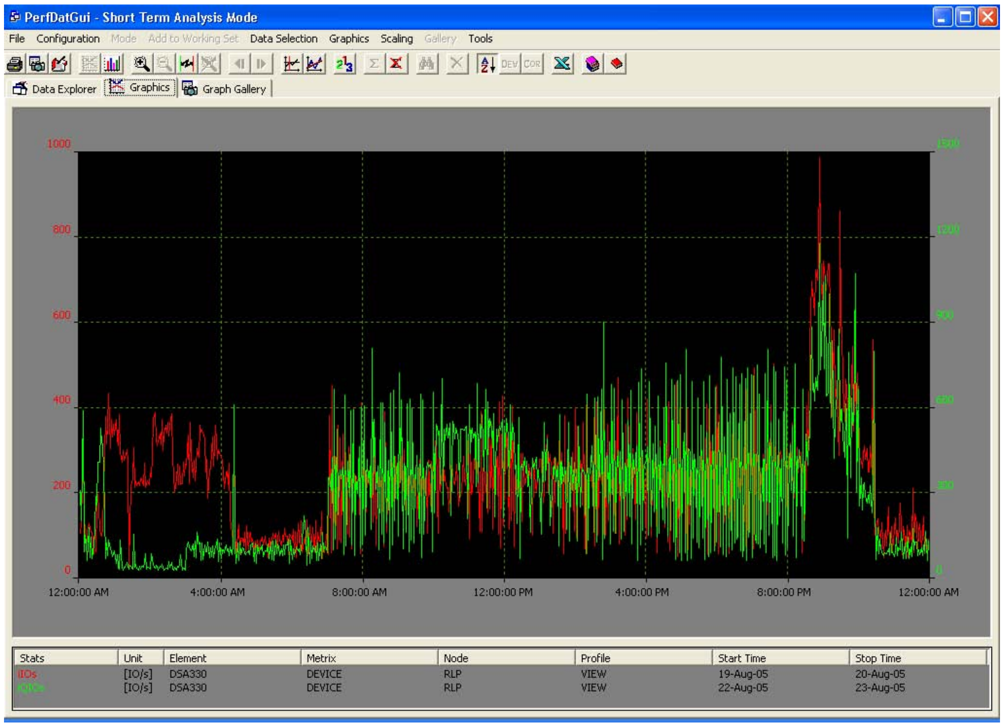
Statistics	Description	Unit
iCpuLoad	CPU Load total	[%]
iCpuWrkLoad	CPU Work load (= iCpuload - iMpSync)	[%]
iIntr	CPU Mode Interrupt	[%]
iMPSync	CPU Mode MPSync	[%]
iKernel	CPU Mode Kernel	[%]
iExec	CPU Mode Exec	[%]
iSuper	CPU Mode Super	[%]
iUser	CPU Mode User	[%]
iMemUseRate	MEM Percentage of memory used	[%]
iMemUsed	MEM In use	[MB]
iMemFree	MEM Free	[MB]
iMemMod	MEM Modified	[MB]
iNPagCurr	NPAG Current size	[MB]
iNPagInit	NPAG Initial size	[MB]
iNPagMax	NPAG Maximum size	[MB]
iNPagFree	NPAG Free space	[MB]
iNPagUsed	NPAG Used space	[MB]
iNPagLVarBlk	NPAG Largest Var Block	[kB]
iNPagSVarBlk	NPAG Smallest Var Block	[Bytes]
iNPagFreeBlks	NPAG Number of free blocks	[#]
iNPagLookBlks	NPAG Free Blocks on Lookasides	[#]
iNPagLookSize	NPAG Lookaside Space	[MB]
iNPagExpSuc	NPAG Successful Expansions	[#]
iNPagExpFail	NPAG Failed Expansions	[#]

```
SELECT TIME, iIOs FROM DEVICE ALIAS RLP_VIEW DATE 19-AUG-2005 ELEMENT DSA330;
SHOW STATISTICS FROM DEVICE ALIAS RLP_VIEW DATE 22-AUG-2005;
SELECT TIME, iQIOs FROM DEVICE ALIAS RLP_VIEW DATE 22-AUG-2005 ELEMENT DSA330;
SHOW ELEMENTS * FROM DEVICE ALIAS RLP_VIEW DATE 22-AUG-2005 ORDERED BY iIOs DESCENDING WHERE TIME >= 22-AUG-2005 00:02:00, TIME
<= 23-AUG-2005 00:00:00;
SHOW STATISTICS FROM DEVICE ALIAS RLP_VIEW DATE 22-AUG-2005;
SHOW STATISTICS FROM SYSTEM ALIAS RLP_VIEW DATE 22-AUG-2005;
```

Figure 5 - HP PERFDAT GUI data explorer for easy data navigation

- State of the art graphical features like:
  - Stack/unstack functionality
  - Zoom in/out
  - Shift left/right
  - Data scanning
  - Ability to scale graph separately
  - Auto, native, and manual scaling capability
  - Data overlay capability (graphs of different time periods can be overlapped to allow visual comparison)

Figure 6 shows an example of the graphical data overlay capability of the HP PERFDAT GUI. The I/O rate caused by the RLP cluster (consisting of 4 members - see also Figure 5) on DSA303 is compared between 19-Jun-2005 and 22-Jun-2005.

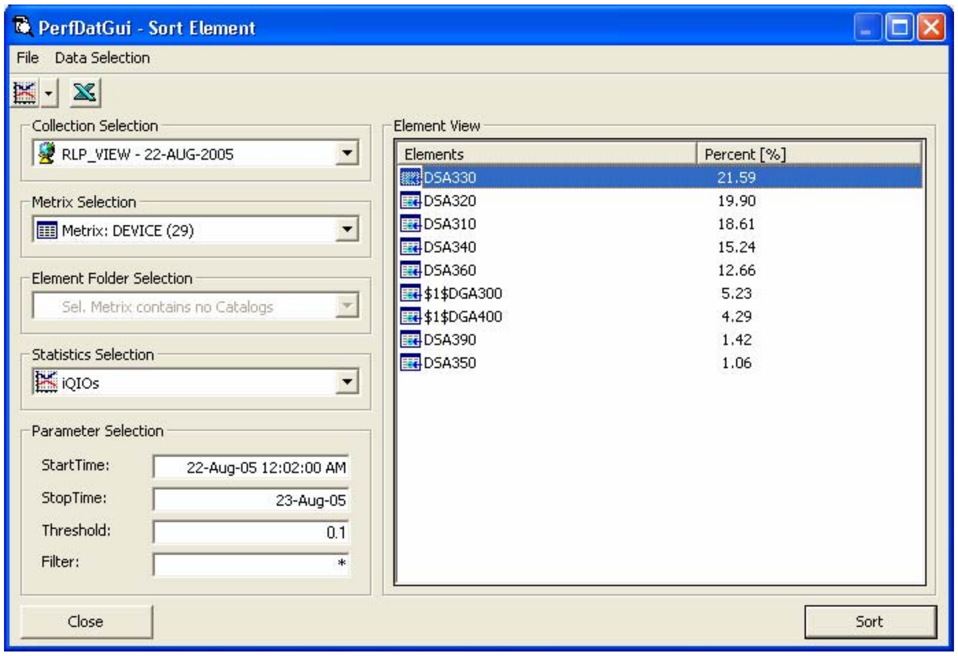


Stats	Unit	Element	Metrix	Node	Profile	Start Time	Stop Time
IIOs	[IO/s]	DSA330	DEVICE	RLP	VIEW	19-Aug-05	20-Aug-05
iQIOs	[IO/s]	DSA330	DEVICE	RLP	VIEW	22-Aug-05	23-Aug-05

Figure 6 - Example of the data overlay capability of the HP PERFDAT GUI

- Data sorting:

  The data-sorting feature can be used to sort elements of a metric by any statistics of that metric. Thus, it is very easy to identify the top resource consuming elements. For example, you can sort devices according to the I/O rate. As a result, the HP PERFDAT GUI displays the devices ordered by their I/O rate and their contribution to the overall system I/O rate in percent.



Collection Selection: RLP_VIEW - 22-AUG-2005

Metrix Selection: Metrix: DEVICE (29)

Element Folder Selection: Sel. Metrix contains no Catalogs

Statistics Selection: iQIOs

Parameter Selection

StartTime: 22-Aug-05 12:02:00 AM

StopTime: 23-Aug-05

Threshold: 0.1

Filter: *

Element View

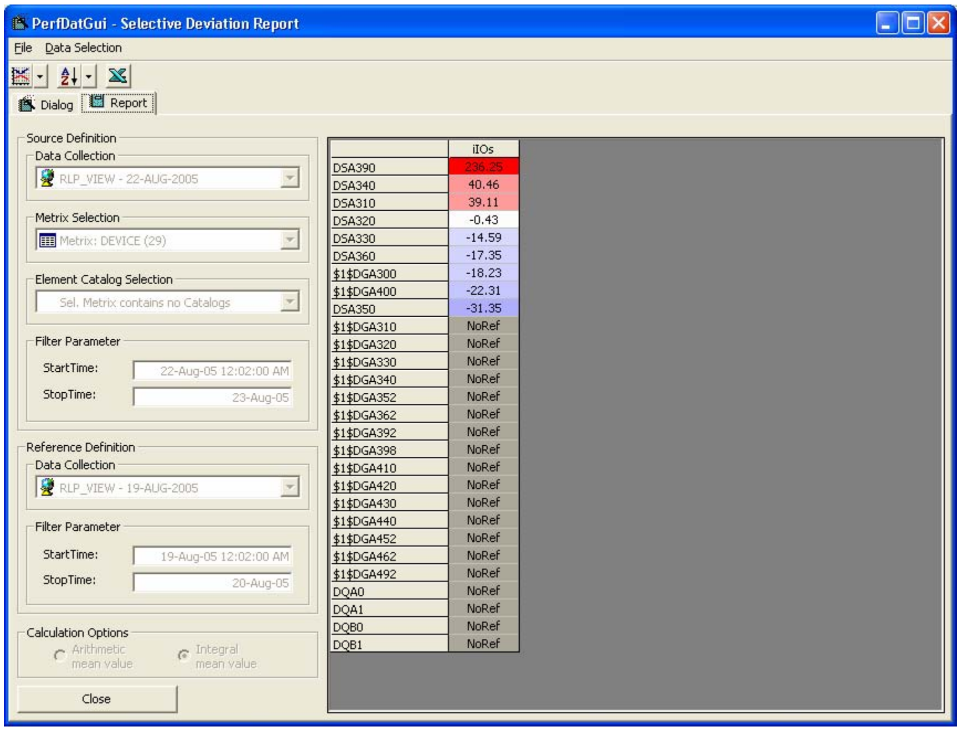
Elements	Percent [%]
DSA330	21.59
DSA320	19.90
DSA310	18.61
DSA340	15.24
DSA360	12.66
$1$DGA300	5.23
$1$DGA400	4.29
DSA390	1.42
DSA350	1.06

Figure 7 - Example of the data sorting capability of the HP PERFDAT GUI

- Deviation analysis capability:

The deviation analysis feature can be applied to detect any change in the workload of the system. This makes it easy to validate the workload shift due to any change (new hardware, new software releases, etc.) to your system or show how the workload evolves over time on an unchanged system.

One of the deviation analysis features is to create a deviation report that is especially valuable for "before and after" performance analysis. Figure 8 show such a deviation analysis report performed for devices. (NoRef means that no data is available for the device on the reference day, in this example 19-Jun-2005.)



Source Definition
Data Collection: RLP_VIEW - 22-AUG-2005
Metrix Selection: Metrix: DEVICE (29)
Element Catalog Selection: Sel. Metrix contains no Catalogs
Filter Parameter
StartTime: 22-Aug-05 12:02:00 AM
StopTime: 23-Aug-05

Reference Definition
Data Collection: RLP_VIEW - 19-AUG-2005
Filter Parameter
StartTime: 19-Aug-05 12:02:00 AM
StopTime: 20-Aug-05

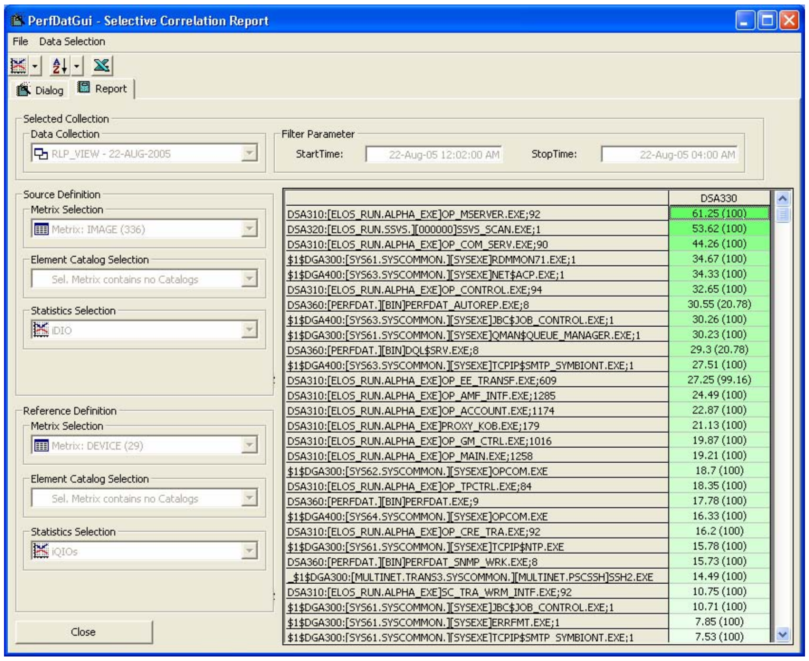
Calculation Options
Arithmetic mean value [✓ Integral mean value]

	iIOs
DSA390	236.25
DSA340	40.46
DSA310	39.11
DSA320	-0.43
DSA330	-14.59
DSA360	-17.35
$1$DGA300	-18.23
$1$DGA400	-22.31
DSA350	-31.35
$1$DGA310	NoRef
$1$DGA320	NoRef
$1$DGA330	NoRef
$1$DGA340	NoRef
$1$DGA352	NoRef
$1$DGA362	NoRef
$1$DGA392	NoRef
$1$DGA398	NoRef
$1$DGA410	NoRef
$1$DGA420	NoRef
$1$DGA430	NoRef
$1$DGA440	NoRef
$1$DGA452	NoRef
$1$DGA462	NoRef
$1$DGA492	NoRef
DQA0	NoRef
DQA1	NoRef
DQB0	NoRef
DQB1	NoRef

Figure 8 - Deviation report

- Correlation analysis capability:

  The data correlation feature can be used for easy dependency analysis. Correlation reports are valuable for system characterization as well as for root-cause performance analysis. Figure 9 shows an example of such a correlation report that illustrates the correlation between the direct I/O rate of all images and the I/O rate of DSA303 sorted in descending order.



Selected Collection
Data Collection: RLP_VIEW - 22-AUG-2005
Filter Parameter
StartTime: 22-Aug-05 12:02:00 AM
StopTime: 22-Aug-05 04:00 AM

Source Definition
Metrix Selection: Metrix: IMAGE (336)
Element Catalog Selection: Sel. Metrix contains no Catalogs
Statistics Selection: iDIO

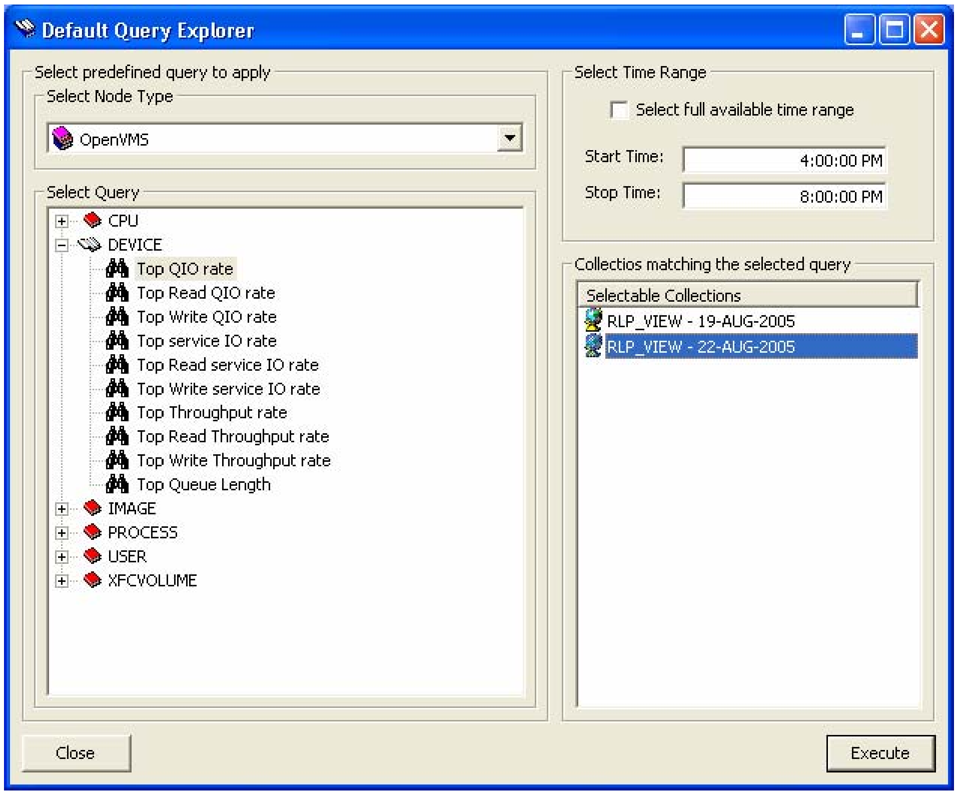
Reference Definition
Metrix Selection: Metrix: DEVICE (29)
Element Catalog Selection: Sel. Metrix contains no Catalogs
Statistics Selection: iQIOs

	DSA330
DSA310:[ELOS_RUN.ALPHA_EXE]OP_MSERVER.EXE;92	61.25 (100)
DSA320:[ELOS_RUN.SSVS.][000000]SSVS_SCAN.EXE;1	53.62 (100)
DSA310:[ELOS_RUN.ALPHA_EXE]OP_COM_SERV.EXE;90	44.26 (100)
$1$DGA300:[SYS61.SYSCOMMON.][SYSEXE]RDMMON71.EXE;1	34.67 (100)
1DGA400:[SYS63.SYSCOMMON.][SYSEXE]NET$ACP.EXE;1	34.33 (100)
DSA310:[ELOS_RUN.ALPHA_EXE]OP_CONTROL.EXE;94	32.65 (100)
DSA360:[PERFDAT.][BIN]PERFDAT_AUTOREP.EXE;8	30.55 (20.78)
1DGA400:[SYS63.SYSCOMMON.][SYSEXE]JBC$JOB_CONTROL.EXE;1	30.26 (100)
1DGA300:[SYS61.SYSCOMMON.][SYSEXE]QMAN$QUEUE_MANAGER.EXE;1	30.23 (100)
DSA360:[PERFDAT.][BIN]DQL$SRV.EXE;8	29.3 (20.78)
1DGA400:[SYS63.SYSCOMMON.][SYSEXE]TCPIP$SMTP_SYMBIONT.EXE;1	27.51 (100)
DSA310:[ELOS_RUN.ALPHA_EXE]OP_EE_TRANSF.EXE;609	27.25 (99.16)
DSA310:[ELOS_RUN.ALPHA_EXE]OP_AMF_INTF.EXE;1285	24.49 (100)
DSA310:[ELOS_RUN.ALPHA_EXE]OP_ACCOUNT.EXE;1174	22.87 (100)
DSA310:[ELOS_RUN.ALPHA_EXE]PROXY_KOB.EXE;179	21.13 (100)
DSA310:[ELOS_RUN.ALPHA_EXE]OP_GM_CTRL.EXE;1016	19.87 (100)
DSA310:[ELOS_RUN.ALPHA_EXE]OP_MAIN.EXE;1258	19.21 (100)
$1$DGA300:[SYS62.SYSCOMMON.][SYSEXE]OPCOM.EXE	18.7 (100)
DSA310:[ELOS_RUN.ALPHA_EXE]OP_TPCTRL.EXE;84	18.35 (100)
DSA360:[PERFDAT.][BIN]PERFDAT.EXE;9	17.78 (100)
$1$DGA400:[SYS64.SYSCOMMON.][SYSEXE]OPCOM.EXE	16.33 (100)
DSA310:[ELOS_RUN.ALPHA_EXE]OP_CRE_TRA.EXE;92	16.2 (100)
1DGA300:[SYS61.SYSCOMMON.][SYSEXE]TCPIP$NTP.EXE	15.78 (100)
DSA360:[PERFDAT.][BIN]PERFDAT_SNMP_WRK.EXE;8	15.73 (100)
_$1$DGA300:[MULTINET.TRANS3.SYSCOMMON.][MULTINET.PSCSSH]SSH2.EXE	14.49 (100)
DSA310:[ELOS_RUN.ALPHA_EXE]SC_TRA_WRM_INTF.EXE;92	10.75 (100)
1DGA300:[SYS61.SYSCOMMON.][SYSEXE]JBC$JOB_CONTROL.EXE;1	10.71 (100)
$1$DGA300:[SYS61.SYSCOMMON.][SYSEXE]ERRFMT.EXE;1	7.85 (100)
1DGA300:[SYS61.SYSCOMMON.][SYSEXE]TCPIP$SMTP_SYMBIONT.EXE;1	7.53 (100)

Figure 9 - Correlation report

- Predefined top statistics queries:

  The graphical user interface provides a top statistics explorer (see Figure 10). The top statistics explorer sorts the elements of a metric according to a selected predefined statistics of that metric, selects the data of the 6 top resource consuming elements and displays them as a line graph automatically (see Figure 11). The statistics available in the top statistics explorer are configured in a text file for simple customization.



Select Node Type: OpenVMS

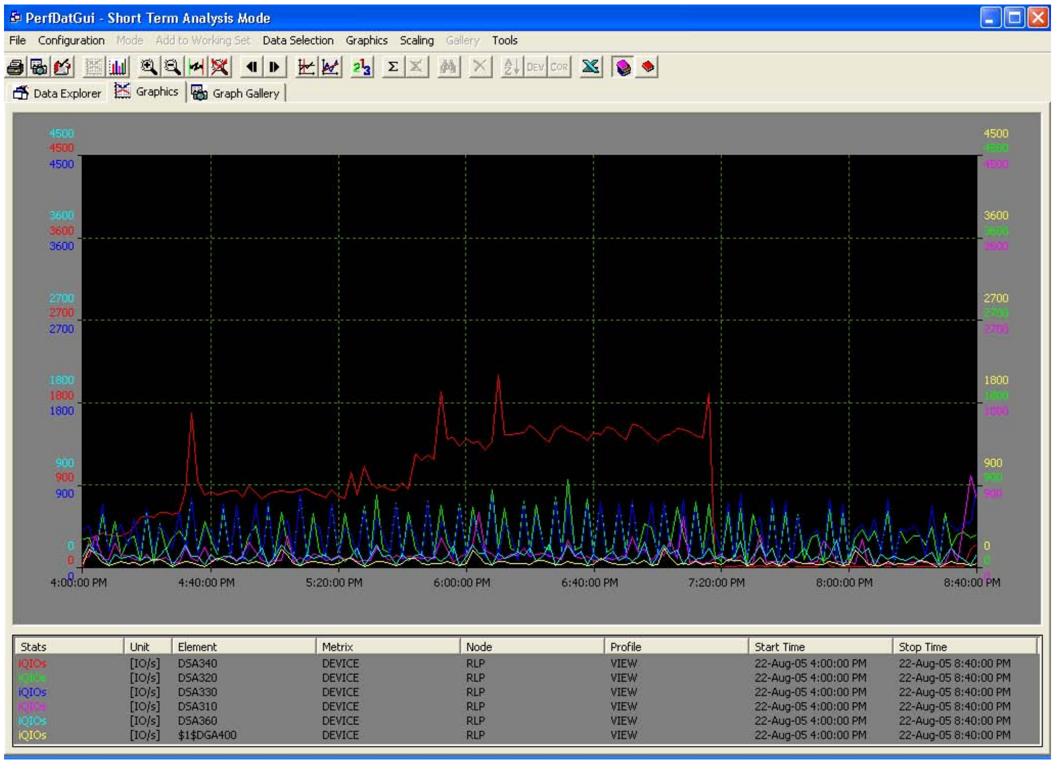
☐ Select full available time range

Start Time: 4:00:00 PM

Stop Time: 8:00:00 PM

RLP_VIEW - 22-AUG-2005

Figure 10 - HP PERFDAT GUI Top Statistics Explorer


PerfDatGui - Short Term Analysis Mode
File Configuration Mode Add to Working Set Data Selection Graphics Scaling Gallery Tools
Data Explorer Graphics Graph Gallery
4500
3600
2700
1800
900
0
4:00:00 PM
4:40:00 PM
5:20:00 PM
6:00:00 PM
6:40:00 PM
7:20:00 PM
8:00:00 PM
8:40:00 PM

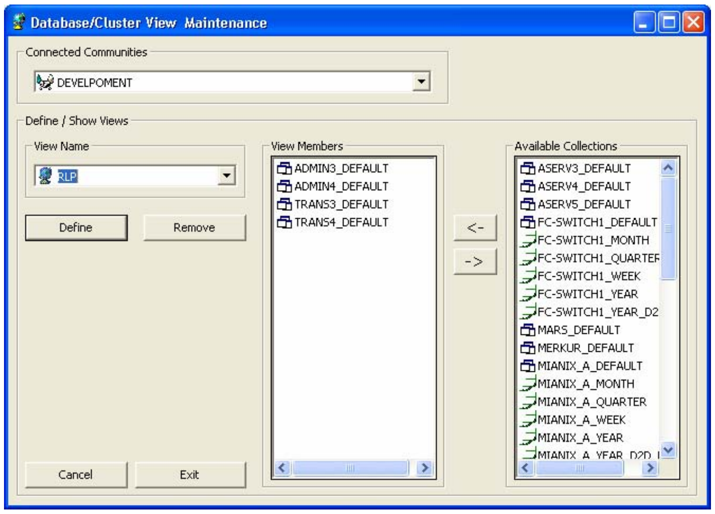
Stats	Unit	Element	Metrix	Node	Profile	Start Time	Stop Time
iQIOs	[IO/s]	DSA340	DEVICE	RLP	VIEW	22-Aug-05 4:00:00 PM	22-Aug-05 8:40:00 PM
iQIOs	[IO/s]	DSA320	DEVICE	RLP	VIEW	22-Aug-05 4:00:00 PM	22-Aug-05 8:40:00 PM
iQIOs	[IO/s]	DSA330	DEVICE	RLP	VIEW	22-Aug-05 4:00:00 PM	22-Aug-05 8:40:00 PM
iQIOs	[IO/s]	DSA310	DEVICE	RLP	VIEW	22-Aug-05 4:00:00 PM	22-Aug-05 8:40:00 PM
iQIOs	[IO/s]	DSA360	DEVICE	RLP	VIEW	22-Aug-05 4:00:00 PM	22-Aug-05 8:40:00 PM
iQIOs	[IO/s]	$1$DGA400	DEVICE	RLP	VIEW	22-Aug-05 4:00:00 PM	22-Aug-05 8:40:00 PM

Figure 11 - HP PERFDAT GUI Top Statistics Explorer – graphics output

- Private query explorer:

  You can easily define private data queries. This feature supports standard performance analysis performed on a regular basis. Once the queries for standard performance analysis have been defined, these queries can be simply applied to any other performance database that matches the criteria defined by the private queries and all the standard performance analysis is done. This saves valuable time compared to having to select all the data manually using the GUI data explorer whenever a standard analysis is performed.

- Easy to handle cluster view management and maintenance (see Figure 12).



Connected Communities: DEVELPOMENT

Define / Show Views

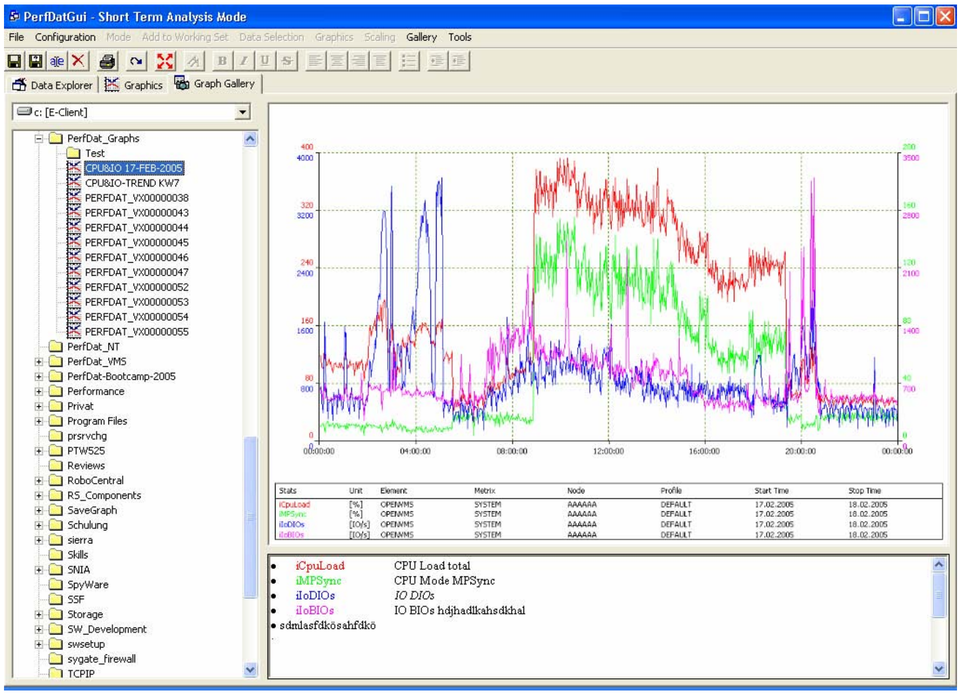
View Name: RLP

View Members:
ADMIN3_DEFAULT
ADMIN4_DEFAULT
TRANS3_DEFAULT
TRANS4_DEFAULT

Available Collections:
ASERV3_DEFAULT
ASERV4_DEFAULT
ASERV5_DEFAULT
FC-SWITCH1_DEFAULT
FC-SWITCH1_MONTH
FC-SWITCH1_QUARTER
FC-SWITCH1_WEEK
FC-SWITCH1_YEAR
FC-SWITCH1_YEAR_D2
MARS_DEFAULT
MERKUR_DEFAULT
MIANIX_A_DEFAULT
MIANIX_A_MONTH
MIANIX_A_QUARTER
MIANIX_A_WEEK
MIANIX_A_YEAR
MIANIX_A_YEAR_D2D

Figure 12 - Cluster view maintenance dialog window

- Multi window support for multi screen systems.
- Export capability to Excel.
- Any graphical output created can be stored locally on the PC for offline analysis / presentation (Graph Gallery - see Figure 13).



Stats	Unit	Element	Metrix	Node	Profile	Start Time	Stop Time
iCpuLoad	[%]	OPENVMS	SYSTEM	AAAAAA	DEFAULT	17.02.2005	18.02.2005
iMPSync	[%]	OPENVMS	SYSTEM	AAAAAA	DEFAULT	17.02.2005	18.02.2005
iIoDIOs	[IO/s]	OPENVMS	SYSTEM	AAAAAA	DEFAULT	17.02.2005	18.02.2005
iIoBIOs	[IO/s]	OPENVMS	SYSTEM	AAAAAA	DEFAULT	17.02.2005	18.02.2005

iCpuLoad	CPU Load total
iMPSync	CPU Mode MPSync
iIoDIOs	*IO DIOs*
iIoBIOs	IO BIOs hdjhadlkahsdkhal

- sdmlasfdkösahfdkö

Figure 13 - HP PERFDAT GUI Graph Gallery to view and maintain graphics stored locally on the PC

PERFDAT Environment

The PERFDAT environment consists of so-called communities (see Figure 14). A community is a logical partition of the whole environment and defines the database view when accessing the data via any system within a community. All systems of particular interest can be configured within the context of a community. No rules exist that limit the configuration of such communities (such as cluster boundaries or location of the systems). The number of possible communities ranges from one to the total number of systems within the whole environment. Figure 14 shows an example of partitioning the environment into communities and the role of the systems within the communities.

The role of the systems within a community is defined by the software components running on the systems.

- OpenVMS collector system.
- SNMP agent system (collects data from SNMP server systems).
- Archive system.
- Access server.
- SNMP server system (provides performance data via SNMP).

OpenVMS Collector System

A system is an OpenVMS collector system if the HP PERFDAT OpenVMS data collector is running on that system.

SNMP Agent

A system is a SNMP agent node if the HP PERFDAT SNMP extension is running on that system and data collections are active for remote systems providing performance data via SNMP (such as Tru64 UNIX or Brocade switches).

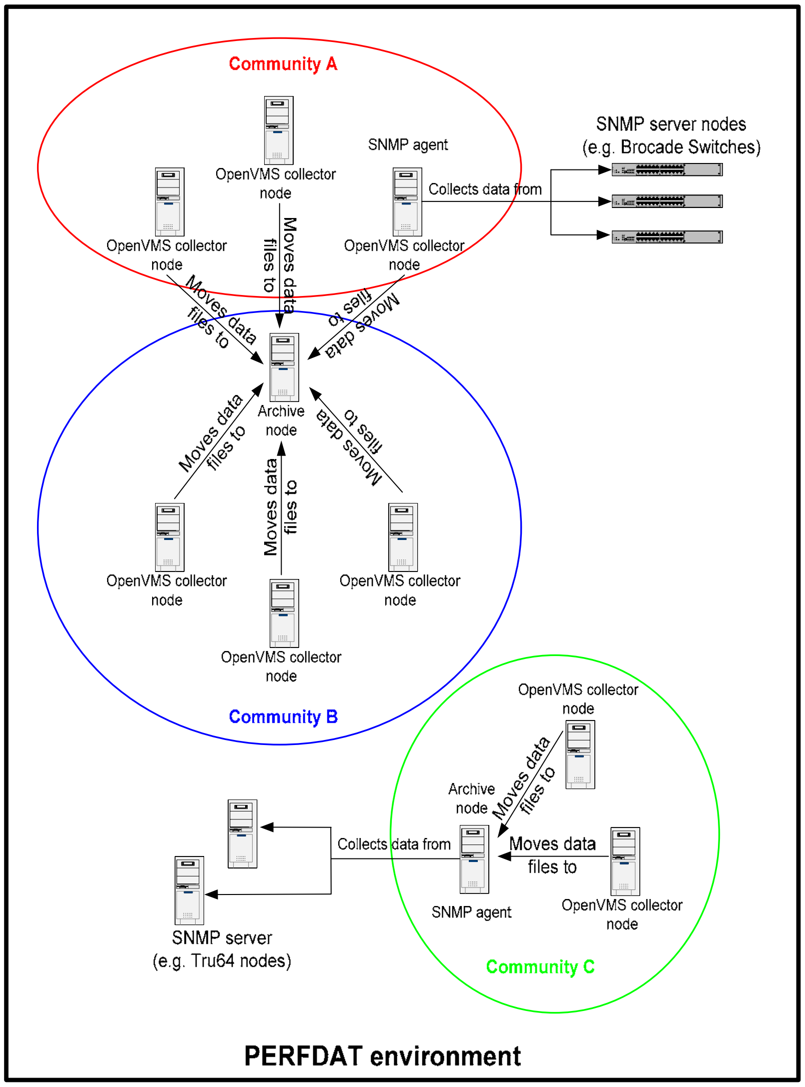


Figure 14 - HP PERFDAT environment example.

Archive System

A system is called an archive system if the HP PERFDAT Archiving service is configured on HP PERFDAT OpenVMS collector systems and/or SNMP agent to move collection data files to this system periodically.

Access Server

A system is called an access server, if the Data Query interface (DQL) is configured and started.

SNMP Server System

An SNMP server system is any non-OpenVMS system that provides performance data via SNMP and is supported by the SNMP extension.

As shown in Figure 14, an OpenVMS system configured within the PERFDAT environment can play several roles – it can be an OpenVMS collector system, a SNMP agent system, an archive system, and (not shown in Figure 14) an access server.

The main reason for defining communities is to have a selective view on the data when accessing the performance database via an access server. If you have configured your environment as shown in Figure 14, for example, and you access the performance database via any node of community C, all data collected by the community members is visible, regardless of where the data is actually stored within the community.

Accessing data via a collecting node provides a community-specific view of the distributed performance database. This behavior changes if the access server is an archive node. In this case, access is granted to all data stored locally regardless of whether the data files were created by any community member or not.

Community Example:

Looking at Figure 13, there is one node that is the archive node for community A and B, but it is a member of community B only. If you access data via that node the performance database view consists of all data of community A and all data of community B that has already been moved to the archive node. Consequently, even if the archive node is not a member of any community the user can access all data stored locally on the archive node when accessing the archive node.

If the archive node is not member of a community this does not imply that the user loses access to the data already moved to the archive node when accessing a community member. The archive node defined on the collecting node will always be queried for community data regardless of whether it is part of the community or not.

There are several reasons for defining archive nodes:

- Centralized data storage – single backup and restore location
- The statistic package provides several methods for advanced data analysis that reduce analysis time and provide the ability to identify performance bottlenecks without expert knowledge. These methods are very powerful, but running these methods may cause heavy I/O load on the system. Thus, if the data is stored on a production system, analysis runs can increase I/O load significantly and overall system performance may suffer. If data is stored on an archive node the data analysis can be done without negatively influencing the production systems.
- The auto-trend engine extracts trend and capacity reports from performance raw data. It is triggered once a day on any collecting node within your environment. Depending on the number of reports to be created automatically and their definition, the auto-trend engine may also stress the I/O subsystem. If an archive node is in use, only raw data already stored on that node will be accessed by the auto trend engine. Thus, the auto trend engine has – as long as the archive node is up and accessible -- no influence on the performance of the I/O subsystem on the collecting node.

HP PERFDAT Roadmap

New features planned for the next releases of HP PERFDAT:

- The current release of HP PERFDAT provides an easy-to-use utility to import Oracle Rdb performance data offline from binary RMU/SHOW STATISTICS files into the distributed performance database. With the next release of HP PERFDAT Oracle Rdb performance monitoring will be fully integrated into HP PERFDAT.
- The next release of HP PERFDAT will contain a performance advisory report engine. The performance advisory reporting engine will analyze performance data periodically. If a rule is fired, it will provide advisories on how to keep system performance on track.
- A future release of HP PERFDAT will contain a performance data collector for HSV storage controllers.

HP PERFDAT was developed based on OpenVMS customer requirements and in close cooperation with customers in order to exactly meet their needs. The HP PERFDAT roadmap is also triggered by HP PERFDAT customer feedback and requirements. Thus, the roadmap may change depending on HP PERFDAT customer needs.

Summary

HP PERFDAT performance solution for OpenVMS was developed based on a customer-defined requirements list. Due to its rock-solid, layered, easily extendable software design we were able to fulfill all additional customer needs that have been brought up since its first release by adding new software layers and components within a reasonable time frame.

HP PERFDAT performance solution for OpenVMS addresses today's OpenVMS customers most urgent performance management issues by providing an unprecedented level of insight into multi-system performance at low cost. A complete suite of highly automated collection, filtering, charting, and trend analysis capabilities provides the customer's IT staff with accurate and complete performance information for effective performance lifecycle management.

Customers in different business areas – banks, insurance companies, semiconductor industries, healthcare, lotteries, telecom providers, and manufacturing industries -- use HP PERFDAT.

For all of them, HP PERFDAT has proven the added value it provides to system and performance management. Here are just a few examples:

- A bank faced a problem when the processing time of the end-of-day jobs increased from one day to the other unexpectedly. Since they had been using HP PERFDAT continuously they had detailed performance information about the system. They used the GUI to compare performance before and after the degradation and thus they were able to identify the root cause of their problem. The write I/O response time of some (not all) production disks increased significantly. The reason was that after adding new units on their storage arrays the write-back cache performance of one of their storage arrays has degraded significantly for particular LUNs.
- A telecom provider applied a Fibre Channel patch to their systems running OpenVMS V7.3-2. After rebooting the system the overall system performance decreased significantly. After a short look at the performance data provided by HP PERFDAT and comparing the data to data collected before the patch has been applied, they found that the MPSync rate had increased dramatically. The fast-path assignments of the Fibre Channel and network devices had changed. They switched the fast-path assignments manually and the systems again performed as expected from the end-user perspective. To prove to the end users that this was the only cause of the performance loss, they ran some automated performance comparisons using the HP PERFDAT deviation analysis feature that showed that all OpenVMS subsystems performed in the same manner after manually switching the fast path assignments as they did before applying the patch.
- A lottery company upgraded to a new OpenVMS and middleware version. During the application tests, application processes sometimes crashed. These process crashes were not reproducible. Sometimes their tests succeeded and sometimes they failed. The crash and error messages provided by the middleware indicated that the crash reason was due to resource problems but there was no hint of which resources were affected. By analyzing the performance data provided by HP PERFDAT they were able to prove that the process crashes were due to misconfigured memory parameters -- not OpenVMS quotas but internal middleware parameters. By adjusting these memory parameters they were able to fix this problem. In addition HP PERFDAT supported them in tuning their system. They used HP PERFDAT during all of their testing with high resolution and received a clear and complete picture of how any configuration changes of the middleware affected application and overall performance. The tuning phase led to a configuration set that provides proven optimal performance.
- A healthcare company running a huge system with more then 3000 concurrent users experienced system performance that degraded occasionally and unexpectedly during the night. By using HP PERFDAT, system managers were able to pinpoint the source of the problem. It turned out that operators – without informing system management in advance -- submitted cleanup jobs that deleted up to 200.000 files at once from a directory which in turn caused the response and performance problems.
- Another bank found that the modification date of the SYSUAF files were being modified periodically although none of the system management team changed any SYSUAF records. When they first detected this security issue they used HP PERFDAT to identify the source. They started an additional data collection that monitored all I/Os from all processes to the SYSUAF files only. The next time the modification date of the SYSUAF files changed, the HP PERFDAT performance data directly showed them which process accessed the SYSUAF files and what type of access was used (Read, Write, Control I/Os).

In order to help you get the most out of HP PERFDAT within the minimum of time, the HP PERFDAT performance solution for OpenVMS is distributed as a service that includes:

- Software usage.
- Software maintenance service.

- Software upgrade service (Right to use new versions).
- Installation support and initial configuration service.

For more information

For further information about HP PERFDAT performance solution for OpenVMS please email the author, HP PERFDAT support at or our partner, Compinia GmbH & Co KG, at PERFDAT@COMPINIA.DE.

If you want to test HP PERFDAT performance solution for OpenVMS please contact the HP PERFDAT support link at or PERFDAT@COMPINIA.DE to receive a copy of the software. When installing the OpenVMS kit for the first time, a 30-day full trial license is automatically applied. The HP PERFDAT GUI requires no separate license.

OpenVMS Technical Journal V8

Bringing Vegan Recipes to the Web with OpenVMS

Bernd Ulmann (ulmann@vaxman.de)

Introduction

In 2001 my girl friend, Rikka, who was studying computer science, had a lab concerning databases. Being a vegan, she decided to develop a demonstration database for vegan recipes with a simple web front end. Since she loves to cook the database grew from a couple of recipes to several dozen very quickly. It turned out that there were people in Germany quite interested in vegan cooking so we decided to make the database publicly available via the Internet.

This was the time when area-wide DSL infrastructure became available in Germany and we decided to use her LINUX-based PC as a small MySQL and Apache server using some CGI scripts written in Perl. After a couple of years of growing interest it became clear that this interim solution was not adequate any longer. The PC encountered minor hardware problems and LINUX just isn't for someone like me running a VAX-7820 at home. So we decided to migrate the recipe database -- which has grown to the largest German database of vegan recipes and sees thousands of requests per day -- from the small LINUX system to FAFNER, the VAX.

We need to talk a bit about the details of the database model and the CGI scripts to understand the decisions described in the following, so these points will be outlined below.

Structure of the database and CGI scripts

The database model of the recipe database features twelve tables, some of which are described in the following:

Recipes	The recipe itself. This table contains the name and a descriptive text for each recipe as well as some foreign keys to other tables.
Units	All necessary units for measuring ingredients.
Ingredients	All ingredients used to create a meal.
recipe_ingredients	Since recipes and ingredients form an n-times-m relationship, this table

	is necessary to link a particular recipe with all of its ingredients.
categories	Categories for the recipes like pasta, salad, and so on.
recipe_categories	n-times-m link between categories and recipes.

Access to the recipe database is performed by three CGI scripts written in Perl. The first of these, RZ1.PL, creates an HTML form containing several selection lists: one for the categories, one for the ingredients, and so on. These lists can be used to narrow the search for a particular recipe. If you are looking for recipes using “chanterelle” you could select "Chanterelle" in the ingredients selection list and then click the search button on the bottom of the form created by RZ1.PL.

This will trigger the next stage of recipe selection, namely the Perl script RZ2.PL. This CGI script now reads all selection parameters and creates a proper SQL statement to select all recipes from the database satisfying the conditions selected in the previous step.

Using this list of available recipes, RZ2.PL creates another HTML form with only one selection list containing all titles of the matching recipes. In the current example this list will contain seven entries, one of them being "Mushroom Carpaccio." Selecting a recipe from this list and clicking the search button at the bottom of this second form will trigger RZ3.PL, the last CGI script.

RZ3.PL now reads the title of the recipe selected before and starts gathering all necessary information to display the recipe, a picture of the resulting meal, and so forth. This is by far the most complicated step since all twelve tables are involved in this operation.

Bringing all this to OpenVMS

The first step in bringing all this to OpenVMS was to decide which database to use. Since there is a port of MySQL for OpenVMS Alpha, it seemed like a good idea to try porting it to OpenVMS VAX. Unfortunately this turned out to be very complicated. The first obstacle -- compiler qualifiers not applicable for a VAX -- was resolved easily using some small Perl scripts to automatically modify the build procedures.

Later it turned out that porting MySQL to OpenVMS VAX was not easy at all. In fact, I had to give up after some time. One of the problems was that I was unable to change all occurrences of 64-bit integers (long long) by setting appropriate #define clauses. Some floating-point routines proved to be quite tricky, too, so I eventually decided to abandon the idea of running MySQL on a VAX. For a short moment I thought about replacing FAFNER with a more modern Alpha-based system like an AlphaServer 4100 but I couldn’t do it – living without a large VAX in the basement seemed horrible to me.

Having ruled out MySQL, I took a look at other systems like Postgres and the like. After digging into many lines of source and thinking about databases over and over again it became clear that using RDB was the way to go. No more tinkering with products not being designed for OpenVMS and no more concessions.

How to migrate from MySQL to RDB

After installing RDB version 7.0 on FAFNER it was time to set up an empty database to hold all of the recipe data. This was done with a DCL procedure as shown below (ellipses denote omitted code):

```
$ sql
create database alias recipes filename disk$rdb_data:[000000]recipes
number of buffers is 1000
number of users is 100
row cache is enabled
!
grant all on database alias recipes to [ulmann];
grant select,update on database alias recipes to [http$nobody];
!
set dialect 'sql92';
...
create table recipes.recipes (
        rnr int not null,
        name varchar (80) not null,
        enr int not null,
        anr int not null,
        regnr int not null,
        description long varchar,
        pnr int,
        photo varchar (80),
        source int,
        counter int,
        story long varchar,
        monthcounter int,
        tim char(14),
        instim char(14),
        cookbook int,
        primary key (rnr));
...
commit;
...
create unique index recipes.irecipes on recipes.recipes (rnr);
...
commit;
...
grant all on table recipes.recipes to [ulmann];
grant select,update on table recipes.recipes to [http$nobody];
...
commit;
exit
$exit
```

Having done this resulted in an empty database with all necessary tables (the fact that the user HTTP$NOBODY needs update privilege for the recipe table stems from the fact that two counters will be incremented to keep track of recipe requests). The next problem was to move all of the data from the MySQL database running on LINUX to the new RDB database running on VMS.

The first idea was to create one text file for each table on the MySQL system and extract the table data using standard SQL. Using Perl for necessary format changes and standard RDB methods it should be possible to load the raw data from these files into the corresponding RDB tables.

After a quick experiment it turned out to be a clumsy and inelegant solution so it was abandoned. Then the idea of using a Perl script to connect to the MySQL system running on LINUX and the RDB system running on OpenVMS emerged. This script, MYSQL2RDB.PL, makes use of three Perl modules -- Net::MySQL, DBI, and finally DBD::RDB.

Net::MySQL, a pure Perl implementation of the MySQL interface, was chosen to connect to the MySQL system due to the fact that this script did not need low overhead since it was to be used only once. This seemed to justify not compiling a native MySQL interface on the VAX.

After connecting to MySQL and RDB, the script copies one table after the other using an array containing the names of all tables to be copied. The main loop looks like this:

```
for my $table (@tables)
```

```
{
  print "Deleting from $table...\n";
  $rdb -> do ("delete from recipes.$table");

  $mysql -> query ("select * from $table");
  my $record_set = $mysql -> create_record_iterator;
  my @fields = $record_set -> get_field_names;

  my $statement = "insert into recipes.$table (" .
                join (',', @fields) . ') values (' .
                join (',', map {'?'} 0..$#fields) . ')';

  my $rdb_sth = $rdb -> prepare ($statement);
  my $counter = 0;

  print "Inserting into recipes.$table: ";
  while (my $record = $record_set -> each)
  {
    $rdb_sth -> execute (@$record);
    $rdb -> commit unless $counter++ % 50;
  }
  $rdb -> commit if $counter % 50;
  print "\nInserted $counter lines into recipes.$table\n";
}
```

First all data is read from the MySQL source table using the statement select * from $table. Applying the method create_record_iterator results in an iterator which can be used to loop over the result set of the select operation.

To be able to insert the data just read into the RDB destination table a proper insert statement has to be created. First, a list of all column names will be needed which can be generated by applying the get_field_names method. The statement resulting from the line my $statement = will look like this:

```
insert into recipes.<table_name>
(<column_name>, <column_name>, ...)
values
(?, ?, ...)
```

After preparing this statement the data is copied in a loop performing an execute call for each row of raw data.

In the first version of this migration script the connection to RDB was made with default values implying autocommit. This made the script run incredibly slow since some tables contain tens of thousands of rows of data. Turning autocommit off and performing an explicit commit every 50 rows increased the performance by nearly a factor of 100.

Another lesson learned was that writing a db-trace file (which was enabled during debugging) takes more time than writing to the database, thus slowing things down considerably.

After modifying these procedures, the script copied the whole database flawlessly from MySQL to RDB in less then two minutes. The next step concerned the CGI scripts.

Selecting a web server and tuning Perl-based CGI scripts

Selecting a web server for this application was simple: WASD. There is just nothing that can compare to WASD when it comes to web servers for OpenVMS. It is true that, as a VAX, FAFNER can't use the Apache-based CSWS. But, the fact is, WASD has far less overhead than CSWS.

With the web server set, the next thing to do was configure WASD in a way that allowed the use of Perl-based CGI scripts. This turned out to be a simple task after having a look at the documentation.

After adapting the three Perl programs RZ1.PL, RZ2.PL and RZ3.PL in a way that allowed the use of RDB, the time for a first test had come. It worked out of the box! But it was much slower than the original system running under LINUX.

To be honest, this was exactly what I expected. The LINUX system was running on a recent PC with a several-GHz-CPU and lots of memory while FAFNER, a VAX-7820, is quite slow in comparison. It was clear that some tuning would be necessary.

The first thing to do was eliminate the overhead resulting from starting the rather large Perl interpreter every time a Perl-based CGI script was to be run. Fortunately, WASD offers everything necessary for this -- namely CGIplus and RTE, the Run-Time Environment. Using these techniques it is possible to have the Perl interpreter start only once (more or less) and the scripts parsed only once, too.

After installing and compiling the necessary software for this, the response time of the system skyrocketed. The invocation of CGI scripts became three to five times faster -- as fast as on the LINUX system!

How to tune RDB using Perl

The next problem to tackle involved accessing the RDB database system on FAFNER. It turned out to be a lot slower than using MySQL on the LINUX system. It was clear that something had to be done. It quickly became obvious that a crucial index was missing on the RDB database. Creating this index made things a lot faster but still not as fast as on the LINUX system. I decided to ask my friend, Thomas Kratz (a Perl guru), if he thought that a caching database proxy written in Perl might be a solution.

The idea behind this is as follows: A dedicated server process, the proxy will connect to the RDB database and listen to a particular port for connections from client processes. If some process wants to perform a database operation it will wrap up its request in a hash data structure and send this to the proxy. The proxy will unwrap the request and have a look into its local cache. If no appropriate data has been cached previously it will route the request to the RDB system, cache the resulting data set and send it back to the client program. If the data was found in the proxy's cache the RDB request will be omitted and the cached data sent immediately.

After only two evenings of programming, Thomas came up with a small yet powerful Perl program which did everything necessary to serve as an RDB proxy. It expects a request in the form of a hash like this:

```
{
  SELECT => 'SELECT DESCRIPTION FROM RECIPES',
  ALIAS  => 'RECIPES_PRODUCTION',
}
```

The first key may be either SELECT or DO distinguishing between requests that do not alter data and those that do. (The latter will normally result in clearing all affected cache entries. There is a way to circumvent this in case it is clear that only minor changes like incrementing a counter in the database will be performed. But this leads into too much detail.) The second key, ALIAS, contains the name of the RDB database on which the statement is to be performed, so a single proxy server can be used to process requests from the production system as well as requests from a test system.

The problem of sending a complete Perl data structure through a socket interface was solved like this: The calling program, the client (RZ1.PL, etc.), first opens a socket to write to:

```
my $sock = IO::Socket::INET->new
```

```
(
  PeerAddr  => $host,
  PeerPort  => $port,
  ReuseAddr => 1,
) or return("Can't connect to $host:$port: $!\n");
```

Then it sets up the data structure to be sent using $statement and writes it to the socket as shown below:

```
my $stmt_type = $stmt =~ /^\s*select/i
              ? 'SELECT'
              : 'DO'
              ;
print $sock encode_base64(nfreeze(
  {
   $stmt_type => $stmt,
   ALIAS      => 'RECIPES_PRODUCTION',
  }
), '', ), "\n";
```

The scalar $stmt_type contains SELECT or DO depending on the type of the SQL statement to be performed. This will be used as a part of the structure

```
{
  $stmt_type => $stmt,
  ALIAS      => 'RECIPES_PRODUCTION',
}
```

which is packed using the nfreeze function of the module Storable and base64 encoded before being written to the socket.

The proxy server then reads $raw from the socket and restores the data structure using a statement like this:

```
my $request_ref = eval {thaw(decode_base64($raw))};
```

This will result in a reference to a hash having the exactly same structure as the hash sent by the client. This hash will then be parsed by the proxy server to extract all necessary information to perform the action being requested.

Sending the resulting data set from the server to the client works quite the same but with a tiny difference: While the request from the client will always be short enough to be written to the socket in a single step, the resulting data set may be several hundred kB in size, exceeding the maximum amount of data which can be written to a socket in one operation. This requires the server to send the results in chunks using a sequence like this:

```
print $sock $_, "\n"
  for unpack("A$chunk_size" x (int(length($response) / $chunk_size) + 1),
    $response);
```

This will split the contents of the scalar $response into several smaller chunks of data which will be sent to the client through the socket $sock. The client then concatenates these chunks and works with the data thus returned.

Advantage of using a proxy server

Using this proxy server results in a dramatic reduction in response time as the following example shows:

The first request from RZ1.PL to build the category selection list takes 3.52 seconds using the proxy since the cache is empty and the request has to be actually performed on the RDB system. All following requests for building this list require about 0.34 seconds – a factor of ten faster than the direct access method!

Conclusion -- or sheer CPU speed is not everything

Looking back I have to admit that I underestimated the efforts necessary to port this application from a LINUX system to a VAX running OpenVMS. If the destination was a more modern system like an Alpha or an Itanium things would have been much easier since it would have been possible to use MySQL. This, and the sheer computational and IO speed of such a system, would have been more than sufficient to get faster response times than the LINUX system was able to offer.

Nevertheless I am glad that I decided to stay with my VAX. I learned a lot of things I would have missed otherwise and using all of the techniques sketched above it was possible to get performance from a VAX that was equal to if not better than that delivered by the original LINUX system.

Choosing RDB as the heart of this strictly non-commercial application was a good decision. It is quite easy to use and integrates perfectly with OpenVMS running on the VAX. Backing up databases and restoring them is a matter of seconds and performance is quite impressive for a VAX.

There was never any doubt that Perl is the ideal language for nearly everything -- not just CGI scripts as could be seen above. The minor drawback of being an interpreted language is more than compensated for by the expressive power of Perl as well as the possibility of using a variety of programming styles – functional, object-oriented, and purely imperative. No other language would have allowed writing the CGI scripts and the proxy server in such a short amount of time.

Of course, all of the techniques described above can also be used on Alpha and Itanium systems with equal reductions in response time. And the MySQL/RDB-conversion script may be applied in other areas as well.

The overall system has been running flawlessly for several months now with an increasing load of recipe requests. The VAX is rock solid and is down only when we encounter a power failure -- about twice a year. The LINUX system has been shut down and the last non-OpenVMS system has vanished from the house.

For more information

You may contact author at ulmann@vaxman.de, to get to the latest issue of the OpenVMS Technical Journal, go to:

http://www.hp.com/go/openvms/journal.

.