HP OpenVMS Technical Journal V2 (July 2003) 1

A Survey of Cluster Technologies 7

Local Area Network Cluster Interconnect Monitoring 30

A Survey of Cluster Technologies

Ken Moreau
Solutions Architect, OpenVMS Ambassador, MCSE

Overview

This paper discusses the cluster technologies for the operating systems running on HP server platforms, including HP-UX, Linux, NonStop Kernel, OpenVMS, Tru64 UNIX and Windows 2000. I describe the common functions which all of the cluster technologies perform, show where they are the same and where they are different on each platform, and introduce a method of fairly evaluating the technologies to match them to business requirements. I do not discuss performance, base functionality of the operating systems, or system hardware.

This article draws heavily from the HP ETS 2002 presentation of the same name.

Introduction

Clustering technologies are highly inter-related, with almost everything affecting everything else. But I have broken this subject into five areas:

- Single/multi-system views, which defines how you manage and work with a system, whether as individual systems or as a single combined entity.
- Cluster file systems, which defines how you work with storage across the cluster. Cluster file systems are just coming into their own in the UNIX world, and I will talk about how they work in detail.
- Configurations, which defines how you assemble a cluster, both physically and logically.
- Application support, which discusses how applications which are running on your single standalone system today, can take advantage of a clustered environment. Do they need to change, and if so how? What benefits are there in a clustered environment?
- Resilience, which talks about when bad things happen to good computer rooms. This covers things like host-based RAID, wide area "stretch" clusters, extended clusters, and disaster tolerant scenarios.

I cover the capabilities of Linux LifeKeeper, NonStop Kernel G06.13, Serviceguard 11i both for HP-UX and Linux, TruCluster V5.1b, OpenVMS Cluster Software V7.3-1, and Windows 2000 DataCenter system clusters.

For Linux, I focus on the High Availability side, not the HPTC (i.e., Beowulf) technologies.

Single-System and Multi-System-View Clusters

In order to evaluate the cluster technologies fairly, we need to define four terms: scalability, reliability, availability and manageability.

- Availability defines whether the application stays up, even when components of the cluster go down. If I have two systems in a cluster and one goes down but the other picks up the workload, that application is available even though half of my cluster is down. Part of availability is failover time, because if it takes 30 seconds for the application to fail over to the

other system, the users on the first system think that the application is down for those 30 seconds.

- Reliability defines how well the system performs during a failure of some of the components. If I get sub-second query response and if my batch job finishes in 8 hours with all of the systems in the cluster working properly, do I still get that level of performance if one or more of my systems in the cluster is down? If I have two systems in a cluster and each system has 500 active users with acceptable performance, will the performance still be acceptable if one of the systems fails and there are now 1,000 users on a single system? Keep in mind that your users neither know nor care how many systems there are in your cluster, they simply care whether they can rely on the environment to get their work done.

  Notice that reliability and availability are orthogonal concepts, where you can have one but not the other. How many times have you logged into a system (i.e., it was available), but it was so slow as to be useless (i.e., it was not reliable)?

- Scalability defines the percentage of useful performance you get from a group of systems. For example, if I add a second system to a cluster, do I double my performance, or do I get a few percentage points less than that? If I add a third, do I triple the performance of one, or not?

- Manageability tells us how much additional work it is to manage those additional systems in the cluster. If I add a second system to my cluster, have I doubled my workload because now I have to do everything twice? Or have I added only a very small amount of work, because I can manage the cluster as a single entity?

Multi-system-view clusters are generally comprised of two systems, where each system is dedicated to a specific set of tasks. Storage is physically cabled to both systems, but each file system can only be mounted on one of the systems. This means that the applications cannot simultaneously access data from both systems at the same time. It also means that the operating system files cannot be shared between the two systems, so there needs to be a fully independent boot device (called a "system root" or "system disk") with a full set of operating system and cluster software files for each system.

Multi-System-View Clusters in an Active-Passive Mode

Multi-system-view clusters in an active-passive mode are the easiest for vendors to implement. They simply have a spare system on standby in case the original system fails in some way. The spare system is idle most of the time, except in the event of a failure of the active system. This is why it is called active-passive, because one of the systems is active and the other is not during normal operations. This is classically called N+1 clustering, where N=1 for a two system cluster. For clusters with larger numbers of systems, you would have a spare server that could take over for any number of active systems.

Failover can be manual or automatic. Because both systems are cabled to the same storage array, the spare system will be monitoring the primary system and can start the services on the secondary system if it detects a failure of the primary system. The “heartbeat” function can be over the network or by some private interface.

In a multi-system-view cluster in an active-passive mode, the availability, reliability, scalability, and manageability characteristics can be described as follows:

- Availability is increased because you now have two systems available to do the work. The odds of both systems being broken at the same time are fairly low but still present.

- Reliability can be perfect in this environment, because if the two systems are identical in terms of hardware, the application will have the same performance no matter which system it is running on.

- Scalability is poor (non-existent?) in an active-passive cluster. Because the applications cannot access a single set of data from both systems, there is no scalability in a multi-system-view cluster. You have two systems' worth of hardware doing one system's worth of work.
- Manageability is poor, because it takes approximately twice as much work to manage as that of a single system. Because there are two system roots, any patches or other updates need to be installed twice, backups need to be done twice, etc. Further, you have to test the failover and failback, which adds to the system management workload

Notice that the second system is idle most of the time, and you are getting no business benefit from it. So the other alternative is to have both systems working.

Multi-System-View Clusters in an Active-Active Mode

The physical environment of a multi-system-view cluster in an active-active mode is identical to that of the active-passive mode. There are generally two systems, physically cabled to a common set of storage, but only able to mount each file system on one of the systems. The difference in this case is there are multiple systems that are performing useful work as well as monitoring each other's health. However, they are not running the same application on the same data, because they are not sharing anything between the systems.

For example, a database environment could segment their customers into two groups, such as all people with last names from A-M and from N-Z. Then each group would be set up on a separate partition on the shared storage, and each system would handle one of the groups. This is known as a "federated" database. Or one of the systems could be running the entire database and the other system could be running the applications that access that database.

In the event of a failure, one system would handle both groups.

This is called an N+M cluster, because any of the systems can take over for any of the other systems. One way to define N and M is to think about how many tires you have on your automobile. Most people automatically say four, but then realize that they really have five tires, operating in an N+1 environment, because four tires are required for minimum operation of the vehicle. A variation is to use the "donut" tires, which offer limited performance but enough to get by. This can be thought of as having 4½ tires on the vehicle. The key is to define what level of performance and functionality you require, and then define N and M properly for that environment.

Failover can be manual or automatic. The "heartbeat" function can be over the network or by some private interface.

In a multi-system-view cluster in an active-active mode, the availability, reliability, scalability, and manageability characteristics can be described as follows:

- Availability is increased because you now have two systems available to do the work. Just as in the active-passive environment, the odds of both systems being broken at the same time are fairly low but still present.
- Reliability is not guaranteed in this situation. If each system is running at 60% of capacity, then a failure will force the surviving system to work at 120% of capacity, and we all know what happens when you exceed about 80% of capacity.
- Scalability is poor in this situation because each workload must still fit into one system. There is no way to spread a single application across multiple systems.
- Manageability is slightly worse than the active-passive scenario, because you still have two independent systems, as well as the overhead of the failover scripts and heartbeat.

Failover of a Multi-System-View Cluster (Active-Active or Active-Passive)

One of the factors affecting availability is the amount of time it takes to accomplish the failover of a multi-system-view cluster, whether active-active or active-passive. The surviving system must:

- Notice that the other system is no longer available, which is when the "heartbeat" function on the surviving system does not get an answer back from the failed system.
- Mount the disks that were on the failing system. Remember that the file systems are only mounted on one system at a time: this is an unbreakable rule, and a definition of multi-system-view clusters. So the surviving system must mount the disks that were mounted on the other system. If you have a large number of disks, or large RAIDsets, this could take a while.
- Start the applications that were active on the failing system.
- Initiate the recovery sequence for that software. For databases, this might include processing the re-do logs in order to process any in-flight transactions which the failing system was performing at the time of the failure.

In large environments, it is not unusual to have this take 30-60 minutes. And during this recovery time, the applications that were running on the failed system are unavailable, and the applications that were running on the surviving system are suffering from reliability problems, because the single system is now doing much more work.

Single-System-View Clusters

In contrast, single-system-view clusters offer a unified view of the entire cluster. All systems are physically cabled to all storage and can directly mount all storage on all systems. This means that all systems can run all applications, see the same data on the same partitions, and cooperate at a very low level. Further, it means that the operating system files can be shared in a single "shared root" or "shared system disk," reducing the amount of storage and the amount of management time needed for system maintenance. There are no spare systems. All systems can run all applications at all times. And in a single-system-view cluster, there can be many systems. In a single-system-view cluster, the availability, reliability, scalability, and manageability characteristics can be described as follows:

- Availability is increased because you now have multiple systems to do the work. The odds of all systems being broken at the same time is now much lower, because you can have potentially many systems in the cluster.
- Reliability is much better, because with many systems in the cluster, a failure of a single system will allow that workload to be spread across many systems, increasing their load only slightly. For example, if each system is running at 60% capacity and one server out of four fails, you would take 1/3 of the load and place it on each of the other systems, increasing their performance to 80% of capacity, which will not affect reliability significantly.
- Scalability is excellent because you can spread the workload across multiple systems. So if you have an application which is simply too big for a single computer system (even one with 64 or 128 CPUs and hundreds of gigabytes of memory and dozens of I/O cards), you can have it running simultaneously across many computer systems, each with a large amount of resources, all directly accessing the same data.
- Manageability is much easier than the equivalent job of managing this number of separate systems, because the entire cluster is managed as a single entity. So there is no increase in management workload even when you have lots of smaller systems.

Notice how this gives some advantages in failover times over multi-system-view clusters by not having to do quite so much work during a failover:

- The surviving system must detect the failure of the other system. This is common between the two schemes.
- The surviving systems do not have to mount the disks from the failed system: they are already mounted.
- The surviving systems do not have to start the applications: they are already started.
- The execution of the recovery script, is common between the two schemes, and can begin almost instantly in the single-system-view cluster case. The application recovery time will be similar on both systems, but if you have a larger number of smaller systems, you can achieve parallelism even in recovery, which means your recovery might be faster in this case as well.

How Do the Clusters on HP Systems Fit into These Schemes?

So now that we understand the terms, how do the clusters on HP systems fit into these schemes?

	Multi-system view	Single-system view	Shared root
LifeKeeper Linux, Windows	Yes	No	No
Serviceguard	Yes	No	No
NonStop Kernel	Yes	Yes	Each node (16 CPUs)
TruCluster	No	Yes	Yes
OpenVMS Cluster Software	Yes	Yes	Yes
Windows 2000 DataCenter	Yes	No	No

Linux Clustering

Linux clustering is split between massive system compute farms (Beowulf and others) and a multi-system-view, failover clustering scheme. I am not talking about the High Performance Technical Computing market here, which breaks down a massive problem into many (hundreds or thousands) of tiny problems and hands them off to many (hundreds or thousands) of small compute engines. The point is that this is not a high availability environment, because if any of those compute engines fails, that piece of the job has to be restarted from scratch.

The high availability efforts going on with SuSE and others are focused on multi-system-view clusters consisting of two systems so that applications can fail over from one system to the other. I will talk later about some cluster file system projects such as Lustre, GFS and PolyServe, but these do not offer shared root, so systems in Linux clusters require individual system disks.

I will not be discussing the work being done by HP as part of the Single System Image Linux project, because it is not yet a product. But when this becomes a product, it will have significant capabilities that match or exceed every other clustering product.

Serviceguard

Serviceguard is a multi-system-view failover cluster. Each system in a Serviceguard cluster requires its own system disk. There are excellent system management capabilities via the Service Control Manager and the Event Management Service, including the ability to register software in the System Configuration Repository, get system snapshots, compare different systems in the cluster, etc. It is also well integrated with HP/OpenView.

Himalaya NonStop Kernel

The Himalaya NonStop Kernel can be configured either as a multi-system-view cluster or, more commonly, as a single-system-view cluster. It offers the best scalability in the industry, in fact, true linear scalability because of the superb cluster interconnect, both hardware and software. Each cabinet of up to 16 processors can share a system disk and is considered one system.

TruCluster V5.1b

TruCluster V5.1b represents a major advance in UNIX clustering technology. It can only be configured as a single-system-view, with all of the focus on clustering being to manage a single system or a large cluster in exactly the same way, with the same tools, and roughly the same amount of effort. It offers a fully shared root, with a single copy of almost all system files.

OpenVMS Cluster Software

OpenVMS Cluster Software has always been the gold standard of clustering. It also can be configured as either multi-system view or single-system view, although the most common is single-system view. It supports a single or multiple system disks.

Windows 2000 DataCenter

Windows 2000 DataCenter is a multi-system-view failover scheme. Applications are written to fail over from one system to another. Each system in a Windows 2000 DataCenter cluster requires its own system disk. This is not to say that there aren't tools like the Cluster Administrator, which can ease some of this burden, but they are still separate system disks that must be maintained.

Cluster File Systems

Cluster file systems are how systems communicate with the storage subsystem in the cluster. There are really two technologies here: how a group of systems communicates with volumes that are physically connected to all of the systems, and how a group of systems communicates with volumes that are only physically connected to one of the systems.

Network I/O allows all of the systems in a cluster to access data, but in a very inefficient way that does not scale well. Let's say that volume A is a disk or tape drive which is physically cabled to a private IDE or SCSI adapter on system A. It cannot be physically accessed by any other system in the cluster, so if any other system in the cluster wants to access files on it, it must do network I/O, usually by some variation of NFS.

Specifically, if system B wants to talk to the device that is mounted on system A, the network client on system B communicates to the network server on system A, in the following way:

- An I/O is initiated across the cluster interconnect from system B to system A.
- System A receives the request, and initiates the I/O request to the volume.

- System A gets the data back from the volume, and then initiates an I/O back to system B.

Notice that there are three I/Os for each disk access. For NFS, there is also significant locking overhead with many NFS clients. This leads to poor I/O performance with an active/active system.

So why does every system offer network I/O? To deal with single-user devices that cannot be shared, such as tapes, CD-ROM, DVD or diskettes, and to allow access to devices that are on private communications paths, such as disks on private IDE or SCSI busses.

In contrast, direct access I/O (also known as concurrent I/O) means that each system is able to independently access any and all devices, without bothering any other node in the cluster. Notice that this is different from direct I/O, which simply bypasses the file systems cache. Most database systems do direct I/O both in a clustered and non-clustered environment, because they are caching the data anyway, and don't need to use the file systems cache.

Implementing direct access I/O allows a cluster file system to eliminate two out of three I/Os involved in the disk access in network I/O, because each system talks directly over the storage interconnect to the volumes. It also provides full file system transparency and cache coherency across the cluster.

Now, you may object that we could overwhelm a single disk with too many requests. Absolutely true, but this is no different from the same problem with other file systems, whether they are clustered or not. Single disks, and single database rows, are inevitably going to become bottlenecks. You design and tune around them on clusters in exactly the same way you design and tune around them on any other single member operating system, using the knowledge and tools you use now.

The I/O attributes of HP cluster offerings are summarized in the following table.

	Network I/O	Direct Access I/O	Distributed Lock Manager
LifeKeeper Linux, Windows	NFS	Oracle raw devices, GFS	Supplied by Oracle
Serviceguard	Yes	Oracle raw devices	OPS Edition
NonStop Kernel	Data Access Manager	Effectively Yes	Not applicable
TruCluster	Device Request Dispatcher	Cluster File System	Yes
OpenVMS Cluster Software	Mass Storage Control Protocol	Files-11 on ODS-2 or -5	Yes
Windows 2000 DataCenter	NTFS	Supplied by Oracle	Supplied by Oracle

Pretty much every system in the world can do client/server I/O, here called network I/O. And this makes sense, because every system in the world has the problem of sharing devices that are not on shared busses, so you need to offer this functionality.

FailSafe and Serviceguard do it with NFS or Veritas, NonStop Kernel does it with the Data Access Manager (DAM), TruCluster does it both with the Device Request Dispatcher (DRD) and the Cluster File System, OpenVMS Cluster Software does it with the Mass Storage Control Protocol (MSCP), and Windows 2000 does it with NTFS and Storage Groups.

The more interesting case is direct access I/O. One point I need to make here is that Oracle offers direct access I/O on raw devices on almost every system they support. Raw devices are not as functional as file systems, but they do exist and they do (at least with Oracle) offer direct access I/O on every major computing platform in the industry.

One of the Linux projects being done by HP and Cluster File Systems Inc for the US Department of Energy will enhance the Lustre File System originally developed at Carnegie Mellon University. It is focused on high-performance technical computing environments. Oracle has developed a cluster file system for the database files for Linux, as part of Oracle 9i Real Application Clusters 9.2. It is a layer on top of raw devices.

NonStop Kernel is interesting, because, strictly speaking, all of the I/O is network I/O, and yet because of the efficiencies and reliability of NSK, and the ability of NSK to transparently pass ownership of the volume between CPUs, it shows all of the best features of direct access I/O without the poor performance and high overhead of all other network I/O schemes. So, effectively, NSK offers direct access I/O, even though it is done using network I/O. The NonStop Kernel (effectively NonStop SQL) utilizes a "shared-nothing" data access methodology. Each processor owns a subset of disk drives whose access is controlled by processes called the Data Access Managers (DAM). The DAM controls and coordinates all access to the disk so a DLM is not needed.

Serviceguard and Windows 2000 DataCenter do not offer a direct access I/O methodology of their own, but rely on Oracle raw devices. Oracle has developed a cluster file system for Windows 2000 DataCenter for the database files, as part of Oracle 9i Real Application Clusters 9.2. It is a layer on top of raw devices.

TruCluster offers a cluster file system (CFS) which allows transparent access to any file system from any system in the cluster. However, all write operations, as well as all read operations on files smaller than 64KBytes, are done by the CFS server system upon request by the CFS client systems. In effect, all write operations and all read operations on small files are performed using network I/O. The only exception to this is applications which open the file with O_DIRECTIO.

OpenVMS Cluster Software extends the semantics of the Files-11 file system transparently into the cluster world. A file which is opened for shared access by two processes on a single system, and the same file which is opened for shared access by two processes on two different systems in a cluster, will act identically. In effect, all file operations are automatically cluster-aware.

Every operating system has a lock manager for files in a non-clustered environment. A distributed lock manager simply takes this concept and applies it between and among systems. Oracle, because they need to run in the same way across many operating environments, developed their own distributed lock manager, which is available on Linux and Windows systems.

Serviceguard includes a distributed lock manager as part of the Serviceguard Extension for RAC.

NSK does not even have the concept of a distributed lock manager. All resources (files, disk volumes, etc) are local to a specific CPU, and all communication to any of those resources is done via the standard messaging between CPUs, so any locking required is done locally to that CPU. But again, because of the efficiencies of the implementation, this scales superbly well.

TruCluster has the standard set of UNIX APIs for local locking, and a separate set of APIs that were implemented to allow applications to perform distributed locking. Applications need to be modified to use the distributed locking APIs in order to become cluster-aware.

OpenVMS Cluster Software uses the same locking APIs for all locks, and makes no distinction between local locks and remote locks. In effect, all applications are automatically cluster-aware.

Quorum

Before discussing cluster configurations, it is important to understand the concept of quorum. Quorum devices (which can be disks or systems) are a way to break the tie when two systems are equally capable of forming a cluster and mounting all of the disks, and will prevent cluster partitioning.

When a cluster is first configured, you assign each system a certain number of votes, generally 1. Each cluster environment defines a value for the number of "expected votes" that there should be for optimal performance. This is almost always the number of systems in the cluster. From there, we can calculate the "required quorum" value, which is the number of votes that are required to form a cluster. If the "actual quorum" value is below this number, the software will refuse to form a cluster, and will generally refuse to run at all.

For example, assume there are two members of the cluster, system A and system B, each with one vote, so the required quorum of this cluster is 2.

Now in a running cluster, expected votes is the sum of all of the members with which the connection manager can communicate. Since the cluster interconnect is working, there are 2 systems available and no quorum disk, so this value is 2. So actual quorum is greater than or equal to required quorum, and we have a valid cluster.

But what happens if the cluster interconnect fails? The cluster is broken, and a cluster transition occurs.

The connection manager of system A cannot communicate with system B, so actual votes becomes 1 for each of the systems. Applying the equation, actual quorum becomes 1, which is less than the number of required quorum that is needed to form a cluster, so both systems stop and refuse to continue processing.

But what would happen if one or both of the systems continue on its own? Because there is no communication between the systems, they would both try to form a single system cluster, as follows:

- System A decides to form a cluster, and mounts all of the disks.
- But system B also decides to form a cluster, and also mounts all of the disks. This is called cluster partitioning.
- The disks are mounted on two systems that cannot communicate with each other. This usually leads to instant disk corruption, as they both try to create, delete, extend and write to files without doing things like locking or cache coherency.

So how do we avoid this? A quorum device.

Same configuration as before, but here we have added a quorum disk, which is physically cabled to both systems. Each of the systems has one vote, and the quorum disk has one vote. The connection manager of system A can communicate with system B and with the quorum disk, so expected votes is 3. This means that the quorum is 2. But the cluster interconnect fails again. So what happens this time?

- Both systems attempt to form a cluster, but system A wins the race and accesses the quorum disk first. Because it cannot connect to system B, and the quorum disk watcher on system A

observes that at this moment there is no remote I/O activity on the quorum disk, system A becomes the founding member of the cluster, and writes into the quorum disk things like the system id of the founding member of the cluster, and the time that the cluster was newly formed. System A then computes the votes of all of the cluster members (itself and the quorum disk for a total of 2) and observes it has sufficient votes to form a cluster. It does so, and then mounts all of the disks on the shared bus.

- System B comes in second and accesses the quorum disk. Because it cannot connect to system A, it thinks it is the founding member of the cluster, so it checks this fact with the quorum disk, and discovers that system A is in fact the founding member of the cluster. But system B cannot communicate with system A, and as such, it cannot access the quorum disk. So system B then computes the votes of all of the cluster members (itself only, so only one vote) and observes it does not have sufficient votes to form a cluster. Depending on other settings, it may or may not continue booting, but it does not attempt to form or join the cluster. There is no partitioning of the cluster.

In this way only one of the systems will mount all of the disks in the cluster. If there are other systems in the cluster, the value of required quorum and expected quorum would be higher, but the same algorithms will allow those systems that can communicate with the founding member of the cluster to join the cluster, and those systems which cannot communicate with the founding member of the cluster to be excluded from the cluster.

Cluster Configurations

The following table summarizes important configuration characteristics of HP cluster technologies.

	Max Systems In Cluster	Cluster Interconnect	Quorum Device
LifeKeeper Linux, Windows	16	Network, Serial	Yes (Optional)
Serviceguard	16	Network, HyperFabric	Yes = 2, optional >2
NonStop Kernel	255	SystemNet, TorusNet	No
TruCluster	8 generally, 32 w/Alpha SC	100Enet, QSW, Memory Channel	Yes (Optional)
OpenVMS Cluster Software	96	CI, Network, MC, Shared Mem	Yes (Optional)
Windows 2000 DataCenter	4	Network	Yes

Linux is focusing on multi-system-view failover clusters, so FailSafe supports up to 16 systems in a cluster. These systems are connected by either the network or by serial cables. Any of the systems can take over for any of the other systems. Quorum disks are supported but not required.

Serviceguard can have up to 16 systems, using standard networking or HyperFabric as a cluster interconnect. The one special requirement here is that all cluster members must be present to initially form the cluster (100% quorum requirement), and that >50% of the cluster must be present in order to continue operation. The quorum disk or quorum system is used as a tie breaker when there is an even number of production systems and a 50/50 split is possible. For two nodes, a quorum device is required, either a disk or system. Quorum devices are optional for any other size cluster. Cluster quorum disks are supported for clusters of 2-4 nodes, and cluster quorum systems are supported for clusters of 2-16 systems.

NonStop Kernel can have up to 255 systems in the cluster, but given the way the systems interact, it is more reasonable to say that NonStop Kernel can have 255 systems * 16 processors in each system = 4,080 processors in a cluster. SystemNet is used as a communications path for each system, with TorusNet providing the cluster interconnect to the legacy K-series, and SystemNet providing a more modern cluster interconnect for the S-series, including remote datacenters. NSK does not use the quorum scheme, because it is a shared nothing environment where the quorum scheme does not make any sense.

TruCluster can have up to eight systems of any size in a cluster. For the HPTC market, the Alpha System SuperComputer system farm can have up to 32 systems. This configuration uses the Quadrix Switch (QSW) as an extremely high-speed interconnect.

OpenVMS Cluster Software supports up to 96 systems in a cluster, spread over multiple datacenters up to 500 miles apart. Each of these can also be any combination of VAX and Alpha systems, or, in 2004, any combination of Itanium and Alpha systems, for mixed architecture clusters.

TruCluster and OpenVMS Cluster Software recommend the use of quorum disks for 2 node clusters but make it optional for clusters with larger numbers of nodes.

Windows 2000 DataCenter can have up to four systems in a cluster, but keep in mind that Windows 2000 DataCenter is a services sale, and only Microsoft qualified partners like HP can configure and deliver these clusters. The only cluster interconnect available is standard LAN networking.

Application Support

The following table summarizes application support provided by the HP cluster technologies.

	Single-instance (failover mode)	Multi-instance (cluster-wide)	Recovery Methods
LifeKeeper Linux, Windows	Yes	No	Scripts
Serviceguard	Yes	No	Packages and Scripts
NonStop Kernel	Yes (takeover)	Effectively Yes	Paired Processing
TruCluster	Yes	Yes	Cluster Application Availability
OpenVMS Cluster Software	Yes	Yes	Batch /RESTART
Windows 2000 DataCenter	Yes	No	Registration, cluster API

Single-Instance and Multi-Instance Applications

When talking about applications and clusters, there are really two ways to think about it: single-instance applications and multi-instance applications. Notice that these terms are the opposite of multi-system-view and single-system-view cluster terms that we started with, but those are the terms we are stuck with.

Multi-system-view clusters allow single-instance applications, which offer failover of applications for high availability, but don't allow the same application to work on the same data on the different systems in the cluster.

Single-system-view clusters allow multi-instance applications, which offer failover for high availability, but also offer cooperative processing, where applications can interact with the same data and each other on different systems in the cluster.

A good way to determine if an application is single-instance or multi-instance is to run the application in several processes on a single system. If the applications do not interact in any way, and therefore run properly whether there is only one process running the application or every process on a single system is running the application, then the application is single-instance.

An example of a single-instance application is telnet, where multiple systems in a cluster can offer telnet services, but the different telnet sessions themselves do not interact with the same data or each other in any way. If a given system fails, the user simply logs in to the next system in the cluster and re-starts their session. This is simple failover. And this is the way that many systems including Linux and Windows 2000 DataCenter, and all of our competitors, set up NFS services, as a single instance application in failover mode.

If, on the other hand, the applications running in the multiple processes interact properly with each other, such as by sharing cache or locking data structures to allow proper coordination between the application instances, then the application is multi-instance.

An example of a multi-instance application is a cluster file system that allows multiple systems to all offer the same set of disks as services to the rest of the environment. Having one set of disks being offered by multiple systems requires a cluster file system with a single-system-view cluster, which can be offered either in the operating system software itself (TruCluster and OpenVMS Cluster Software) or by third party software (Oracle 9i Real Application Clusters). So, even though Linux FailSafe, Serviceguard and Windows 2000 get a "No" for multi-instance applications, Oracle is the exception to this. As I mentioned before, NonStop Kernel does this in a different way, but offers effectively the same functionality.

Recovery Methods

Recovery methods implement how the cluster will recover the applications that were running on a system which has been removed from the cluster, either deliberately or by a system failure. For multi-system-view clusters like Linux, this is done by scripts which are invoked when the heartbeat messages between the cluster members detects the failure of one of the systems. Two examples for Linux are 'mon' and 'monit.'

For fault tolerant systems like NonStop Kernel, recovery is done by paired processes, where a backup process is in lockstep with a primary process, ready to take over in the event of any failure.

Serviceguard has extensive tools to group applications and the resources needed to run them, into "packages" which are then managed as single units. The system administrator can define procedures to recover and re-start the application packages onto another system in a cluster in the event of server failure. This is one of the ways Serviceguard leads the industry in UNIX clustering technology.

Both TruCluster and OpenVMS Cluster multi-instance applications have built-in methods that enable recovery from failing systems. Both TruCluster and OpenVMS Cluster Software can monitor some applications and recover them automatically. TruCluster does this via the Cluster Application Availability facility, and OpenVMS Cluster Software does this by the /RESTART switch on the batch SUBMIT command.

There are three main recovery methods with Windows 2000 Datacenter:

- Generic application/generic service. This doesn't require development of any kind, simply a one-time registration of the application for protection by Windows 2000 Datacenter. There is also a wizard to guide administrators through this process step by step.
- Custom resource type. The application itself is unchanged, but the application vendor (or other party) develops a custom resource DLL that interfaces an application with Windows 2000 Datacenter to do application-specific monitoring and failover. Again, this doesn't require any development at integration time, simply registration of the application using the custom resource DLL.
- Cluster API. Here the application is modified to explicitly comprehend that it's running in a clustered environment and can perform cluster-related operations, e.g., failover, query nodes, etc.

Cluster Resilience

The following table summarizes cluster resilience characteristics of HP cluster technologies.

	Dynamic Partitions	Data High Availability	Disaster Tolerance
LifeKeeper Linux, Windows	No	Distributed Replicated Block Device (DRBD)	Extended Mirroring
Serviceguard	vPars	Multi-Path I/O (a) MirrorDisk/UX	Extended Clusters
NonStop Kernel	No	Multi-Path I/O (p), RAID-1, Process Pairs	Remote Database Facility
TruCluster	No	Multi-Path I/O (a) LSM RAID-1	StorageWorks Continuous Access
OpenVMS Cluster Software	Galaxy	Multi-Path I/O (p) HBVS RAID-1	DTCS, StorageWorks Continuous Access
Windows 2000 DataCenter	No	SecurePath (p) NTFS RAID-1	StorageWorks Continuous Access

Dynamic Partitions

One of the major selling points of clusters is high availability, even when things go wrong, such as storage subsystem failures or peak workloads beyond expectations, and even when things go very wrong, such as physical disasters. So how do clusters on HP systems cope with these things?

Dynamic partitions protect against peaks and valleys in your workload. Traditionally you built a system with the CPUs and memory for the worst-case workload, accepting the fact that this extra hardware would be unused most of the time. And with hard partitioning becoming more popular because of system consolidation, each hard partition would require enough CPUs and memory for the worst-case workload. But dynamic partitioning lets you share this extra hardware between partitions of a larger system, so that, for example, you can allocate the majority of your CPUs to the on-line processing partition during the day, and move them to the batch partition at night. Only HP-UX with vPars and OpenVMS with Galaxy offers this functionality. Both offer the ability to dynamically move CPUs between the partitions either via a GUI or the command line. Galaxy offers the ability to share physical memory that is available to two or more Galaxy partitions. HP-UX offers the capability for the Work Load Management (WLM) tools to move the CPUs in response to goal-based operational requirements.

Notice that all of the above software runs in the base operating systems, not just in the clustering products. Both vPars and Galaxy partitions can be clustered just as any other instances of the respective operating systems can be clustered.

Data High Availability

Data high availability assumes that all the proper things are being done at the storage sub-system level, such as redundant host adapters, redundant paths from the host adapters to the storage controllers (through redundant switches if you are using FibreChannel), redundant storage controllers configured for automatic failover, and the appropriate RAID levels on the disks themselves. And some of this redundancy requires cooperation from the host operating system, specifically in the area of multi-path I/O.

Multi-path I/O allows the system to have multiple physical paths from the host to a specific volume, such as multiple host adapters connected to redundant storage controllers. This is extremely common with FibreChannel, but is also achievable with SCSI.

Support for multi-path I/O can be either active or passive. Active multi-path I/O means that both paths are active at the same time, and the operating system can load balance the I/O requests between the multiple physical pathsby choosing the host adapter that is least loaded for any given I/O. In the event of a path failure (caused by the failure of the host adapter or a switch or a storage controller), the operating system would simply re-issue the I/O request to another path. This action would be transparent to the application.

Passive multi-path I/O means that only one path is active at one time, but the other path is ready to take over in the event of the first path failing. This is accomplished in the same way as the active multi-path I/O, by having the system re-issue the I/O request. The above chart shows whether the operating system supports active (a) or passive (p) multi-path I/O.

But all of these technologies are not adequate if you have multiple simultaneous disk failures, such as by physical destruction of a storage cabinet.

The first level of defense against this is host-based RAID, which performs mirroring or shadowing across multiple storage cabinets. Linux does it with Distributed Replicated Block Device (DRBD), where one of the systems writes to a local disk and then sends an update to the other system over the network so it can write a copy of that data to its local disk.

HP-UX uses MirrorDisk/UX to maintain up to three copies of the data. The software needed to enable active multi-path I/O varies depending on the storage system being used by HP-UX. PowerPath is an EMC product, which is compiled into the HP-UX kernel to give active multi-path I/O to EMC storage arrays, where the HP Logical Volume Manager (LVM) gives active multi-path I/O to the XP and EVA storage sub-systems.

NonStop Kernel provides data high availability using a combination of RAID-1 (mirrored disks), active multi-path I/O with multiple SystemNet Fabrics, multiple controller paths, etc, and process pair technology for the fault tolerant Data Access Managers (DAM).

Tru64 UNIX supports RAID-1 and RAID-5 with the Logical Storage Manager, which can protect any disk including the system root. LSM also supports active multi-path I/O.

OpenVMS supports RAID-1 with Host-Based Volume Shadowing, which can maintain up to three copies of any disk including the system disk. OpenVMS supports passive multi-path I/O, with operator controlled load balancing.

Windows 2000 DataCenter does it with NTFS mirroring. Many storage vendors, including HP, offer software with their storage offerings, which layers on top of Windows 2000 to allow passive multi-path I/O. On Windows, the software is called SecurePath.

Disaster Tolerance

Disaster tolerance is what you need to protect computer operations and data from physical disasters. In my area of Florida, we worry about hurricanes. In other areas of the world, we worry about tornadoes or blizzards. And everybody worries about earthquakes, power failures and fires. The only way to protect from these things is to make sure that your data is stored somewhere far away, and to do this in as close to real-time as possible. There are multiple kinds of data replication, but the two major ones are physical replication and logical replication.

Physical replication can be done by either the system or the storage sub-system. The systems use the same software that is used for data high availability detailed above, except that the second (or in some cases, the third) copy of the data is in another physical location. Serviceguard uses MirrorDisk/UX, NonStop Kernel uses the Remote DataCenter Facility (RDF), TruCluster uses LSM and OpenVMS Cluster Software uses host-based Volume Shadowing. The rules for accessibility of the remote volumes by the remote systems are the same as if the remote volumes and the systems were in the same room as the source systems and volumes. Only TruCluster and OpenVMS Cluster Software can share volumes between systems, whether local or remote. Having the operating system accessing volumes across a wide area FibreChannel system is called an “extended cluster”.

We have the same situation here as we had for RAID, in that the storage sub-system offers significant capabilities for physical replication, regardless of the host operating system or clustering software capabilities. StorageWorks Continuous Access for the EVA and the XP storage arrays support the clusters of HP environments and most competitive UNIX systems.

Both the system-based data high availability software and the storage-based Continuous Access offers active/active bi-directional data replication. Exactly what does that mean?

Assume you have a production system in Los Angeles, which is working fine. You want to safeguard your data, so you decide to build a disaster recovery site in San Bernardino, about 100km (60 miles) away from Los Angeles.

First, you put in a group of FibreChannel switches, and connect them using the FibreChannel to ATM adapters to the FibreChannel switches in your Los Angeles data center. Then you put a duplicate set of storage in San Bernardino, and begin replicating the production system’s data from Los Angeles to San Bernardino. This is known as active/passive replication, because San Bernardino is simply a data sink: no processing is going on there, mostly because there are no systems on that side.

This works well, but as times goes on you need more than this: you need processing to continue in San Bernardino even if your Los Angeles data center is wiped out. So you put some systems in San

Bernardino, and physically cable them to the storage. But the target volume of the Continuous Access copy is not available for mounting by the systems, no matter what operating system they are running, so the San Bernardino site is idle, simply waiting for a signal to take over the operations from Los Angeles. This signal can be automated or manual, but in either case, the storage sub-system would break the Continuous Access link, the systems would mount the volumes, and would then initiate any recovery mechanisms that have been defined for the application.

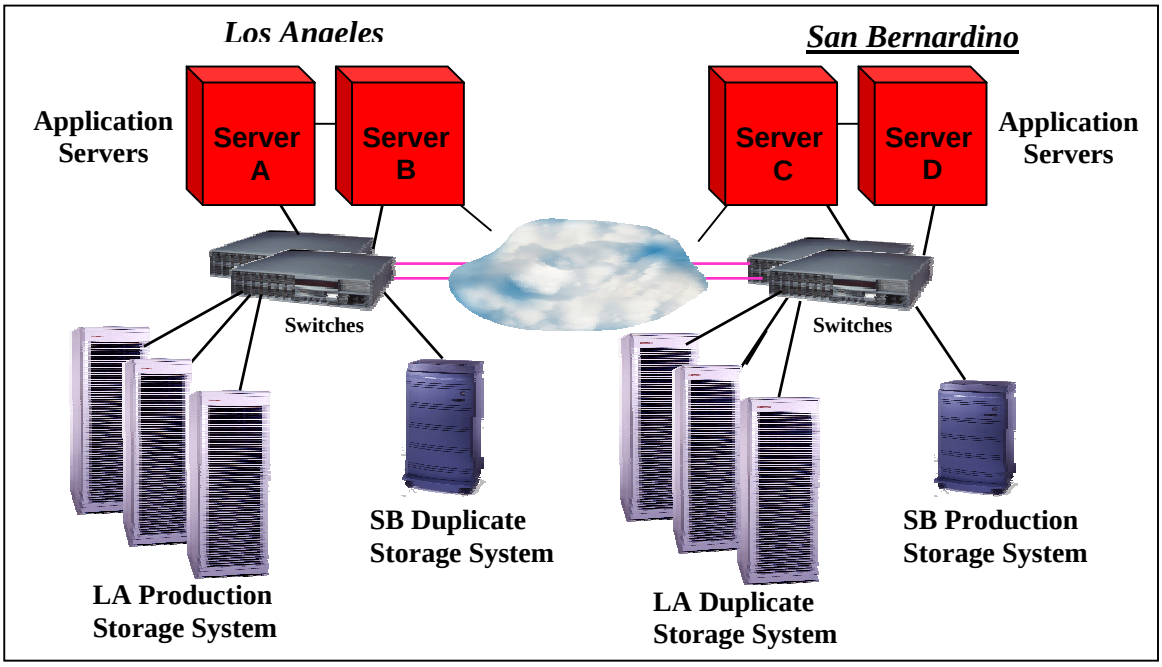
But your CFO strenuously objects to having a group of systems in San Bernardino just sitting idle, so you split your workload and give half of it to Los Angeles and half of it to San Bernardino. Notice that this is a multi-system-view implementation, because the target volume of a Continuous Access copy is not available for mounting by the systems. So, just as you duplicated the Los Angeles storage in San Bernardino, now you duplicate the San Bernardino storage in Los Angeles, which you then connect to the systems in Los Angeles as well, and you setup a Continuous Access copy from San Bernardino to Los Angeles.

So now you have your Los Angeles production data being replicated to San Bernardino, and your San Bernardino production data being replicated to Los Angeles. You could survive the loss of either data center, and have all of your data in the other one.

This is known as active/active bi-directional data replication. Even though each set of storage is only being replicated in one direction, your business has data being replicated across the enterprise.

Notice that the replication is being done by the FibreChannel. The hosts don't know or care that it is happening.

Failover requires you to explicitly stop the replication process in the surviving datacenter, and then explicitly mount the storage subsystems on the systems in the surviving datacenter, to get back into production. Failback requires the same type of operation, where you have to synchronize the data between the two storage subsystems, and then place one of them back into source mode and the other into target mode, in order to restore the standard configuration.



Notice that in both cases, system-based data high availability and StorageWorks Continuous Access, the data being replicated is actually being written multiple times, whether by the system or

by the storage controller. And all of the data on the source disk is being written to the target disk, whether it is mission-critical database files or temporary files in a scratch area. This requires careful planning to ensure that you are replicating everything you need (such as the startup scripts for the database application, which exists outside the database files and is probably not on the same disk volumes as the database files), but not too much (such as /tmp).

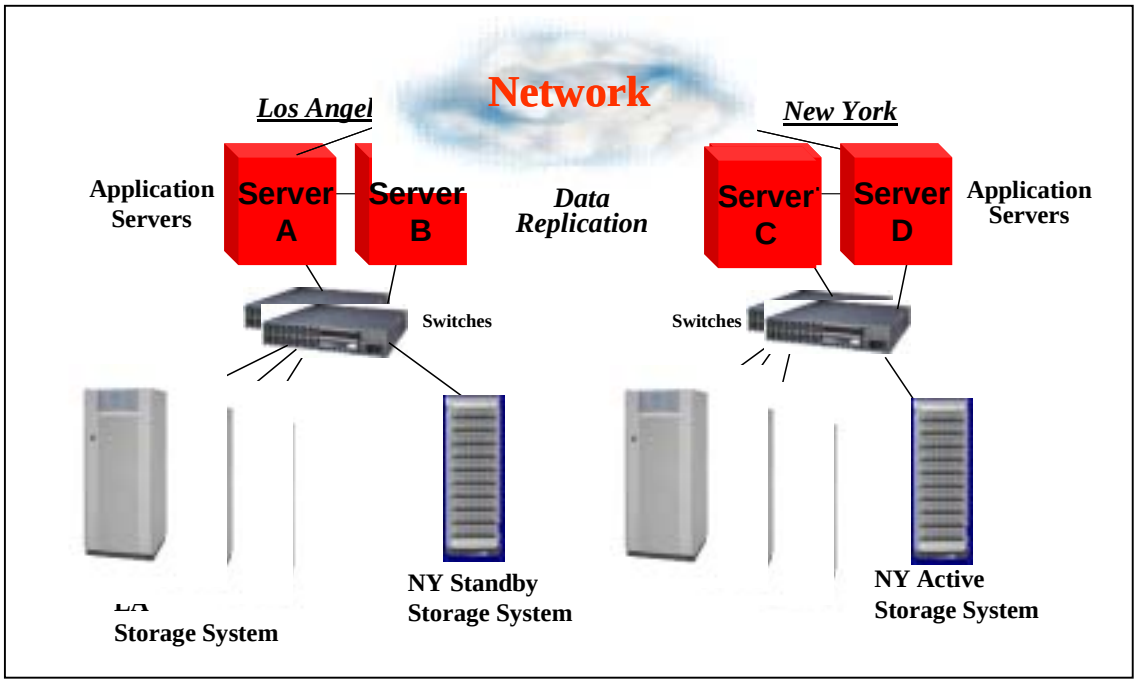
So how is this different from logical replication? Well, the biggest difference is what is being replicated and how the replication is being done.

Logical replication ignores the disk volumes and replicates the transactions themselves. In the same way that physical replication takes a single write operation and applies it to multiple disk volumes, logical replication takes a single update transaction and applies it to multiple databases. The communications can be done over standard networking, and the systems that operate the multiple databases may or may not be clustered, depending on the database software chosen.

Once again, you have your data center in Los Angeles, and it is working fine. Once again you decide you need a disaster recovery site. But in this case, you can put that disaster recovery site anywhere you want, because we are using standard networking technology. So you choose New York for your disaster recovery site.

You put a duplicate of your data center in New York, and then connect them with a fast networking link. Notice that this is a fully functional duplication of the data center, as it requires both systems and storage. However, it is not required to be a physical duplication: if you only require that some data is replicated, or if you can accept some lesser performance when a failover occurs, you can have fewer computing resources in New York than in Los Angeles. Then you use logical replication to replicate the database records.

You can either leave it like this, in active/standby mode, or you can have the New York site start to run some type of production. Notice that it has to be different from the production run in Los Angeles, because these are two different systems or clusters and cannot see the same data. But in the same way we had bi-directional physical replication, you can have bi-directional logical replication. This provides both data protection and failover capabilities from New York to Los Angeles. This is an active/active logical replication scheme. But keep in mind, just like in the previous example, failover and failback are semi-automated processes, with some human intervention required.



Keep in mind what is being replicated here. In the previous example of physical replication, the storage subsystem was taking whatever bits were written to the disk and copying them across to the other side. The storage subsystem doesn't have a clue about file systems, files, database records, re-do logs or transactions. The advantage of this is that *everything* is being replicated: database files, log files, flat files, everything. But the disadvantage is that a simple operator error (like "rm –r *" or "DELETE [*...]*.*;*" will be replicated to the target disk in real time.

But logical replication is replicating database transactions, not volumes. It does not replicate things like security information, patches to the operating system or applications, scripts, or any of the myriad of other files which are needed to maintain the site, all of which have to be replicated and maintained by hand. But the advantage is that an operator error that works on files cannot destroy your entire environment.

One last point about the differences between physical and logical replication. Physical replication does not understand the underlying structure of the data that it works on: it understands which disk blocks have changed, but not which files those blocks belong to or whether the disk block that was changed was a database row or a database index. Therefore, it is impossible to ensure atomicity of a transaction with physical replication. That is, there is a window of time where the write operation of (for example) the row information was successfully replicated but the index information has not yet been replicated. If the systems that originated the transaction fail, or the communications link fails at that moment, the replicated database is probably corrupt.

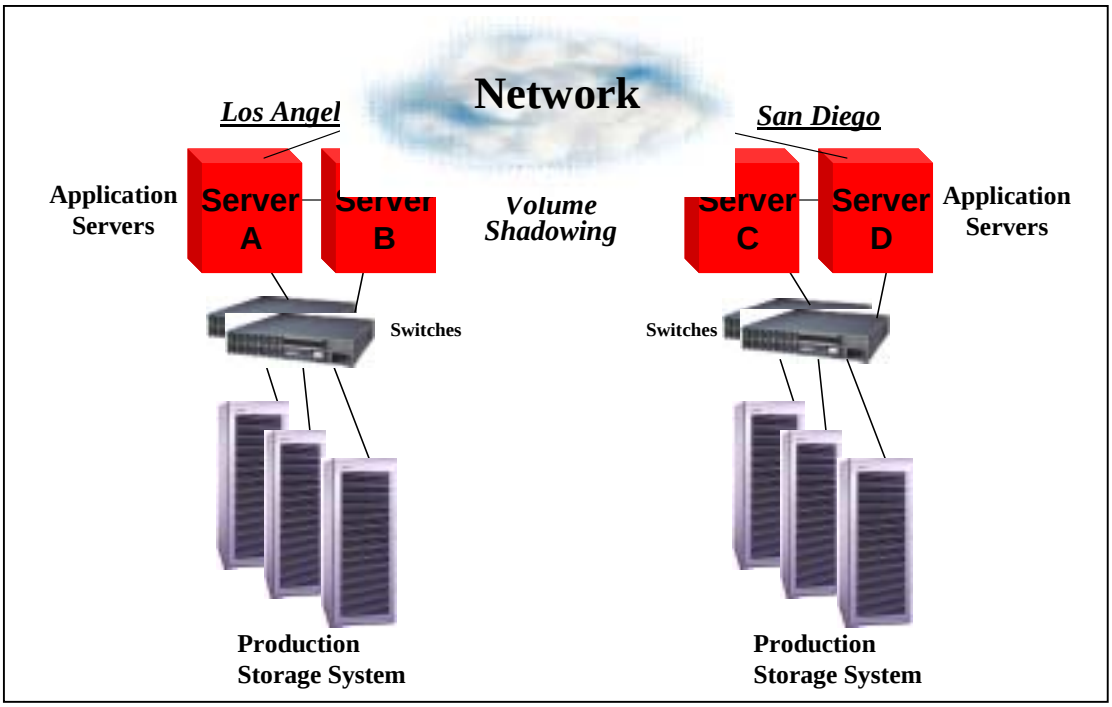
So, how is this different with OpenVMS Cluster Software and Disaster Tolerant Cluster Services (DTCS)? Well, one difference is where the physical replication is being done, but a more important difference is that the target volumes are fully accessible from any member of the wide area cluster.

Once again, you have your data center in Los Angeles, and it is working fine. Once again you decide you need a disaster recovery site. But in this case, you can put that disaster recovery site up to 800 km (500 miles) away, because we are using standard networking technology. So you choose San Diego for your disaster recovery site.

You put a duplicate of your data center down in San Diego, and then connect the two sites with an ATM fabric. There are lots of rules around this concerning latency and throughput, and quorum schemes get really interesting, so I really recommend you allow HP to help you design and implement it. But we have lots of sites working with this; it isn't that difficult.

Then you use host-based Volume Shadowing for OpenVMS to replicate the data. Because HBVS is host-based, all of the systems can mount the volumes directly, with full read/write access to the drives using direct access I/O. Any writes made by any of the systems will be written to the storage systems in both Los Angeles and San Diego, and all other systems will be aware of them with full cache coherency. Any reads will come from the local disks, so it is very efficient.

There are no standby systems here and no passive storage arrays. We recommend that the ATM fabric be redundant, so you are protected even in the event of network failure. But even if one of the sites was completely wiped out, the other site would recover very quickly, because the other systems already have the disks mounted and operational. It would simply be a cluster transition, and you are back in business.



So, why is there a distance limitation for this scenario, but not for extended clusters or storage based Continuous Access? Because this scenario requires a lot of two-way communication between the sites, and because of the speed of light.

Light travels in a vacuum at 186,282 miles per second. Let's round that off to 200,000 miles per second to make the math easy. The speed of 200,000 miles per second means light travels 200 miles per milli-second. We are worried about round-trip distances, so light can travel up to 100 miles away and back in 1 milli-second. But light travels somewhat slower in fibre than in a vacuum, and there are the inevitable switch delays, so the rule of thumb is that it takes1 milli-second for every 50-mile round trip. So 500 miles adds 10*1 = 10 milli-seconds to the latency of a disk access. Given normal disk access latency of 10-15 milli-seconds, this merely doubles the latency, and the OpenVMS Host-Based Volume Shadowing software can cope with that. But if you get much more than that, the software might think the disk at the other end has gone off-line, and the software will incorrectly break the shadow set. If we increase the timeout feature to get around this, the software will think the disk is just being slow when it really is inaccessible, and we will have unacceptable delays in breaking the shadow set when it needs to be broken.

In the physical and logical replication scenarios, each volume on one side is merely sending data to the other side, and eventually getting an acknowledgement back. The acknowledgement is not overly time critical, and you can keep sending new data while waiting for the acknowledgement of the data you have already sent. The other side cannot write data to the target disk, so there is no conflict resolution necessary. So the additional latency of the speed of light is not as important, so the distance limitations are almost non-existent, as long as you don't exceed the distance limitations of your interconnect technology.

Summary

For obvious reasons, every operating system offers a high availability option. But their capabilities vary widely: some systems offer 2-nodes with manual failover measured in minutes, other systems

offer 16 nodes with automated failover time measured in seconds, while still others offer hundreds or thousands of processing units with absolutely transparent recovery from failure. And each system knows how to protect itself in this increasingly insecure world: some systems do it by depending on FibreChannel replication, others do it by depending on a database to move the data around the country, and others offer true active/active multi-site clusters over hundreds of miles.

It is up to you as technologists to understand these technologies, and to pick the right one for the business task. But you have an additional job that you might not think of: letting your senior management know the exact capabilities of the technology you chose. Because if you implemented a 2-node, manual failover system with no disaster tolerance, and your management thinks you have implemented an infinitely expandable fault tolerant system with zero recovery time even if the primary datacenter is completely and unexpectedly wiped out, you have a problem. And you will only discover you have a problem when the system behaves exactly as you implemented it, and management discovers that it didn't do what they thought it would do.

So the answer is to document exactly what your chosen solution will and won't do, and get full agreement from your senior management that this is what they want it to do. And if they come back and say they need it to do more, then you request more money to get it to do what they need. In either scenario, HP is available to provide the software and services to help you design and implement the chosen solution.

Acknowledgements

I wish to thank Kirk Bresniker, Dan Cox, Jon Harms, Keith Parris, Bob Sauers and Wesley Sawyer for their invaluable input and review of this material.

For More Information

On LifeKeeper for Linux

- http://www.hp.com/linux for general information on HP servers and Linux
- http://h18000.www1.hp.com/solutions/enterprise/highavailability/linux/index.html for general information on HP servers and Linux and high availability solutions
- http://www.hp.com/hpinfo/newsroom/press/08aug02b.htm for information on Lustre
- http://www.compaq.com/Solutions/enterprise/highavailability/linux/description.html#specs for specifications on LifeKeeper with ProLiant servers
- http://linux-ha.org
    - "What Linux-HA Can Do Now"
    - "LAN Mirroring Technologies"
- http://www.missioncriticallinux.com/technology/cluster/ for the white paper on Mission Critical Linux
- http://technet.oracle.com/tech/linux/ for information on Oracle and Linux
- http://www.steeleye.com/products/linux/#2 and http://www.steeleye.com/pdf/literature/lkpr4linux.pdf for information on SteelEye LifeKeeper
- http://www.kernel.org/software/mon/ for information on the Service Monitoring Daemon
- http://www.tildeslash.com/monit/ for information on the Monit Utility

On Serviceguard for HP-UX and Linux

- http://www.hp.com/go/ha for general information on Serviceguard and high availability
- http://www.hp.com/products1/unix/operating/infolibrary/reports/2002Unix_report.pdf for the DH Brown 2002 UNIX Function Review report
- http://docs.hp.com/hpux/onlinedocs/ha/highly_avail_clust.pdf for Disaster Tolerant and Highly Available Cluster Technologies
- http://www.hp.com/products1/unix/highavailability/ar/mcserviceguard/infolibrary/index.html, Information Library
    - 5nines Architecture Overview
    - System Cluster Technologies and Disaster Tolerance
    - Data Protection
- http://www.software.hp.com/cgi-bin/swdepot_parser.cgi/cgi/displayProductInfo.pl?productNumber=B2491BA for MirrorDisk/UX

On NonStop Kernel

- http://nonstop.compaq.com/view.asp?PAGE=TIM_Prod for access to the Total Information Manager (TIM) product, for full NSK information
- http://h71033.www7.hp.com/object/tdnsk3pd.html for details on NSK and high availability
- http://h71033.www7.hp.com/page/RDF_SW.html for information on Remote DataCenter Facility

On TruCluster

- http://h30097.www3.hp.com/ for general information on Tru64 UNIX and TruCluster
- http://h18000.www1.hp.com/products/quickspecs/11444_div/11444_div.HTML for the QuickSpecs on TruCluster V5.1b
- http://h30097.www3.hp.com/docs/pub_page/cluster51B_list.html for the documentation. Specifically:
    - http://h30097.www3.hp.com/docs/base_doc/DOCUMENTATION/V51B_HTML/ARHGVETE/TITLE.HTM , Cluster Technical Overview
    - http://h30097.www3.hp.com/docs/base_doc/DOCUMENTATION/V51B_HTML/ARHGVETE/TITLE.HTM, Cluster Technical Overview, Section 2.2
    - http://h30097.www3.hp.com/docs/base_doc/DOCUMENTATION/V51B_HTML/ARHGVETE/TITLE.HTM, Cluster Technical Overview, Section 3
    - http://h30097.www3.hp.com/docs/base_doc/DOCUMENTATION/V51B_HTML/ARHGWETE/TITLE.HTM, Hardware Configuration, Section 1.3.1.4
    - http://h30097.www3.hp.com/docs/base_doc/DOCUMENTATION/V51B_HTML/ARHGYETE/TITLE.HTM, Cluster Administration, Section 4.3, Calculating Cluster Quorum
    - http://h30097.www3.hp.com/docs/base_doc/DOCUMENTATION/V51B_HTML/ARHH0ETE/TITLE.HTM, Highly Available Applications, Chapter 1, Cluster Applications

- http://h18000.www1.hp.com/products/quickspecs/10899_div/10899_div.HTML for the QuickSpecs for the Logical Storage Manager V5.1b
- http://www.hp.com/techservers/ for the AlphaServer SC home page

On OpenVMS Cluster Software

- http://h71000.www7.hp.com/ for general information on OpenVMS and OpenVMS Cluster Software
- http://h71000.www7.hp.com/openvms/products/clusters/index.html for information on OpenVMS Cluster Software V7.3-1
- http://h18000.www1.hp.com/info/SP2978/SP2978PF.PDF for OpenVMS Cluster Software V7.3-1 SPD
- http://h71000.www7.hp.com/doc/index.html for the documentation. Specifically:
  - http://h71000.www7.hp.com/doc/731FINAL/4477/4477PRO.HTML, OpenVMS Cluster Systems, Section 2.3.2 shows quorum algorithm, and Chapter 7, Setting Up and Managing Cluster Queues
  - http://h71000.www7.hp.com/doc/731FINAL/6318/6318PRO.HTML, Guidelines for OpenVMS Cluster Configurations, Chapter 8, Configuring OpenVMS Clusters for High Availability and Appendix D, Multi-Site OpenVMS Clusters
  - http://h71000.www7.hp.com/doc/731FINAL/5423/5423PRO.HTML, Volume Shadowing for OpenVMS, Section 1.5 discusses WAN based RAID-1 for disaster tolerance

On Windows 2000

- http://www.microsoft.com/windows2000/en/datacenter for general Windows 2000 DataCenter information
- http://www.microsoft.com/windows2000/en/datacenter/help/ for the documentation. Specifically:
  - Choosing a Cluster Model, emphasizes that Windows 2000 DataCenter is a multi-system-view cluster, and acknowledges the lack of single-system-view capabilities.
  - Checklist: Preparing a Server Cluster, states that each system in the cluster must have its own system disk.
  - Server Clusters says up to 4 systems in a server cluster
  - Cluster Hardware and Drivers says the network is the only cluster interconnect
  - Quorum Disk describes the use of the quorum disk, and Cluster Database discusses how the cluster database from each system is written to the recovery log on the quorum disk
  - Windows Clustering, Server Clusters, How To…, Perform Advanced Administrative Tasks
  - Choosing a RAID Method
  - Disaster Protection discussing the lack of WAN RAID

On Single System Image Linux

- http://sourceforge.net/projects/ssic-linux for information on this HP project

On StorageWorks Continuous Access

- http://h18006.www1.hp.com/storage/software.html for general information on StorageWorks high availability solutions
- http://www.compaq.com/products/quickspecs/10281_na/10281_na.html, QuickSpecs for StorageWorks Continuous Access
- http://h18006.www1.hp.com/products/storage/software/conaccesseva/index.html, SANworks Continuous Access Overview and Features

Books

- "Clusters for High Availability", Peter Weygant, ISBN 0-13-089355-2
- "In Search of Clusters", Gregory F. Pfister, ISBN 0-13-899709-8

Local Area Network Cluster Interconnect Monitoring

Keith Parris
Systems/Software Engineer, HP Multivendor Systems Engineering

Overview

Local Area Network (LAN) technology is increasingly replacing older technologies such as Computer Interconnect (CI) and Digital Storage Systems Interconnect (DSSI) as the interconnect used for communications within the majority of OpenVMS Cluster configurations. This article describes the LAVC$FAILURE_ANALYSIS tool which is supplied with OpenVMS and which can be used to monitor and troubleshoot LANs that are used as cluster interconnects.

LAN Cluster Interconnect Monitoring using LAVC$FAILURE_ANALYSIS

Background

When support for OpenVMS Cluster communications over a Local Area Network (LAN) was first introduced, the resulting configuration was known as a Local Area VAX Cluster, or "LAVC" for short.

Support for multiple LAN adapters in a single system was introduced in VAX/VMS version 5.4-3. This allowed LAVC configurations with a significant amount of LAN redundancy to be configured, and allowed the cluster to continue operating and survive the failure of network adapters and various other network components.

One challenge that is always present when redundancy is configured is the need to monitor the redundant components to detect failures, or else, as components continue to fail over time, the entire system may fail when the last remaining working component fails.

To assist with this problem, supplied along with OpenVMS is a tool called LAVC$FAILURE_ANALYSIS. Its purpose is to monitor and report any LAN component failures. It reports these using Operator Communication (OPCOM) messages. It also reports when a failure is repaired.

Implementing LAVC$FAILURE_ANALYSIS

There is a template program for LAVC$FAILURE_ANALYSIS found in the SYS$EXAMPLES: directory on an OpenVMS system disk. The template program is called LAVC$FAILURE_ANALYSIS.MAR. The template program is written in Macro-32 assembly language, but you don't need to know how to program in Macro-32 just to use it.

To use the LAVC$FAILURE_ANALSYS facility, the program must be:

1. Edited to insert site-specific information
2. Compiled (on Alpha; assembled on VAX)
3. Linked, and
4. Run on each node in the cluster (preferably at boot time)

Maintaining LAVC$FAILURE_ANALYSIS

The program must be re-edited and rebuilt whenever:

1. The LAVC LAN is reconfigured
2. A node's MAC address changes (for example, when an HP Customer Services representative replaces a LAN adapter)
3. A node is added or removed (permanently) from the cluster

How Failure Analysis Is Done

PEDRIVER, the Port Emulator code which makes a LAN look and act like a CI (Computer Interconnect) port to the OpenVMS Cluster code, transmits multicast "Hello" packets every 2-3 seconds or so from each LAN adapter that is enabled for cluster communications. These "Hello" packets are sent to a multicast address that is associated with the cluster group number, and which has a MAC address of the form AB-00-04-01-xx-yy, where "xx-yy" is based on the cluster group number plus an offset. Each cluster member enables receipt of packets addressed to the MAC address associated with its own cluster group number, and uses the receipt of Hello packets to discover new communications paths and track the reachability of a node via a given path.

In the information added by editing the LAVC$FAILURE_ANALYSIS program to customize it for a given cluster, OpenVMS is told what the LAN configuration should be (in the absence of any failures). The LAN configuration is represented as a mathematical "graph" with "nodes" and "connections" between the nodes in the graph. From this information, OpenVMS infers which LAN adapters should be able to "hear" Hello packets from which other LAN adapters. By checking for the receipt of Hello packets as expected, OpenVMS can thus determine if a path is working or not.

By analyzing Hello packet receipt patterns and correlating them with the mathematical graph of the network, OpenVMS can tell which nodes in the mathematical network graph are passing Hello packets and which appear to be blocking Hello packets. OpenVMS determines a Primary Suspect (and, if there is any ambiguity as to exactly what specific component has failed because more than one failure scenario might cause the observed symptoms, also identifies one or more Alternate Suspects), and reports these via OPCOM messages with a "%LAVC" prefix.

Primary Suspects are reported with a message prefix of the form "%LAVC-W-PSUSPECT". Alternate Suspects are reported with a message prefix of the form "%LAVC-I-ASUSPECT". When the failure that caused a Suspect to be reported is repaired, a message with a prefix of the form "%LAVC-S-WORKING" is generated.

Getting Failures Fixed

Since notification is done via OPCOM messages, someone or something needs to be scanning OPCOM output and taking action. If no human is actively watching the OPCOM output, the error notification may be overlooked.

In one disaster-tolerant cluster there were two expensive DS-3 inter-site links configured, and LAVC$FAILURE_ANALYSIS was put into place to monitor the inter-site links, but the OPCOM message reporting a failure of one of the links one day was missed and the failure was not discovered until six days later. In the intervening time, a failure of the other link would have caused a loss of communications between the two sites of the disaster-tolerant cluster, which would likely have been noticed quickly, but would not have been very convenient.

Products such as TecSys Development Inc.'s ConsoleWorks, Computer Associates' Unicenter Console Management for OpenVMS (previously known as Console Manager), Ki Networks' Command Line Interface Manager (CLIM), or Heroix RoboMon can scan for the %LAVC messages generated by LAVC$FAILURE_ANALYSIS and take some appropriate action (sending e-mail, sending a notification via pager, etc.).

Gathering Information

To implement LAVC$FAILURE_ANALYSIS, data must be gathered about the LAN configuration. The data required includes:

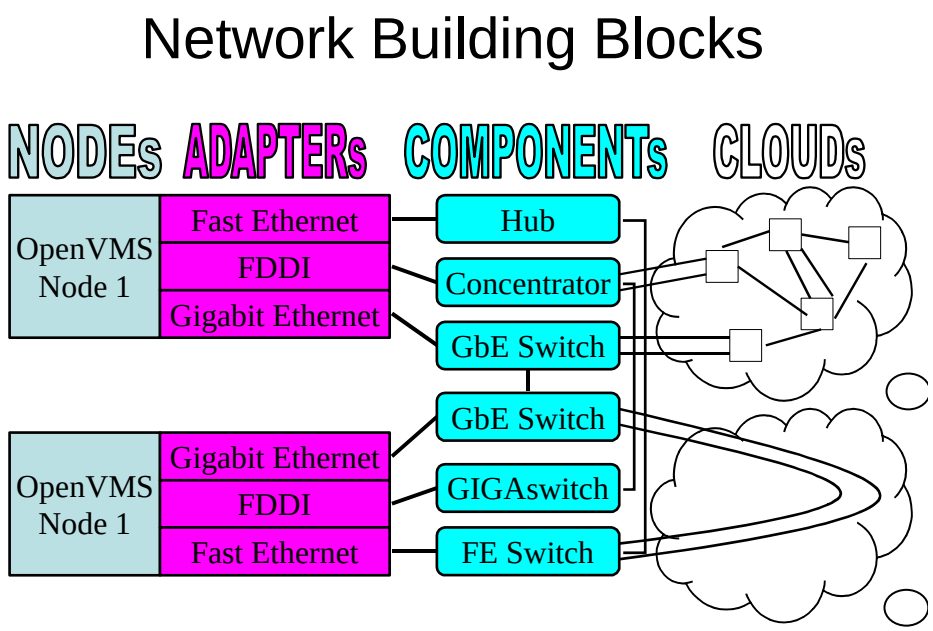
- OpenVMS nodes, and the LAN adapters in each node
- Hubs, switches, bridges, and bridge/routers (routers which have bridging enabled)
- Links between all of the above

Network Building Blocks

For the purposes of LAVC$FAILURE_ANALYSIS, OpenVMS considers LAN building blocks as being divided into 4 classes:

- NODE: An OpenVMS system
- ADAPTER: LAN adapters or Network Interface Cards (NICs) in each OpenVMS system
- COMPONENT: Hubs, switches, bridges, bridge-routers
- CLOUD: Combinations of components that can't be diagnosed directly

These relationships are illustrated in the following diagram:



Handling Network Loops

The algorithm used for LAVC$FAILURE_ANALYSIS can't deal with loops in the mathematical network graph. Yet redundancy is often configured among LAN components (and this is a good thing). The bridges' Spanning Tree algorithm automatically shuts off redundant (backup) links unless and until a failure occurs. But Hello packets don't get through these backup links, so LAVC$FAILURE_ANALYSIS can't track them directly.

For these cases, you replace the redundant portion of the network with a "network cloud" that includes all of the redundant components. Then OpenVMS can determine if the network "cloud" as a whole is functioning or not, and doesn't have to worry about the internal details of which links are turned on or off by the Spanning Tree protocol at any given time.

Handling Multiple LANs

Note that multiple, completely separate LANs don't count as "loops" in the network, and OpenVMS can track each one separately, independently, and simultaneously.

Gathering Information

Let's look a bit more closely at the data required for LAVC$FAILURE_ANALYSIS:

- Node names and descriptions
- LAN adapter types and descriptions, and their:
    - Media Access Control (MAC) address (e.g., 08-00-2b-xx-xx-xx, 00-00-F8-xx-xx-xx)
    - plus DECnet-style MAC address for Phase IV (e.g., AA-00-04-00-yy-zz)
- Network components and descriptions
- Interconnections between all of the above

The names and descriptions supplied will be used in the OPCOM messages which are generated upon detection of failures and when failures are repaired.

Getting MAC address info

A DCL command procedure similar to the following can be used to help sift through the output from SDA> SHOW LAN/FULL and pick out the device names and MAC address data.

```
$! SHOWLAN.COM
$!
$     write sys$output "Node ",f$getsyi("nodename")
$     temp_file := showlan_temp.temp_file
$     call showlan/out='temp_file'
$     search 'temp_file' "(SCA)","Hardware Address" –
            /out='temp_file'-1
$     delete 'temp_file';*
$     search/window=(0,1) 'temp_file'-1 "(SCA)"
$     delete 'temp_file'-1;*
$     exit
$!
$ showlan: subroutine
$      analyze/system
show lan/full
exit
$     endsubroutine
```

Editing the Template Program

Once the necessary data has been gathered, you will need to edit a copy of the LAVC$FAILURE_ANALYSIS.MAR program found in the SYS$EXAMPLES: directory.

There are five sections to edit, as follows:

Edit 1

In Edit 1, you can give descriptive names to nodes, adapters, components, and clouds. These names become names of some macros. Later in the code, you will create invocations of these macros, calling them by the names defined here.

These macro names are for your convenience and reference inside the program, and do not appear external to the program.

Edit 1 example, taken from the template LAVC$FAILURE_ANALYSIS.MAR program:

```
;     Edit 1.
;
;           Define the hardware components needed to describe
;           the physical configuration.
;

      NEW_COMPONENT     SYSTEM      NODE
      NEW_COMPONENT     LAN_ADP     ADAPTER
      NEW_COMPONENT     DEMPR       COMPONENT
      NEW_COMPONENT     DELNI       COMPONENT
      NEW_COMPONENT     SEGMENT     COMPONENT
      NEW_COMPONENT     NET_CLOUD   CLOUD
```

Not very many networks today contain DELNI and DEMPR boxes or thickwire Ethernet cables, but you might create names like SWITCH, HUB, CISCO, DS3_LINK, and so forth here. Note that you can create multiple names for a given type of network building block, as we see done here with COMPONENTs.

Edit 2

In Edit 2, you create an "ASCII art" drawing to document the LAVC LAN configuration. This has no functional effect on the code, but helps you (and others who follow you) understand the information in the sections of data that follow.

In the drawing, you choose brief abbreviated names for each network building block (Node, Adapter, Component, or Cloud). These abbreviated names are only used within the program, and do not appear externally.

Edit 2 example:

```
;    Edit 2.
;
;              Diagram of a multi-adapter LAV cluster.
;
;
;   Sa   -----+---------------+---------------+-------------+-----   Sa
;             |               |               |             |
;             |             MPR_A             |             |
;             |          .----+----.          |             |
;             |         1|        1|         1|             |
;           BrA       ALPHA     BETA       DELTA          BrB
;             |         2|        2|         2|             |
;             |          `----+----'          |             |
;             |             LNI_A             |             |
;             |               |               |             |
;   Sb   -----+---------------+---------------+-------------+-----   Sb
```

Edit 3

In Edit 3, you name and provide a text description for each system and its LAN adapter(s), and the MAC address of each adapter. The name and text description will appear in OPCOM messages indicating when failure or repair has occurred. The MAC address is used to identify the origin of Hello messages.

For DECnet Phase IV, which changes the MAC address on all LAN adapters (which it calls Circuits) that it knows about from the default hardware address to a special DECnet address when it starts up, you provide both:

- The hardware MAC address (e.g., 08-00-2B-nn-nn-nn), and
- The DECnet-style MAC address, which is derived from the DECnet address of the node (AA-00-04-00-yy-xx)

This way, LAVC$FAILURE_ANALYSIS can work both before and after DECnet Phase IV starts up and changes the MAC address.

DECnet Phase V (DECnet/OSI) does not change the MAC address, so only the hardware address is needed.

Edit 3 example:

```
;      Edit 3.
;
;              Label Node                    Description                       LAN HW Addr          DECnet Addr
;              ----- -----   ----------------------------------------   --------------------  --------------------

      SYSTEM   A,    ALPHA,  < - MicroVAX II; In the Computer room>
      LAN_ADP  A1,   ,       <XQA; ALPHA - MicroVAX II; Computer room>,   <08-00-2B-41-41-01>,  <AA-00-04-00-01-04>
      LAN_ADP  A2,   ,       <XQB; ALPHA - MicroVAX II; Computer room>,   <08-00-2B-41-41-02

      SYSTEM   B,    BETA,   < - MicroVAX 3500; In the Computer room>
      LAN_ADP  B1,    ,      <XQA; BETA - MicroVAX 3500; Computer room>,  <08-00-2B-42-42-01>,  <AA-00-04-00-02-04>
      LAN_ADP  B2,    ,      <XQB; BETA - MicroVAX 3500; Computer room>,  <08-00-2B-42-42-02>

      SYSTEM   D,    DELTA,  < - VAXstation II; In Dan's office>
      LAN_ADP  D1,    ,      <XQA; DELTA - VAXstation II; Dan's office>,  <08-00-2B-44-44-01>,  <AA-00-04-00-04-04
      LAN_ADP  D2,    ,      <XQB; DELTA - VAXstation II; Dan's office>,  <08-00-2B-44-44-02>
```

Note that only NODE building blocks (named SYSTEM in this example) have a node name; all the other building block types have a null parameter in that location (indicated by a comma alone).

Edit 4

In Edit 4, you name and provide a text description for each Component and each Cloud. As with the nodes and adapters you described in Edit 2, the name and text description you provide here will appear in OPCOM messages indicating when failure or repair has occurred.

Edit 4 example:

```
;      Edit 4.
;
;               Label each of the other network components.
;

      DEMPR     MPR_A, , <Connected to segment A; In the Computer room>
      DELNI     LNI_A, , <Connected to segment B; In the Computer room>

      SEGMENT   Sa,  , <Ethernet segment A>
      SEGMENT   Sb,  , <Ethernet segment B>

      NET_CLOUD BRIDGES, , <Bridging between Ethernet segments A and B>
```

Here, you give an abbreviated symbolic name and a description for each of the COMPONENT and CLOUD building blocks. Again, the node name argument is null, indicated by a comma alone, for the second argument to the macros.

Edit 5

In Edit 5, you indicate which network building blocks have connections to each other. This is a list of pairs of building blocks, indicating that the two components in a given pair are connected together.

Pairs of components which have no direct connection between them are simply not listed here.

Edit 5 example:

```
;     Edit 5.
;
;          Describe the network connections.
;

      CONNECTION  Sa,   MPR_A
      CONNECTION        MPR_A,    A1
      CONNECTION                  A1,     A
      CONNECTION        MPR_A,    B1
      CONNECTION                  B1,     B

      CONNECTION  Sa,             D1
      CONNECTION                  D1,     D

      CONNECTION  Sa,   BRIDGES
      CONNECTION  Sb,   BRIDGES

      CONNECTION  Sb,   LNI_A
      CONNECTION        LNI_A,    A2
      CONNECTION                  A2,     A
      CONNECTION        LNI_A,    B2
      CONNECTION                  B2,     B

      CONNECTION  Sb,             D2
      CONNECTION                  D2,     D
```

In this example, the author started with network segments, and indicated each component and LAN adapter that was connected to that segment. Note that it is necessary to explicitly indicate which LAN adapters are connected to a given node, even though that might be implied in the naming convention and thus perfectly obvious to a human being.

The formatting of columns adds to the readability here, with network segments, LAN adapters and nodes in their own columns.

My personal preference is to begin with the nodes at the left, adapters in the next column to the right, and so forth, moving outward from a node to the things to which it is connected. But the order of appearance here, either left to right or top to bottom, does not affect the functional behavior. Only which building blocks are connected in a pair is significant.

Handling MAC Address Duplications with Multiple LANs

If you are running DECnet Phase IV and have multiple independent LANs, you may end up with multiple LAN adapters on a single node with the same MAC address, connected to different LANs.

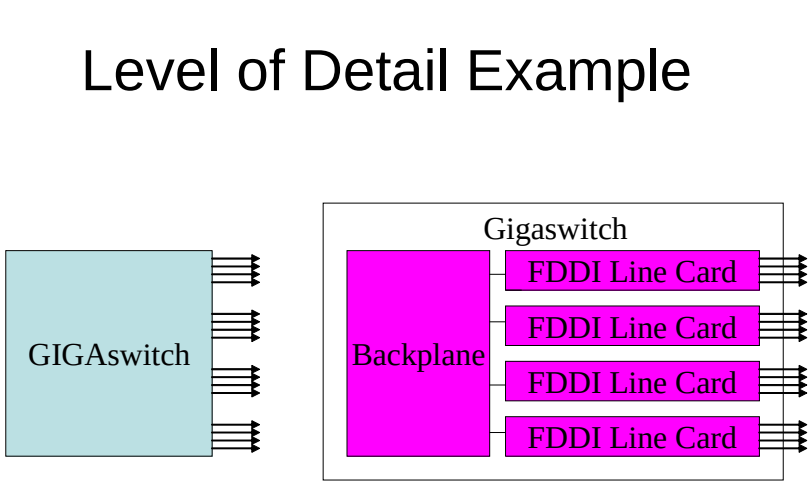
In this case, you will get error messages when you build LAVC$FAILURE_ANALYSIS because of duplicate symbol definitions unless you "comment out" (by putting a semi-colon in the first column) the first line of code after the following comment section:

```
; Within a single extended LAN address must remain unique.  The
; following line checks for unique LAN addresses in a single
; extended LAN configuration.  If your configuration is using
; multiple extended LANs then LAN address can be duplicated between
; the extended LANs.  In this case, the following line should be
; removed.
```

Level of Detail

There is a trade-off between the level of detail which is provided in diagnostic information generated and the amount of work required to initially set up the program and to maintain the program over time. More detail implies more work to set things up in the first place, and more maintenance work as things change, but provides more specific diagnostic information when failures occur and repairs are made.

For example, if a bridge/router or switch is represented as a single black box, notifications can only refer to the box as a whole. If an individual line card or port has failed, but the remainder of the box is working, it will appear to some nodes as if the entire box has failed, while to other nodes it will appear that a remote node's adapter has failed instead, and thus the error messages generated on different nodes will differ, depending on their view of the failure. If the extra effort is taken to identify individual line cards and ports within the box, and indicate to which line card and



port each LAN adapter is connected, then failure reports will be able to identify failures down to the line card level and even to the individual port level. The Level of Detail Example shows, on the left, a GIGAswitch represented as a black box, and, on the right, a GIGAswitch represented as line cards and the backplane connecting them in the box.

Other tools for use with LAVC$FAILURE_ANALYSIS

A DCL command procedure called EDIT_LAVC.COM is available to automatically gather the required information and create a working example LAVC$FAILURE_ANALYSIS.MAR program customized for a given cluster. While the level of detail within the LAN that such an automated tool can provide is limited, it provides a working example and starting point for further customization for a given site. This tool may be included on the V6 Freeware CD in the [KP_CLUSTERTOOLS] directory, but it can also be obtained from http://encompasserve.org/~parris/edit_lavc.com. See the EDIT_LAVC_DOC.TXT file at the same location for more information on how to use this tool.

The DCL command procedure SIFT_LAVC.COM found at the same location can be used to gather all the %LAVC messages from the OPERATOR.LOG files on all cluster nodes, and sort and list them in timestamp order, to gather a better overall picture of the failures as indicated by the varying views from each of the different nodes in the cluster.

Turning On LAVC$FAILURE_ANALYSIS

Once the LAVC$FAILURE_ANALYSIS program has been customized, it needs to be compiled and linked. A command procedure LAVC$BUILD.COM provided in SYS$EXAMPLES: can assist with building the program. Once the program has been built, it should be run once on each member of the cluster. The program loads the network graph data into non-paged pool, and reports the amount of space used.

If you make changes to the program, you can simply run the new version, and it will replace the older network graph data in memory with the newer version.

Because the network graph data goes away when a system reboots, to implement the LAVC$FAILURE_ANALYSIS facility on an ongoing basis, the program should be run once each time on each system as it boots up. This can be done by adding a command to run the program into the SYS$STARTUP:SYSTARTUP_VMS.COM procedure.

Disabling LAVC$FAILURE_ANALYSIS

If you ever wish to turn off LAVC Failure Analysis after it has been started on a given node, you can use the LAVC$FAILURE_OFF.MAR program found at the same location as EDIT_LAVC.COM.

Using LAVC$FAILURE_ANALYSIS to Monitor Non-cluster LAN Components

Sometimes cluster system managers will disable the SCS protocol on certain LAN adapters, particularly if the adapter is to be dedicated to traffic for another network protocol, such as IP. The SYS$EXAMPLES:LAVC$STOP_BUS.MAR program can be used to turn off the SCS protocol on a given LAN adapter, as can the new SCACP utility available in OpenVMS version 7.3 and above.

If SCACP is used instead to lower the priority for SCS traffic on a given LAN adapter rather than turning SCS traffic off entirely on that adapter, then PEDRIVER will transmit Hello packets through the adapter. LAVC$FAILURE_ANALYSIS can then be used to monitor and track failures on those additional LANs as well, while preventing the sending of SCS cluster traffic through that LAN, except as a last resort in the event that all other LANs fail.

Summary

The LAVC$FAILURE_ANALYSIS tool allows monitoring of LAN components and reports failures and repairs in the form of OPCOM messages. This article covered the theory of operation behind the LAVC$FAILURE_ANALYSIS tool, and how to set it up for use in a given cluster.

For more information

Documentation on LAVC$FAILURE_ANALYSIS

LAVC$FAILURE_ANALYSIS is documented in Appendix D of the OpenVMS Cluster Systems Manual (http://h71000.www7.hp.com/doc/731FINAL/4477/4477pro_028.html#network_fail_analysis). Appendix E (subroutines in OpenVMS that support the programs in Appendix D) and Appendix F (general information on troubleshooting LAVC LAN problems) can also be very helpful.

Spanning Tree Algorithm

For more information on the Spanning Tree algorithm, I highly recommend the book "Interconnections" (2nd edition) by Radia Perlman, ISBN 0-201-63448-1.

Contact Information

E-mail: Keith.Parris@hp.com
Web: http://encompasserve.org/~parris/ and http://www.geocities.com/keithparris/

Internet Technologies for OpenVMS

Ken Moreau
Solutions Architect, OpenVMS Ambassador, MCSE

Overview

This paper discusses Internet technologies and web services available on platforms other than OpenVMS, and shows how these technologies and services work in an OpenVMS environment. The topics discussed in this paper include Java, XML, Enterprise Application Integration (EAI), database connectivity, adding graphical front ends to existing terminal-based applications, and web browsers and servers.

Introduction

You have a history with OpenVMS. You have years and possibly decades of experience and knowledge embedded in the data, applications, business logic, and processes that are currently satisfying your business needs better than any other operating environment. You want to protect not only this business logic, but also the skills you have developed that built this business logic.

But there is a lot of pressure to move into the "brave new world" of the Internet. The question becomes, how do we most effectively use the best features of each world?

We do it by taking Internet technologies available on other platforms and making them available on OpenVMS. We do it by simply taking the data, applications, business logic, and processes that are currently running very well on OpenVMS, and exposing them to the larger Internet world, so that, for example, the UNIX client running the browser doesn't even notice it is using OpenVMS. So that the Excel spreadsheet on Windows, or the program running on Linux, doesn't even notice that the data they are fetching is coming from OpenVMS. So that the business-to-business (B2B) application that your business partners demand that you use simply works, and they don't even notice that they are running programs on OpenVMS.

And we do it by having your OpenVMS applications transparently connect to the rich variety of information available on the Internet, so that you can fetch data from another system, and you don't even notice that it is not running OpenVMS. This gives you transparent and seamless integration, while keeping all of the advantages of your OpenVMS systems: security (OpenVMS is absolutely immune to virtually all of the viruses we have seen), reliability, disaster tolerance, scalability, and so on.

Personally, this integration means you can continue to use the skills you already have and add some new ones to help you keep up with the latest and greatest technologies.

Java

Java was invented by Bill Joy and his team at Sun Microsystems, and represents one of the most exciting technologies available today. One of the reasons that Admiral Grace Hopper helped to specify COBOL back in 1959 was so that programs could run on every computer system available at the time. COBOL didn't quite fulfill that promise, and other languages like FORTRAN, C, C++, and BASIC still require work in order to compile and run on every operating system with every compiler from every vendor. But Java goes further than any other language in delivering on that promise. You truly can write Java programs once and run them everywhere. And that includes running them on OpenVMS.

Java is a programming language, with source code that is fairly easily readable. But Java is also a runtime environment with a lot of supporting infrastructure. The infrastructure is the key to Java's portability, because each operating system implements the infrastructure with exactly the same interfaces. Java programs call those interfaces and simply don't care what the underlying operating system is doing.

The two components of the Java environment are the Java Development Kit (JDK) and the Java Run-time Environment (JRE). The JDK contains all of the things that you would expect to be able to develop, debug, and deploy applications. The JRE contains only those components needed to deploy applications.

The combination of the two components comes in the three flavors: the Java 2 Platform Standard Edition (J2SE), which usually runs in client machines such as workstations or PCs, the Java 2 Platform Enterprise Edition (J2EE), which runs on the servers the client machines connect to, and the Java 2 Platform Micro Edition (J2ME), which runs on handheld devices that don't have a hard disk or a keyboard, such as an iPaq running PocketPC.

The J2SE, J2EE, and J2ME all contain a Java Virtual Machine (JVM), which all Java programs run within. Different operating systems implement the exact same virtual machine, so the fact that there are different operating systems underneath the virtual machine is hidden. OpenVMS can implement Java compilers and the JRE with a JVM, and all of the Java programs in the world just work.

A virtual machine is good, but you know how programmers are: they want to extend the base functionality. The Java specification includes the ability to add new functions to the JRE by creating "beans," which are simply run-time modules that the Java programs can then call, just as if they were specified in the original J2xE. Java programs simply state during the installation that they require this or that set of Java beans, enclosed in this or that Enterprise Java Bean (EJB) library. Anyone who wants to can write a Java bean, and can then supply it to the world, just by publishing it and getting other developers interested in using it.

Java programs also want to access data, and there are many database formats to choose from. In the same way that the Open Data Base Connectivity (ODBC) interface allows programs in any language to access databases of different formats, the Java Data Base Connectivity (JDBC) APIs are a set of Java beans that allow Java programs to access databases of different formats. Each database vendor, and a few other companies, have supplied beans that let Java programs talk to their specific databases, so all Java programs can access those databases without having to worry about the details of the database formats.

There are many APIs that have been added to the base specification. To understand the APIs, the first thing you need to know is that the developers are acronym-happy.

Here is a (very) partial list of the Java APIs:

- J2EE – Java 2 Platform Enterprise Edition. The base set of APIs and run-time utilities for every Java program running on a server.

- JAAS – Java Authentication and Authorization Service. If you want to authenticate your users, similar to logging in to OpenVMS and checking your password against the username and password in the SYSUAF.DAT file, you use JAAS.

- JAR – Java Archive, similar to UNIX 'tar' (tape archive) format.

- JCA – Java Connector Architecture, which defines the APIs to allow programs running on the application servers to connect to the traditional, legacy production systems running the core of the business, such as Enterprise Resource Planning (ERP) systems like SAP, or even some of the OpenVMS systems that you develop. The point here is to connect to information that is not in relational databases, because JDBC takes care of that problem. We will see more about this later.

  Developers couldn't let a perfectly good acronym go to waste by only using it for one thing, so JCA also stands for the Java Communications API, which lets Java communicate with voice-mail, FAX, and smart card devices. When you see this acronym, you will just have to figure out for yourself which one they mean.

- JDBC – Java Data Base Connectivity. Similar to ODBC, but callable by Java programs to access data from a variety of databases and systems.

- JDK – Java Development Kit. Development is done with the JDK. Sun invented Java and still sets the specifications for it, and everyone, including HP, licenses the tools from Sun. Sun wants to encourage the use of Java, so the license terms are reasonable.

- JMS – Java Message Service. If one Java program wants to send a message to another Java program, it uses JMS. This shows the power of Java to hide the underlying operating system. On UNIX we might implement this with pipes. On OpenVMS we might implement it with mailboxes. If we are communicating between systems we might use TCP/IP sockets. JMS works either synchronously or asynchronously, using either point-to-point connections or publish/subscribe methods. The Java program doesn't care, it simply uses the JMS interface and it just works.

- JNDI – Java Naming Directory Interface. This allows you to talk to Common Object Request Broker Architecture (CORBA), Lightweight Directory Access Protocol (LDAP), X.500 or Microsoft Active Directory directories, and register your own names and look up other names in your enterprise name directory.

- JRE – Java 2 Platform Run-time Environment. This contains the virtual machine and the set of APIs common to all Java implementations.

- JSP – Java Server Page. Sun observed Microsoft's Active Server Pages, so Java has Java Server Pages. Java Server Pages technology allows web developers and designers to rapidly develop and easily maintain information-rich, dynamic web pages that leverage existing business systems. JSP enables rapid development of web-based applications that are platform independent, and separates the user interface from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content.

- JTA – Java Transaction Architecture. Computer systems are important, and they are very reliable: OpenVMS is extremely reliable. But sometimes they do fail (OpenVMS less so than any other platform), and frequently transactions are distributed among several different systems running different operating systems. People use an external transaction monitor like Tuxedo to watch over the state of transactions inside the entire environment. Java communicates with these and monitors the entire distributed transaction with the JTA.

- JPDA – Java Process Debug Architecture. How many people here write perfect code the first time? For the rest of us, Java specifies a system-independent way to debug Java applications, using the JPDA. It is not a debugger itself, but provides the tools to build debuggers and to run full debugging consoles on remote systems, across a network.

- JRMI – Java Remote Method Invocation, JRMI allows you to call programs and invoke methods outside of the system you are running.

The intent of all of these Java APIs, and the dozens more that I didn't list, is just like the JRE or the JDBC specification: to have a single way of doing something across all operating systems. You call the specific function using the right interface, and your programs that were developed on, for example, a UNIX flavor or Linux, simply run on OpenVMS.

There are many more Java APIs. For more details, enter "Java" in your favorite search engine, or go to http://www.java.sun.com/products. This is the definitive website for all Java work, and is the high level page that will lead you to more information about all of these technologies and a lot more.

How does all of this fit together? There are two major pieces: the client side and the server side, with the server side broken down into several different components.

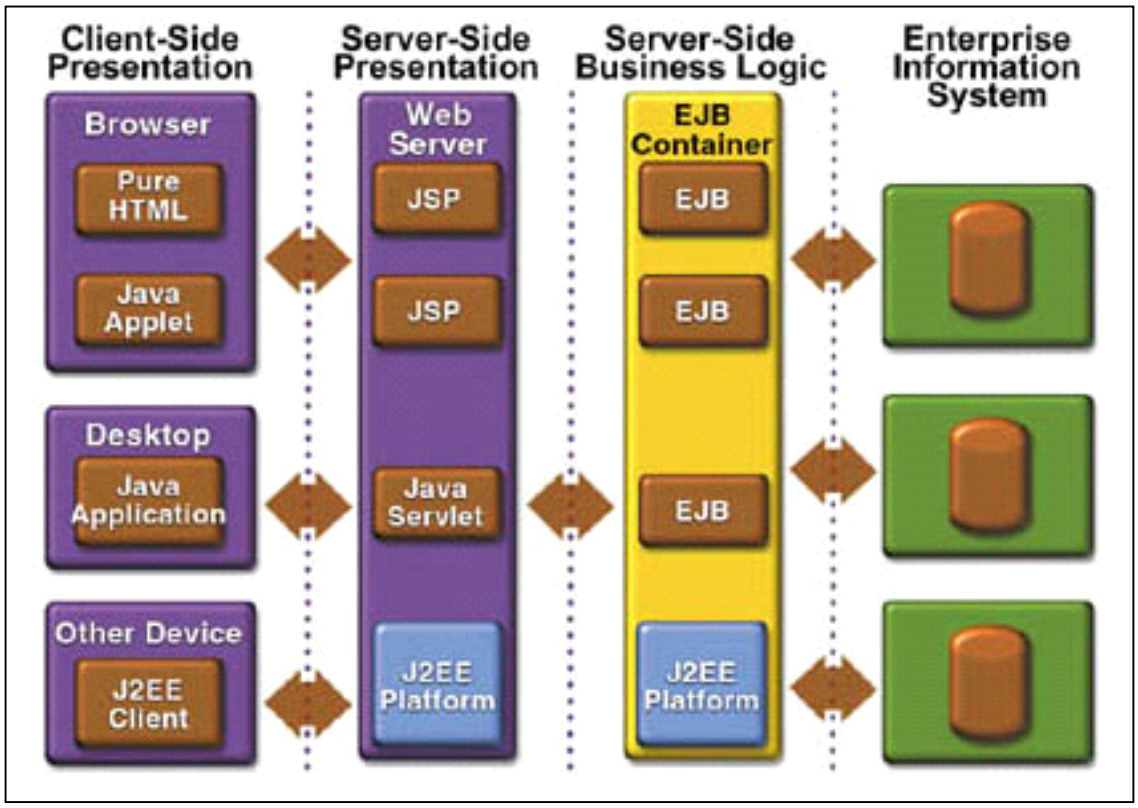
The client side includes the things that we are most familiar with, primarily browsers. The applications can be written in pure HTML or in Java. Some of these are not full applications like we understand them; they are smaller and less rich in functionality. As such, we call them applets instead of applications. You can also write sophisticated applications, because Java is a full 3rd generation language (3GL). You can even use devices other than PCs, such as iPAQs, web-enabled phones, and so on, using the J2ME client.

The server is divided into the presentation layer and the business logic layer. If you have ever used DECforms or FMS to develop applications, you understand the distinction between presentation and business logic.

The presentation layer is written in Java and runs inside a web server, which is the JRE and includes the JVM referenced earlier. Just like before, we can have simple pages written as Java Server Pages (JSP) and simple applications written on the server, called servlets. All of this is written in Java, and depends on the J2xE server environment.

The business logic can be written in any language. (We will talk later about how to integrate your existing programs into this layer, but for the moment we will talk about business logic written in Java.) These programs are created as enterprise Java beans, and are just like the enterprise Java beans that come as part of many packages. Whether you write the bean or someone else does, it is all simply part of the environment, and run in the same environment.

On the back end is the data itself, which is accessed by the JDBC interfaces discussed earlier.

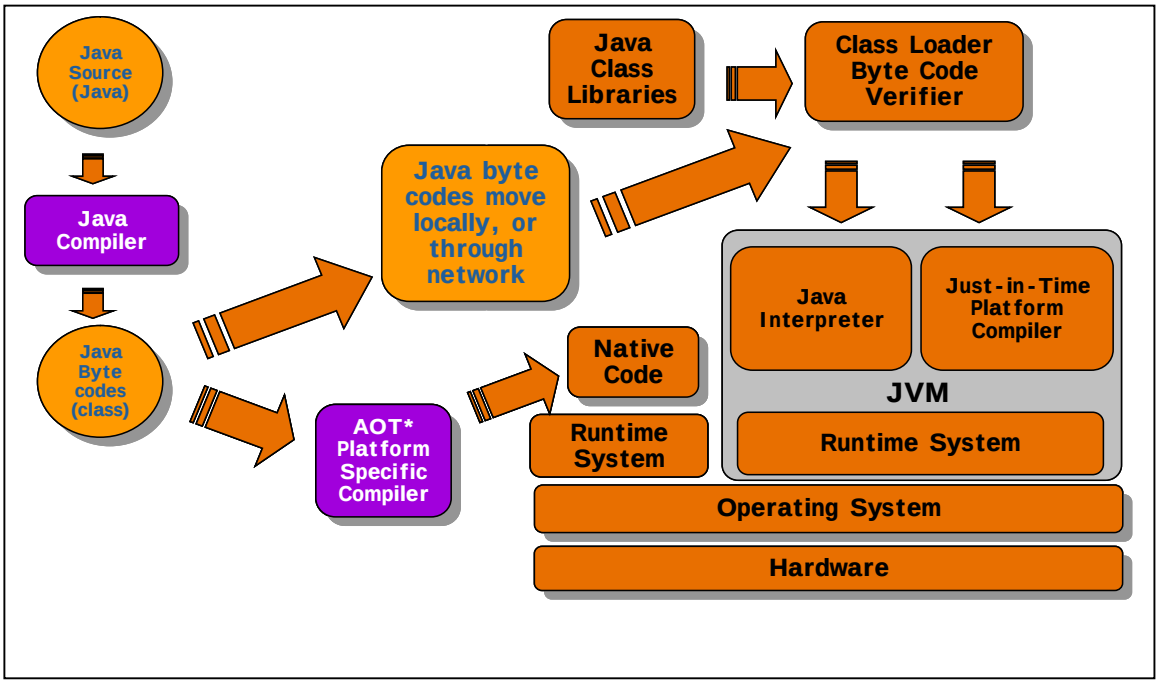


How do you create these beans? In much the same way you do for other languages, but there are a few differences.

Start with source code and run it through the compiler, the same as in any language like FORTRAN, COBOL, or C/C++. The result of this isn't object code, but a system-independent byte stream, which you then send to the system that will run the Java program. This program can be an applet, a servlet, or a full application.

Once the byte stream is on the target system, it is combined with the class libraries and fed to the JVM. You can run the byte code directly through the interpreter, or execute the Just-In-Time (JIT) compiler that is specific to the operating system and hardware. This compiler takes the byte code and turns it into a real native application for that specific O/S and hardware combination.

The interpreter is slower but more flexible, while the JIT compiler makes the application run faster but takes longer to compile, so there is a higher startup cost. One way around this is to use the Ahead-of-Time (AOT) compiler, which pays this cost at compile time. Think of this as a standard compiler and linker, which compiles Java instead of C/C++ or FORTRAN.



Some people want more than a compiler to build applications: they want an Integrated Development Environment (IDE). The IDE for Java is called NetBeans. This was developed by a group of students who needed a Java IDE, and is now distributed by Sun.

Third parties can create their own branded distributions of the NetBeans IDE including additional modules they've written, or can simply make those modules available either commercially or for free to plug into existing distributions of NetBeans. Sun One Studio, OptimalJ and ObjectAssembler are three examples of extensions to the base NetBeans IDE.

There is an "update center" in the IDE that allows modules to be downloaded and installed into a running copy of the IDE. http://netbeans.org hosts an update center server for non-commercial modules. Sun operates an e-commerce enabled update center that allows third parties to sell and distribute commercial modules online.

The NetBeans Update Center has an OpenVMS section for our plug-ins, which are also available from the OpenVMS web site. The C/C++ compiler support is tailored to the OpenVMS C/C++ compilers and includes source colorization, code formatting, source code compilation (but not linking), handling of messages that come from the compiler to update the source window, and provides DCL command procedure execution either in the same window or a separate terminal window.

IBM developed their own IDE framework called Eclipse (aimed at Sun). In 2002 they made it open source, and specifically didn't invite any NetBeans people to the party. Eclipse uses the Standard Widget Toolkit (SWT) written in native code for each platform. IBM does not port this to OpenVMS, so IBM's Eclipse will not be available on OpenVMS. Don't confuse this with HP's Eclipse, which is the Java back-end that HP wrote.

What does all of this mean for OpenVMS?

Everything we talked about is available on OpenVMS: Java, the development environment, J2EE, NetBeans, JDBC -- it's all on OpenVMS. Every time Sun releases a new version, or someone releases a new NetBean or EJB library, we release them on OpenVMS. And I say "we" inclusively,

because a lot of this software comes from places outside of HP. We depend on our software partners like BEA, Attunity, and Apache. They are releasing their code on OpenVMS at the same time that they are releasing it on every other platform.

The bottom line is: Java is the same on OpenVMS as it is on every other operating system. Anything you can do in Java on HP-UX, Windows, Linux, AIX, or even Solaris, you can do on OpenVMS.

There are some differences between Java on UNIX and OpenVMS -- some of which you would expect, and some of which are surprising:

- Most of the development work for Java and the beans is done on UNIX systems. This means that the developers frequently require the use of shell scripts (what we call command procedures) to carry out simple tasks. Because they are on UNIX and/or Windows, these are not written in DCL. When porting a Java program to OpenVMS, you need to convert the UNIX shell scripts to DCL.

- For classic OpenVMS people, UNIX has some bizarre requirements for file names: upper and lower case, strange characters, really long names, lots of dots, and so on. This means that those Java applications that use file names that aren't supported under ODS-2 must be installed on an ODS-5 disk. This is not a requirement for Java itself, which is installed in SYS$COMMON. Many applications use file names that work on an ODS-2 disk, but it is something to be aware of if you get applications from other operating environments.

- UNIX doesn't have the concept of relative or indexed files, and everything is in what we would call STREAM_LF format. Java programs assume that as an access method, so you have to be careful to do an RMS CONVERT to get the data files into that format.

- UNIX has fairly loose process quotas. Sometimes this surprises people who are running Java programs on OpenVMS for the first time, when they don't perform as you expect them to. Check your process quotas to see if you are artificially limiting the performance, and observe what tuning changes you need to make. See the OpenVMS Java documentation for more details.

- The final restriction is on the hardware. Java runs on OpenVMS Alpha today, and will run on OpenVMS Itanium when that ships. Java will not run on OpenVMS VAX.

You can deploy Java on OpenVMS, but you can develop it anywhere. So take advantage of all of the development being done in other parts of your company, or even on the Internet, and compile them on the other platforms and deploy them on OpenVMS, just like any other platform.

Data Integration

Have you ever gone to a form on the web and started filling in the blank fields, and noticed that some of your information popped up to help you fill in the information? For example, your first name, your last name, your phone number, your FAX number, your street address, your city, your state, your ZIP code? How did the form know to fill in the right information at the right spot? This was probably done with eXtensible Markup Language (XML), which is a markup language for documents containing structured information.

Structured information contains both content (words, pictures) and some indication of what role that content plays. A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

XML is similar to HTML in that they both separate form from content. But in HTML, both the tag semantics and the tag set are fixed. <h1> is always a first level heading, is always bold, etc.

XML specifies neither semantics nor a tag set. XML is a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there's no predefined tag set, there can't be any preconceived semantics. So all tags are defined by the creator of the tag, and given meaning and usage by that creator.

As an example, both HTML and XML can show information that looks to us like a date.

- 5 May 2003
- <ChangeDate>5 May 2003</>

The HTML document doesn't know anything about that information except that it should be displayed in bold at that point on the screen. The XML document defines something more about it: it is the change date. Elsewhere the "style sheet" has defined all sorts of attributes about this tag, including what format it is in (so you can display dates differently in different parts of the world), how to display it (so it will always be in bold) and what methods can be applied to it.

So it is not just text, it is actually information.

In our earlier example with the web form, at some point you filled in a form that had tags like <FirstName>, <LastName>, and so on, and your information was associated with those tags. The next time another web page asked you to fill in the form, the style sheet of that form used the standard definitions for those fields, and automatically picked up the information from the first form. It doesn't matter that the first form was running Microsoft IIS on a Windows platform and the second form was running BEA WebLogics on an OpenVMS platform -- XML provided the common definitions between the two forms.

All of the semantics of an XML document are defined either by the applications that process them or by style sheets. Style sheets define the attributes; that is, the format and the functionality of the tag. For example <FirstName> might be defined as text, specifies that there is only one token, and that it is a required value in this form.

You can create new definitions for data that is specific to your application as you need them. You might also consider looking into some of the industry-standard style sheets to use their definitions of common data. Not only is it easier on you, but your forms will integrate and cooperate with the forms of all other forms that use the same style sheet. This again reinforces the point that OpenVMS is just another platform as far as Internet technologies are concerned.

Now that we have common data definitions with all of the other platforms, we need to provide access to the data that is on our OpenVMS systems. We have seen how JDBC and XML provide the calling standards to let other systems look at RMS or Rdb or even Oracle 7/8/9i data on OpenVMS, but a calling standard isn't a running program. Attunity Connect provides that running program. Attunity Connect provides a universal integration solution across a wide range of enterprise systems consisting of varied application and data technologies on legacy platforms. Applications on other systems can simply access the data, without noticing that the data is on an OpenVMS system.

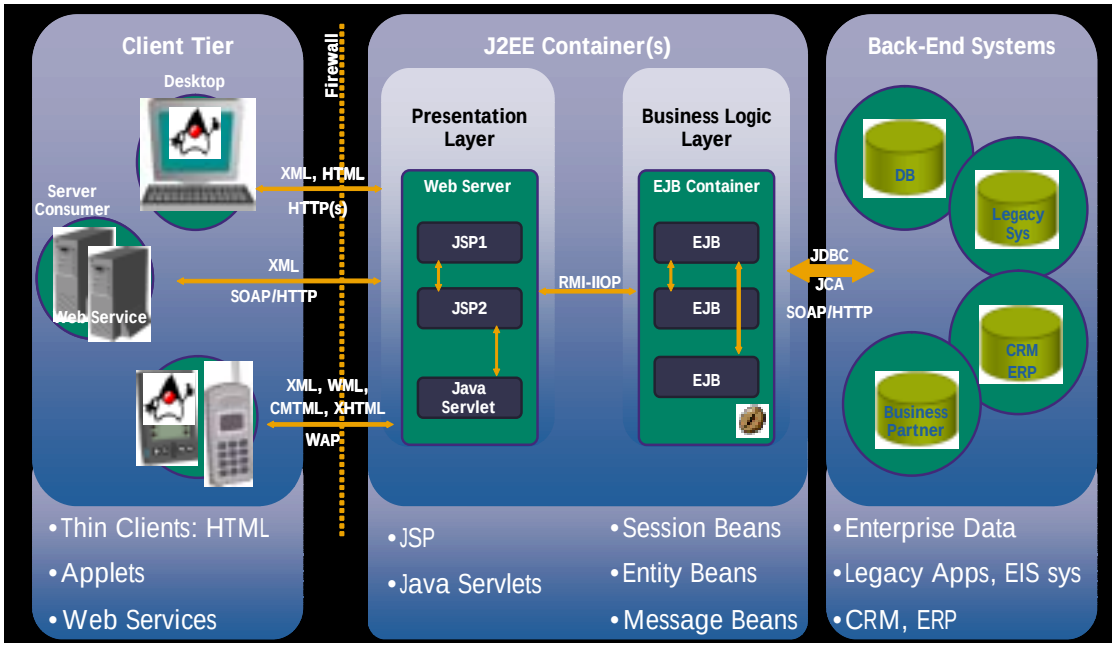
These systems consist of data and application resources that become available to the client application through Attunity Connect. A single Attunity Connect installation integrates applications as well as data sources on the same machine. Attunity Connect provides a universal integration solution both vertically from clients to servers and also horizontally across servers.

J2EE integration is made easy via J2EE Connector Architecture (JCA) and JDBC standards. XML can be also used directly.

What this means is that any client can directly access any data on your OpenVMS system. Java programs using JDBC, 3GL programs using ODBC, or Excel spreadsheets using XML, can just read and write data on your OpenVMS systems, without any special programming on their part.

This frequently makes it easier on you. If you give someone access to your data in their Excel spreadsheet, then they think it is up to them to write the fancy reporting programs and data analysis tools that they need, and not your job to do that for them. So you have shared the data, and they do the work. Sounds better than what is happening now, doesn't it?

So let's put it all together.



Starting from the right-hand side, the back-end systems, we find our enterprise data and many legacy applications. We access the data via JDBC and Attunity Connect, and specify the format and use of the data via XML. (We will talk about SOAP later.)

The new applications are written in Java so it doesn't matter where they run, so let's run them on OpenVMS. We create the business logic layer inside an Enterprise Java Beans container, which communicates to all of the EJB libraries that we imported from the Internet and other places, to give us the rich functionality our users expect. We communicate with the presentation layer running on a web server, again running on OpenVMS, running Java Server Pages and Java servlets. The two components can run one the same or different systems, running the same or different operating systems, and communicate via the Java Remote Method Invocation (JRMI) and the Internet Inter-Orb Protocol (IIOP), which again we will talk about in a few minutes. The session, entity, and message beans are simply data that exists for the life of that object: session beans exist while a session exists between the client and the server, entity beans exist while the object that created them exists, and message beans exist for the life of the message.

The client side is anything we want it to be. Windows clients, Linux clients, PocketPC or Palm clients, OpenVMS workstations, iPaqs, web phones – anything that can talk to a web server and display information. We communicate with this level through HTML, XML, Wireless Access Protocol (WAP), and so on.

The message to keep in mind here is that OpenVMS simply fits into each of the layers, and all of the software needed to run each layer exists on OpenVMS.

Application Integration

For applications to begin working with each other, they have to communicate in some way. The way applications communicate is via messaging. The generic term for this set of routines is Message Oriented Middleware (MOM). Middleware is just a fancy name for programs and APIs that allow one set of programs to interact with another set of programs, whether or not they happen to be on the same system or the same operating environment. You may already be familiar with some message oriented middleware, such as the Application Control Management System (ACMS) or the Reliable Transaction Router (RTR).

Message Oriented Middleware is, as the name implies, middleware focused on getting messages from here to there. But there are some difficulties with the wide varieties of "here" and "there" that exist, including distance between systems, transmission protocols from mailboxes to shared files to UNIX pipes to local area networks to wide area networks, and so on. In addition, networks sometimes aren't as reliable as we need them to be, and some computer systems count the bits in a different order than other computer systems count them (some are big-endian and some are little-endian).

MOM takes care of all of this, by having a set of code on each system, including OpenVMS, which simply accepts messages from somewhere else, performs all of the right transformations on them, and passes them to the calling program. It does the same thing in reverse when one of your programs wants to send a message to some other system. Notice that MOM is totally uninterested in what the messages contain.

Examples of MOM include the previously mentioned RTR, as well as BEA MessageQ, IBM MQSeries, SpiritWave and SpiritJMQ from SpiritSoft (SpiritWave is the general purpose MOM while SpiritJMQ is focused on Java), BEA JMS and Tibco ActiveEnterprise.

Every one of these products runs native on OpenVMS, and simply communicates to a similar messaging system on any other platform. By adding calls to these routines to your existing applications, you can cooperate with applications on other systems, wherever they are.

The other set of tools for application integration is a higher level function, where the communications and messaging are taken for granted, and probably uses one of the tools we just talked about. This is the world of objects.

Objects are very similar to programming APIs, where you have a name of a function, a set of parameters to that function, and a set of return values from the function, all with datatypes carefully specified. The difference between objects and the programming APIs you are used to (such as calling RMS functions or SYS$QIO), is that these objects are not linked into your program. What is linked into your program are stubs, which act like the routine API that your program expects, but don't directly call code in your process address space. Stubs, when invoked by your program, use the MOM to call the routine somewhere else in the world, dynamically at runtime. The MOM passes the parameters, performs any transformation necessary, and then the remote object executes. The MOM then takes care of passing any data and return value back to the calling system, and your program never noticed that anything happened.

There are two forms of objects that currently exist in the industry. The first was created by an industry consortium, called the Common Object Request Broker Architecture (CORBA). They defined how to define stubs, message passing, return values, and so on, and then multiple companies implemented CORBA-compliant code on many platforms. JRMI Internet Inter-Orb Protocol (IIOP) is an example of CORBA-compliant code for the Java world, with IONA Orbix and 2ab Orb2 as two products that implement that specification.

The second form of object was created by Microsoft, and was known by various names including Component Object Model (COM), COM plus (COM+) and Distributed COM (DCOM). Today it is simply called COM. Due to the popularity of Microsoft platforms, all other platform vendors including OpenVMS implemented code to interact with COM objects.

More recently, Microsoft announced a development environment called .NET, running on the server platform called Windows 2003. This is an extension of the object model, and again all the platform vendors are implementing code to interact with .NET objects. Keep in mind that "interact with" doesn't mean that the target code runs on OpenVMS. It specifically means that your code running on OpenVMS communicates via COM with the code running on some other operating system, such as Windows 2003.

One problem with the MOM tools is that they often use protocols that can be filtered out by networking firewalls and other security devices. To get around this problem, people are using the Simple Object Access Protocol (SOAP). SOAP is a way of sending procedure calls from one system to another through Hyper Text Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), which is allowed through firewalls and other security devices, so it is quite transportable. It can be called from anywhere, including web pages and XML documents.

OpenVMS is fully compliant with all of these protocols, and objects running on OpenVMS can both call and be called by objects running on almost any other operating environment through this middleware. This means that you can begin using the web services that are currently running on other systems in your enterprise and throughout the Internet. In the next section we will talk about how to turn your existing code into objects, to make them into web services.

All of the APIs that you need to get your applications to use MOM are available on OpenVMS. However, many application developers today insist on using GUI-based development environments.

The HP Enterprise Toolkit -- OpenVMS Edition provides an interactive development environment based on Microsoft Visual C++ V6.0 and Visual Fortran V6.1. The HP Enterprise Toolkit -- OpenVMS Edition runs on Windows 95, Windows 98, Windows NT, and Windows 2000 systems and allows software developers to develop OpenVMS applications using an interactive PC environment. The developer can use C, C++, Fortran, COBOL, BASIC, Pascal, and Ada to write, compile, debug and tune applications in the familiar PC environment and run them in an OpenVMS computing environment.

The HP Enterprise Toolkit -- OpenVMS Edition adds several components to Visual Studio to provide additional software development functions such as: remote source file editing, remote compilation, linking, and building, remote find in files, remote debugging, source browsing, terminal emulation, and context-sensitive help. It also provides access to OpenVMS documentation, support for team programming, the sharing of project files among team members, user-defined configuration names, workspace-based source code control, support for using file shares to access remote project files, and built-in access to source code control on OpenVMS systems.

This simplifies software development with one environment for developing on multiple platforms, and provides a choice of tools and extends the industry-standard Microsoft development environment to OpenVMS.

In addition, the OpenVMS NetBeans team is currently developing a NetBeans plug-in that will allow distributed OpenVMS development to occur on any desktop that can run NetBeans, such as Linux, HP-UX, OpenVMS or Windows. The team is also creating a plug-in to support debugging of OpenVMS 3GL programs written in languages other than Java, such as C, C++ and FORTRAN, using the NetBeans debugger GUI and the Java Process Debug Architecture (JPDA) API.

HP plans to use this enhanced NetBeans environment to eventually replace the HP Enterprise Toolkit, to allow more flexibility in the development environments, and to provide a more standard development platform.

You have a choice: use the tools that you are used to today, or use the GUI-based development environments that are used for other platforms, either the Enterprise Toolkit from the Microsoft world, or the Java NetBeans environment for the Java world. Either way, you can use all of the message oriented middleware that allows you to communicate with other systems.

Legacy Application Integration

There are two ways to offer legacy OpenVMS applications to the rest of the enterprise as web services: you can add GUI front ends to existing applications with no change to the base application, or you can wrap the applications inside an object.

For applications written with DECforms, HP offers the DECforms Web Connector. This is a layered software product that runs on OpenVMS systems and provides transparent Web access to interactive applications, where DECforms was used to implement the forms-based user interface. The DECforms Web Connector lets you preserve large investments in DECforms-based user interfaces without requiring any programming or application changes. The DECforms Web Connector works by specifying a logical name at run-time that specifies the type of screen to present to the user: either the traditional DECforms screen on their current VT, or re-direct to a GUI platform such as Microsoft Windows or a web browser.

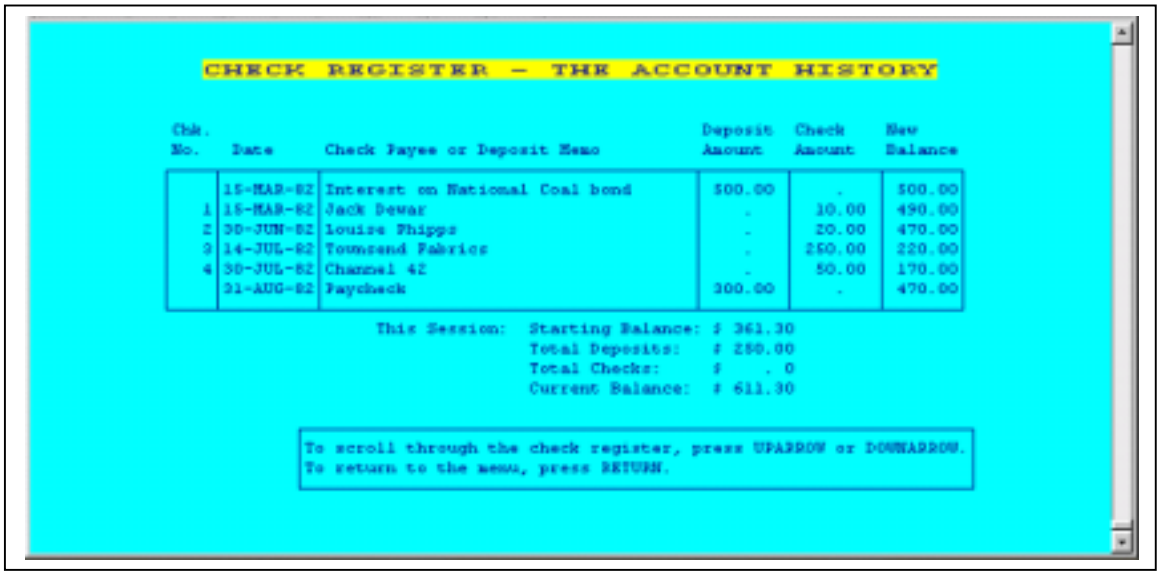
For the Application Control and Management System (ACMS) that runs on OpenVMS, and the cross-platform version (ACMS*xp*), HP offers the TP Web Connector, which web-enables business applications running on ACMS and ACMSxp for Windows NT transaction processing (TP) systems.

Using TP Web Connector, customers can create browser interfaces to any of these TP systems using a desktop tool that supports development with Automation, C-language, or Java. TP Web Connector can also connect Windows NT (MTS) based systems to ACMS or ACMS*xp* for Windows systems. TP Web Connector supports integration of object modeling with critical business applications, all of the popular web server environments, and provides a high performance solution for web-enabling ACMS applications. The aim of this is not to migrate you off of ACMS, but in fact to do exactly the opposite: allow you to keep using ACMS but still give your application access to all of the other Internet technologies.

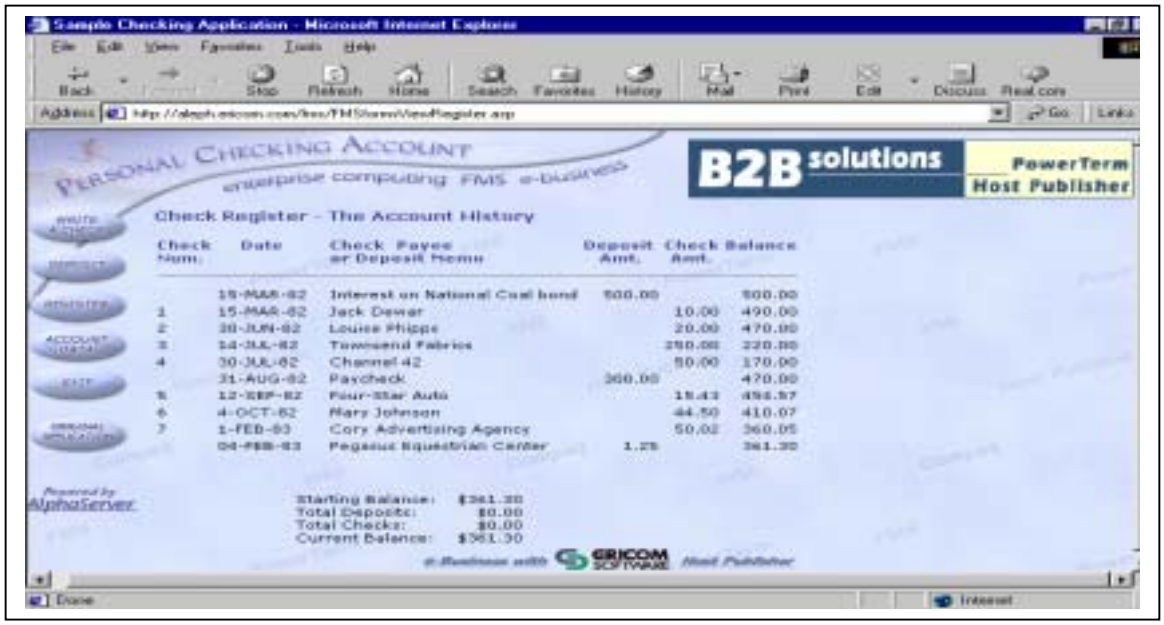
For a more general purpose solution, for applications that don't use DECforms or ACMS, HP works with Ericom to add new GUI front ends to existing "green screen" applications. The Host Publisher can even combine multiple applications into a single screen, for real enterprise application integration (EAI).

Here is an example of a forms-based VT-based application.

This is what the screen looks like on a VT terminal, just the way it has looked for years. Simple, clean, easy for us to work with, but not exactly the most modern look.



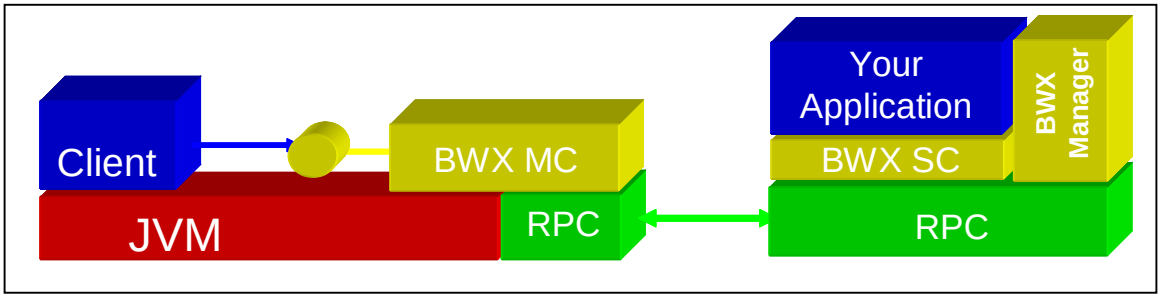
And here is the way the same application looks with a new front end. Same data, same program, no re-compile, but something that people who have grown up on Windows or Macintosh screens are familiar with.



The original screen continues to work just the way it always has.

The other alternative is to encapsulate the existing applications. BridgeWorks can take existing applications, or pieces of applications, and turn them into components that can be used by external programs as Java Beans or as COM objects. The way this works is that BridgeWorks uses the OpenVMS Calling Standard to encapsulate the existing code into a wrapper whose external interface is either an EJB, a light weight Java Bean, or a COM object. Any part of your program that can be called by the OpenVMS Calling Standard can then be used as a standard Web component. You can even encapsulate your command procedures written in DCL!

Here is a detailed view of what HP BridgeWorks generates for the developer.



Your application is encapsulated by BridgeWorks (here abbreviated BWX), hidden by the Server Component (SC) and managed by the BridgeWorks Manager. This is exposed to the world via a message oriented middleware interface called the Remote Procedure Call (RPC) interface. The RPC interface can be either an EJB, Java Bean, or COM object, which you specify when you set up the encapsulation.

Your application communicates with the client side using the standard MOM technologies described in the last section, using either message-based or object-based communications. On the

client side is the Messaging Component, which connects the standard client (browser, etc.) to the BridgeWorks wrapper to the application.

The client can be running on top of a Java Virtual Machine, or not, as is appropriate for your client.

In summary, you have years, even decades, worth of investment in your applications, which you want to preserve and use. Sometimes it makes business sense to re-write the application: you have learned better ways of doing things, it is in a language that is no longer popular, the business requirements have changed, and so on. But sometimes the program is working so well you just need a better front end on it, or a different way to call it. In these cases, OpenVMS has the ability to change the front end to a modern GUI, or encapsulate the functionality of the application into components that other systems will see as standard Java Beans or COM objects.

You can keep your investment, while still cooperating with the Web interfaces in the rest of your company. This allows you to take your applications and offer them as web services to the enterprise.

Systems Integration

OpenVMS has a standard web browser, called the HP Secure Web Browser (CSWB). It is based on the Mozilla open-source project started in 1998 by Netscape Communications Corporation. The Mozilla web browser is designed for standards compliance, performance, and portability. This is an example of the advantages I have been talking about up to this point: there is a perfectly good open-source program available, and it just runs on OpenVMS.

The Secure Web Browser is licensed as part of the OpenVMS operating system license and provides a full-featured, customizable browser with integrated web-browsing and security; HTML document creation and editing; clients for mail and news; Secure Socket Layer (SSL) for security, and an Internet Relay Chat (IRC) client is available.

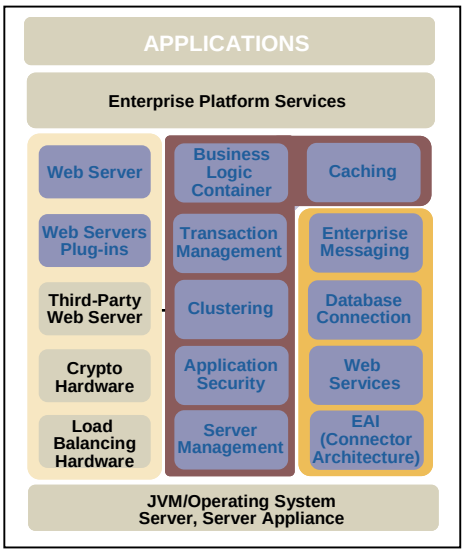
For web servers, we have the same scenario: the HP Secure Web Server (CSWS) is also based on industry standard open-source code, this time from the Apache Software Foundation. Apache is in use on millions of web servers around the world, on every major platform, and OpenVMS has the full functionality of Apache. And with all of the recent fuss over Internet viruses attacking web servers, keep in mind that Gartner Group recommends the use of Apache instead of Microsoft's IIS. So you get the enhanced reliability of Apache combined with the virus-proof nature of OpenVMS, for a very reliable and secure product.

CSWS, like the rest of Apache, is completely modular. Each function can be added to the server as you wish. Most people tend to add all of the available modules. Tomcat is the name of the Java module, with scripting languages like PHP Hypertext Processor (PHP, and yes, the acronym is recursive), Perl and Python modules freely available.

Web servers are all well and good, but all they do is present web pages, whether they are written in HTML or Perl or another language. They don't do everything we need, such as provide directory services, messaging, caching, and so son. For this we need a full application server.

Application servers integrate and coordinate all of the functionality that we have discussed. Your application simply assumes that all of this functionality is there, and the application server simply does it. You as a developer certainly could take pieces from everywhere and build your own application server, in the same way you could build your own operating system. The maintenance would be a nightmare, and there is no need because there are excellent application servers already available to you.

Let's take a look at the components of an application server.



The first component is the web server described in the last section, along with its plug-ins like Java, Perl, and PHP. You can also plug in other components as you need them, including load balancing and high availability with OpenVMS Clusters (that is the "load balancing hardware" component in the lower left of the picture), as well as high security devices like cryptographic hardware and software.

The second component is the middleware for the middleware, if you will. Enterprise beans that you publish with common applications business logic are placed in an EJB container, available for use by any of your applications. The cluster manager handles transactions that are spread across multiple platforms using JTA, clustering of the application servers on different members in a cluster, caching of information in this server and between the application servers, and security of the transactions with JAAS.

The third component is the connection to higher level entities. The actual sending and receiving of messages via MOM is done here through the JMS interface, as is the JDBC connections to any databases opened by the application. Advertising the names of the web services offered by the applications running here is done with JNDI, as well as locating the names of services offered elsewhere and stored in the enterprise directory is done at this level. Finally, any connection to mainframe type applications such as SAP is handled here.

Quite a lot of work is done by the application server. In effect it is a mini-operating system, offering services to the applications that run on it. The advantage of application servers, however, is that they are common across many operating systems, so you can develop applications that run on one application server on one operating system, and they will just run on the same application server on another operating system.

The primary application server that HP recommends for OpenVMS is the BEA WebLogic Server. This is fully supported on OpenVMS, and brings the full functionality of BEA WLS to OpenVMS. Another excellent application server, which is not quite as well known, is the Xoology Concerto web server. Xoology has been running on OpenVMS for many years, and works very well.

One of the problems of MOM is that it creates a disconnect between the calling program and the called procedure. The called procedure is often an object, so there are various ways to invoke them, but one thing we have been ignoring is, how do we find them? How do we look them up, how do we tell which system they are on today, and how do we tell how they wish to be invoked?

The industry solution for this is the Lightweight Data Access Protocol (LDAP). LDAP describes how to access a directory of objects stored in a central location called an Enterprise Directory. In effect, the Enterprise Directory is a phone book of objects for your systems to use to find the services they need. You can store anything in there: people, phone numbers, e-mail addresses, other objects, a BridgeWorks wrapper around a DCL procedure, anything.

As you would expect, different vendors have different ideas on who should hold the data. Microsoft has implemented the Windows 2000 Active Directory, Novell has the Novell Directory Service, and there are two industry standards around this: X.500 and Kerberos. (Kerberos shipped with OpenVMS beginning with Version 7.3.) Finally, the Universal Description, Discovery and Integration (UDDI) protocol is a layer on top of XML and DNS, which acts as a building block to allow systems to quickly find and transact operations between disparate applications.

No matter which of these data holders you choose, OpenVMS can access it. OpenVMS can act as either a Directory System Agent (DSA), which holds the data and lets the Directory User Agents (DUA) on other systems access it, or can act as a DUA and can access the DSA on some other system.

So we can not only have your OpenVMS applications find all of the objects on the other systems, but we can advertise the objects that you will be developing on your OpenVMS systems.

Availability

OpenVMS considers all of the software described in this article as base functionality, and is included in OpenVMS. This software is available from the OpenVMS web site for download, or it is available as a CD bundled with every operating system kit. Further, the support for this software is included in the support contract for the base OpenVMS operating system license:

- The Secure Web Browser and Secure Web Server
- COM object technology
- All of Java, including the software development kit
- NetBeans
- SOAP
- The Attunity Connect JDBC drivers
- RTR
- The Enterprise Directory
- BridgeWorks
- XML for Java and C++

There is no additional license involved; it is all part of the base operating system. If you want to try any of this stuff out, simply grab it off the CD, or download it from our web site at http://h71000.www7.hp.com/ebusiness/technology.html.

I have been telling you how easy it is to bring software to OpenVMS, and how easy it is to take existing OpenVMS software and make it work with the Internet technologies. So you might very well ask if anyone has actually done this. Yes, many people have, and here are some examples:

- The GNU people are developing quite a lot of software that is in the UNIX style, but is specifically and deliberately not UNIX: their very name is Gnu's Not UNIX (which again is a recursive acronym like PHP). They developed a package called GNV, which stands for Gnu's Not VMS. This is a complete implementation of a UNIX shell, which we would call a command line interface, called BASH (the Bourne Again Shell). How many times have you run into a UNIX script that you couldn't run in DCL? GNV solves this problem quite nicely. Also part of GNV is a full UNIX C run-time library. The combination of the two of these makes it very easy to port full UNIX applications to OpenVMS without resorting to major surgery, including the ability of the application to spawn a UNIX script to perform some functions. UNIX system administrators and users really like this, and it is facilitating moving a lot of UNIX applications to OpenVMS.

- Another popular set of tools are GTK+ and libIDL. GTK+ is a development environment for creating graphical user interfaces, and libIDL is a development environment for creating CORBA interface definitions. These are part of many UNIX applications.

- There is a lot of secure information on your OpenVMS system and on other systems. Pretty Good Privacy (PGP) is a very popular package that encrypts information to an acceptable level: maybe not up to the requirements of the Defense community, but pretty good privacy. PGP is available in a package called GnuPG. The Secure Sockets Layer (SSL) is a popular method to secure network access between your OpenVMS applications and the outside world, or applications running on other systems that want to connect to the data on your OpenVMS systems. sTunnel is an implementation of SSL that runs at the network layer, so applications that currently don't understand security (such as telnet) can have secured communication.

- ZIP is an implementation of a very common file compression and packaging utility that runs on every system in the world.

- Finally, OpenVMS does not offer native support for writeable CD-ROMs. CD Record is an open source implementation of this functionality.

All of the above software is free, and available for download from http://h71000.www7.hp.com/opensource/.

Summary

You have a lot invested in your applications, your data, and in OpenVMS. You need to find a way to get the most return on that investment.

Every new technology goes through multiple stages:

- Phase 0 has a few small groups promoting something as the best thing since sliced bread.

- Phase 1 has a few more people adopt it as an alternative to the established standard.

- Phase 2 has people really understanding it, and beginning to see through the hype.
- Phase 3 has the product come back with more modest expectations and more solid engineering, and become a success.

Web services are somewhere between phases 1 and 2 right now, but we need to understand that the push toward inter-operating environments, regardless of platform, will not stop. It may or may not be Java, Windows 2003, XML or something else, but it is coming, and OpenVMS people need to understand this.

HP is partnering with the industry leaders to make this happen. Whether any given technology works out over the long term, these companies are going to be part of the cutting edge, and OpenVMS will be right there with them.

Acknowledgement

A great deal of this material was borrowed from seminars written and presented by John Apps and Mick Keyes. Without their assistance and review, this article would not have been possible. They are the real experts in this area.

For more information

All OpenVMS eBusiness technologies
http://h71000.www7.hp.com/ebusiness/technology.html

OpenVMS Open Source Tools
http://h71000.www7.hp.com/opensource/

OpenVMS Java
http://h18012.www1.hp.com/java/download/index.html

OpenVMS CSWS (based on Apache)
http://h71000.www7.hp.com/OpenVMS/products/ips/apache/csws.html

OpenVMS JServ
http://h71000.www7.hp.com/OpenVMS/products/ips/apache/jserv.html

OpenVMS Perl
http://h71000.www7.hp.com/OpenVMS/products/ips/apache/apache_perl.html

OpenVMS SOAP
http://h71000.www7.hp.com/OpenVMS/products/ips/soap/soap.html

Apache Project
http://www.apache.org/

Apache Java Project
http://java.apache.org/

Apache JServ Project
http://java.apache.org/jserv/index.html

Java Servlets
http://jserv.javasoft.com/

Java Productss
http://www.java.sun.com/products

Web Services
http://www.webservices.org
http://www.ibm.com/developerworks/webservices/

Simple Object Access Protocol (SOAP)
http://msdn.microsoft.com/soap
http://www.develop.com/soap
http://www.soapware.org/
http://xml.apache.org/soap/faq

Books

Java Web Services by David A. Chappell & Tyler Jewell
Understanding Web Services by Eric Newcomer
HTML: The Definitive Guide, Chuck Musciano and Bill Kennedy
Apache Server Bible, Mohammed J. Kabir
Apache Server Unleashed, Rich Bowen and Ken Coar
Apache, The Definitive Guide, Ben Laurie and Peter Laurie
Writing Apache Modules with Perl and C: The Apache API and mod_perl, Lincoln Stein and Doug MacEachern
Professional Apache, Peter Wainwright
SSL & TLS, Designing and Building Secure Systems, Eric Rescorla
OpenVMS with Apache, OSU, and WASD, The Nonstop Webserver, Alan Winston

HTTP 1.1 specification, http://www.ietf.org/rfc/rfc2616.txt

HTML 4.01 specification, http://www.w3.org/TR/html4/

Configuring TCP/IP for High Availability

Matt Muggeridge
TCP/IP for OpenVMS Engineering

Overview

High availability of the network complements other high availability features associated with OpenVMS clustering. In the OpenVMS network, several high-availability technologies can be used in isolation or combined to provide a high-availability solution to meet a multitude of requirements.

The key to configuring a high availability solution of any kind is careful planning with an ethos of *"keep it simple."* Understand where failures might reasonably occur, what types of failures can be tolerated and what failures cannot be tolerated. It is wise to consider the impact of a catastrophic failure and how taking the appropriate precautions can mitigate the impact on the system.

TCP/IP high availability solutions include:

- failSAFE IP[1] – address failover to alternate interfaces
- IP Cluster Alias – superseded by failSAFE IP
- Load Broker/Metric Server – DNS alias name dynamically updated with available addresses
- LAN Failover[2,3]

This paper describes the key difference between these technologies and the environments that best suit their application.

[1] failSAFE IP is introduced with TCP/IP V5.4, which is in field test at the time of writing.
[2] LAN Failover is introduced with OpenVMS V7.3-2, which is in field test at the time of writing.
[3] LAN Failover is not discussed in detail in this paper. Refer to the OpenVMS V7.3-2 documentation.

Comparing High Availability Technologies

Table 1 briefly describes each of the technologies to help the reader compare the features and choose which solution or combination of solutions is best suited to their environment. Each of the technologies is described in depth in subsequent sections.

Table 1 High Availability Network Technologies

	failSAFE IP	IP Cluster Alias	DNS Alias (Load Broker / Metric Server)	LAN Failover
Protects	All IP addresses	Single IP address designated as the cluster address	DNS Alias with list of most available IP addresses	MAC Address
Protocols	IP only	IP only	IP only	All LAN protocols
Scope	Interfaces within a node or cluster	Single interface per node in a cluster	DNS name lookup	Interfaces within a node
NIC	Independent of interface type	Independent of interface type	Not applicable	DE600 and DEGXA
Load Balancing	All interfaces active, balance outgoing connections, higher throughput	One interface in a cluster is assigned the cluster address, no load balancing	Load share inbound connections across DNS alias addresses	One interface in a node is active others are standby, no load balancing
Detects	Failure and recovery: interface, cable, switch, node	Node failure	Most available nodes	Failed interface, cable, and switch
Addressing	May require additional IP addresses per cluster	Requires an additional address per cluster	Multiple addresses listed for DNS alias	LAN virtual interface address automatically generated
Notes	Monitor at least 3 interfaces on a LAN to avoid phantom failures	Superseded by failSAFE IP	Does not protect against all interface failures in multihomed hosts	
Availability	Introduced with TCP/IP V5.4	Long-time feature of TCP/IP Services	Long-time feature of TCP/IP Services	Introduced with OpenVMS V7.3-2

failSAFE IP

The network interface controller (NIC) is often regarded as a single point of failure (SPOF) in a network. Typical failures include NIC failure, disconnected or broken cable, or a dead port on the switch. failSAFE IP removes the NIC as a SPOF. (failSAFE IP is introduced starting with Version 5.4 of HP TCP/IP Services for OpenVMS.)

Introduction to failSAFE IP

failSAFE IP provides IP address redundancy when the same IP address is configured on multiple interfaces. Only one instance of each IP address is active at any time; the other duplicate IP addresses are in standby mode[4]. Standby IP addresses may be configured on multiple interfaces within the same node or across a cluster. The failSAFE service monitors the health of each interface and takes appropriate action upon detecting interface failure or recovery.

When an interface fails, each active IP address on the failed interface is removed and the standby IP address becomes active. If an address is not configured with a standby, then the address is removed from the failed interface until it recovers. Static routes on the failed interface are also removed and migrated to any interface where their network is reachable.

When an interface recovers, it may request the return of its IP addresses. The IP address is returned when the recovering interface is configured as the *home* interface for one or more addresses. When the *home* interface recovers, it requests that the current holder of the address give it up[5]. (The concept of a *home* interface is discussed in Home Interfaces.)

The current holder of an address will not release an address if it would result in dropped connections, nor if the current holder is also designated as a *home* interface for that address. Management intervention can force the removal of an address.

failSAFE IP Configuration Requirements

Configuring failSAFE IP requires two steps:

1. *Assign the same IP address to multiple interfaces*. Only one instance of that address will be active; all other instances will be in standby mode. For simple configurations, use the TCPIP$CONFIG Core Environment menu to assign an IP address to multiple interfaces; see the TCP/IP Services for OpenVMS *Installation and Configuration* guide for more information. Alternately, use the *ifconfig* utility, which provides a greater degree of management control; see Table 2 for more information.
2. *Enable the failSAFE IP service*, which monitors the health of interfaces and takes appropriate action upon detecting interface failure or recovery. This service is enabled using the TCPIP$CONFIG Optional Components menu.

failSAFE IP Service – Interface Health Monitor

The failSAFE IP service monitors the health of interfaces and upon detecting a failure or recovery will take the appropriate action. The service is enabled using TCPIP$CONFIG and is started and stopped with the TCP/IP Services startup and shutdown procedures. Alternately, it may be started or stopped using:

[4] The OpenVMS distributed lock manager is used to ensure only one instance of an IP address is active across a cluster. An exception to this is any address assigned to the loopback interface *'LO0'*. For instance, the localhost address, 127.0.0.1 must be configured on every node in a cluster. See Management Utilities for more information.

[5] An IP address may be configured with multiple home interfaces. By default, the primary address is configured with its interface marked as a home interface.

```
SYS$STARTUP:TCPIP$FAILSAFE_STARTUP.COM

SYS$STARTUP:TCPIP$FAILSAFE_SHUTDOWN.COM
```

The failSAFE IP service:

- Monitors the health of interfaces by periodically reading their *"Bytes received"* counter.
- When required, marks an interface as failed or recovered.
- Maintains static routes to ensure they are preserved after interface failure or recovery.
- Logs all messages to TCPIP$FAILSAFE_RUN.LOG. Important events are additionally sent to OPCOM.
- Generates traffic to help avoid *phantom failures,* (see Avoiding Phantom Failures).
- Invokes a customer written command procedure at the transitions marked by an asterisk in Figure 1 below. (Refer to Site-Specific Customization of failSAFE IP for more detail on site-specific command procedures).

The finite state machine for the failSAFE IP service is shown in Figure 1.

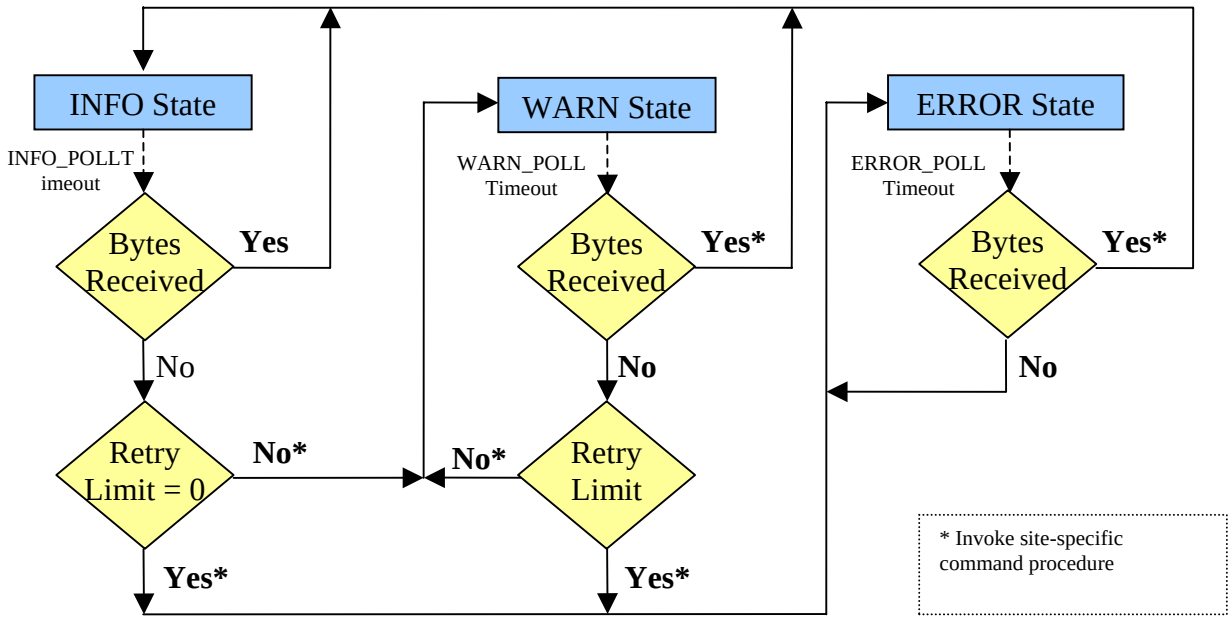


Figure 1 Finite State Machine for failSAFE IP Service – Interface Health Monitor

If the failSAFE IP service is not enabled, then configuring a failSAFE IP address across nodes provides identical functionality to the IP Cluster Alias, described in IP Cluster Alias.

Configuring failSAFE IP Service

By default, the failSAFE IP service monitors all TCP/IP interfaces on a system and periodically polls each interface using default polling intervals. The defaults may be overridden by editing the configuration file defined by the logical name TCPIP$FAILSAFE, which by default is:

SYS$SYSDEVICE:[TCPIP$FSAFE]TCPIP$FAILSAFE.CONF[6]

The configurable parameters are:

Parameter	Description
INTERFACE_LIST [Default: ALL interfaces]	The list of interfaces that failSAFE monitors.
INFO_POLL [Default: 3 seconds]	The polling interval used when the interface is known to be functional. Two INFO_POLL timeouts are required to determine that an interface is not responding, at which time the polling frequency is set to the WARN_POLL period.
WARN_POLL [Default: 2 seconds]	The polling interval used when the interface first stops responding. Polling will continue for RETRY_WARN attempts before the interface is deemed dead, at which time the polling frequency is set to ERROR_POLL and failover occurs.
RETRY_WARN [Default: 1 retry]	The number of warning polls before the interface is deemed dead and the IP addresses associated with it are removed. A value of zero will skip the WARN_POLL cycle.
ERROR_POLL [Default: 15 seconds]	The polling interval used when the interface is considered dead. failSAFE IP will monitor a dead interface at this frequency until it determines the interface has recovered, at which time the polling frequency is set back to the INFO_POLL period.

Detectable Failures

The failSAFE IP service periodically reads the network interface card's (NIC's) "*Bytes received"* counter to determine the health of an interface. This is the same counter that can be viewed using LANCP. For example, to view all interfaces' *"Bytes received"* counters:

```
$ pipe mcr lancp show device/count | search sys$pipe "Bytes received"/exact
```

failSAFE IP guards against any event that prevents the *"Bytes received"* counter from changing or any event that results in the deletion of an IP address, such as:

- Interface hardware failure
- Physical link disconnect
- Shuttng the interface down using TCP/IP management commands
- Shutting down TCP/IP Services

[6] A template file is provided when the service is configured via TCPIP$CONFIG. The template also describes the file syntax.

- Shutting down a node

Application

All environments that require high availability of IP addresses can benefit from failSAFE IP. There are two failure scenarios to consider:

1. If the IP address migrates to an interface on the same node, existing traffic-flow continues uninterrupted and both incoming and outgoing connections are maintained.
2. If the IP address migrates to an interface on another cluster member, existing connections are dropped. However remote clients will be able to establish new connections immediately. In this scenario, failSAFE IP addresses are better suited to UDP applications or those applications that permanently cache IP addresses. It is up to the network administrator whether to configure failSAFE IP addresses across interfaces within the same node or across cluster members.

failSAFE IP will always preferentially fail over addresses to interfaces on the same node before failing over across clustered nodes.

Management Utilities

For many situations, failSAFE IP requires no additional management beyond the initial configuration with TCPIP$CONFIG. This section describes new management commands that are used by the failSAFE IP service – or by system administrators who need to manually intervene with IP address assignment.

A failSAFE IP address may be configured using TCPIP$CONFIG, or manually using the TCPIP management commands. For instance to create an IP address of 10.10.10.1 on interface IE0 and a standby alias address on interface IE1 (pseudo-interface IEB0) the following commands may be used (the *ifconfig* command is shown for comparison):

```
$ TCPIP
TCPIP> SET INTERFACE IE0/HOST=10.10.10.1     ! ifconfig ie0 10.10.10.1
TCPIP> SET INTERFACE IEB0/HOST=10.10.10.1    ! ifconfig ie1 alias 10.10.10.1
```

To view the standby addresses, it is necessary to use the *ifconfig* command. For example:

```
$ ifconfig -a
IE0: flags=c43<UP,BROADCAST,RUNNING,MULTICAST,SIMPLEX>
   *inet 10.10.10.1 netmask ff000000 broadcast 10.255.255.255

IE1: flags=c03<UP,BROADCAST,MULTICAST,SIMPLEX>
     failSAFE IP Addresses:
        inet 10.10.10.1 netmask ff000000 broadcast 10.255.255.255 (on HUFFLE IE0)
```

Note that interface IE1 displays a failSAFE IP address, and that it is active on node HUFFLE, interface IE0.

Greater control of failSAFE IP addresses can be achieved with the *ifconfig* command. The *ifconfig* options that support failSAFE IP are described in Table 2.

Table 2 New ifconfig Options for failSAFE IP

Option	Description
[-]fail	Force an interface to fail by using the *fail* option or to recover by using the *–fail* option.
[-]home	Used when creating IP addresses. By default, all primary IP addresses are created with a home interface. To force an alias address to be created with a home interface, the *home* option must be used.
[-]fs	All IP addresses are created as failSAFE addresses by default, except for addresses assigned to the loopback interface *LO0,* for instance, the localhost address 127.0.0.1. To create an address that is not managed by failSAFE, use the *-fs* option.

Home Interfaces

failSAFE IP addresses may be created with a designated *home* interface. By default, all primary IP addresses are created with a *home* interface. The purpose of a home interface is to provide a preferential failover and recovery target in an effort to always migrate IP addresses to their home interface. This gives the network administrator greater control over how IP addresses are assigned to interfaces. The *ifconfig* management utility may be used to create and display addresses configured with home interfaces. For example to create three addresses:

```
$ ifconfig ie0 10.10.10.1            ! primary has home interface by default
$ ifconfig ie0 alias 10.10.10.2      ! alias does not
$ ifconfig ie0 home alias 10.10.10.3 ! create alias with home interface
```

Note that the *TCPIP SET INTERFACE* command may also be used to create primary and alias addresses. However, it does not support creation of the *home alias* address. For this, *ifconfig* must be used.

When addresses are displayed with the *ifconfig* utility, those addresses with a home interface are marked with an asterisk (*). For example, displaying the addresses created with the previous commands reveals:

```
$ ifconfig ie0
IE0: flags=c43<UP,BROADCAST,RUNNING,MULTICAST,SIMPLEX>
   *inet 10.10.10.1 netmask ff000000 broadcast 10.255.255.255
    inet 10.10.10.2 netmask ff000000 broadcast 10.255.255.255
   *inet 10.10.10.3 netmask ff000000 broadcast 10.255.255.255
```

The asterisk character indicates that the addresses 10.10.10.1 and 10.10.10.3 have a home interface of IE0. Note that *TCPIP SHOW INTERFACE* does not identify addresses with a home interface.

Creating IP addresses with home interfaces helps to maintain the spread of IP addresses across multiple interfaces. This is important for load-balancing and gaining higher aggregate throughput. In the event a home interface recovers after a failure, the addresses may return to their recovered home interface, thus maintaining the spread of addresses across the available interfaces.

Note that an address will *not* migrate toward a home interface if it will result in dropping TCP/IP connections.

Site-Specific Customization of failSAFE IP

A user-defined procedure may be invoked during selected transitions of the failSAFE IP service's finite state machine. Refer to Figure 1 for the state transitions that invoke the site-specific command procedure. These transitions describe one of three events:

1. When the interface first appears to have stopped responding. This is the first warning that a problem may exist, but no action to failover IP addresses is taken yet.
2. When an attempt to generate traffic on the interface fails. After the retry limit is reached, the interface is deemed dead, and IP addresses will be removed from the interface. Failover occurs.
3. When the interface recovers.

The procedure is called with two string parameters:

```
P1 = TCP/IP Interface Name (e.g. "IE0")
P2 = state ("INFO_STATE", "WARN_STATE", "ERROR_STATE")
```

The site-specific procedure may be defined by the logical name TCPIP$SYFAILSAFE; otherwise the default file is:

```
SYS$MANAGER:TCPIP$SYFAILSAFE.COM
```

Logical Names

The logical names in Table 3 may be used to customize the operating environment of failSAFE IP. These logical names must be defined in the LNM$SYSTEM_TABLE for them to take effect.

Table 3 failSAFE IP Logical Names

Logical Name	Description
TCPIP$FAILSAFE	Configuration file that is read by TCPIP$FAILSAFE during startup. If the logical is not defined then the default configuration file is: SYS$SYSDEVICE:[TCPIP$FSAFE]TCPIP$FAILSAFE.CONF This logical must be defined prior to starting the failSAFE IP service.
TCPIP$FAILSAFE_FAILED_<*ifname*>	This logical is used to simulate a failure for the named interface. The logical is translated each time failSAFE IP reads the LAN counters. The <*ifname*> can be determined using the TCPIP SHOW INTERFACE command.
TCPIP$SYFAILSAFE	The name of a site-specific command procedure, which is invoked when one of three conditions occurs: • Iinterface failure

	• Retry failure • Interface recovery If the logical is not defined, then the default procedure is: SYS$MANAGER:TCPIP$SYFAILSAFE.COM
TCPIP$FAILSAFE_LOG_LEVEL	Controls the volume of log messages sent to OPCOM and the log file. If the logical is undefined or has a value of zero, the default log level is assumed. Larger values are used for debugging. This logical name is translated each time failSAFE IP logs a message.
TCPIP$FSACP_LOG_LEVEL	Controls the volume of log messages sent to OPCOM by the ACP. This logical name should be used only when directed by customer support.

Static and Dynamic Routing

When an interface fails, failSAFE IP removes all addresses and static routes from the failed interface. The static routes are reestablished on every interface where the route's network is reachable. This may result in a static route being created on multiple interfaces and is most often observed with the default route.

Dynamic routing may need to be restarted to ensure the dynamic routing protocol remains current with changes in interface availability. If this is necessary, restart the routing process using the TCPIP$SYFAILSAFE procedure, as described in Site-Specific Customization of failSAFE IP. For example, for GATED:

```
$ TCPIP STOP ROUTING /GATED

$ TCPIP START ROUTING /GATED
```

For GATED users, the configuration supports the *scaninterval* option, which allows you to periodically scan the interfaces to detect any changes. Scanning can be forced by issuing the command:

```
$ TCPIP SET GATED/CHECK_INTERFACES
```

For more information on routing protocols refer to the appropriate section in the TCP/IP Services for OpenVMS *Management Guide*.

Best Practices

These best practices are a guide to assist the network administrator to quickly come up to speed with the various aspects of failSAFE IP by avoiding common pitfalls.

Validating failSAFE IP

Most contemporary networks are highly stable and rarely suffer from the problems that require failSAFE IP. Consequently, for the small number of occasions where failSAFE IP is required, it is

critical that it has been previously validated in the environment where it is being deployed. Failure to do this may result in unexpected problems at the critical moment.

Since real failures are rare and sometimes difficult to simulate, the logical name TCPIP$FAILSAFE_FAILED_<*ifname*> has been provided. After configuring failSAFE IP addresses and starting the failSAFE IP service, the validation procedure is as follows:

1. *Establish connections and generate IP traffic*

   Using TELNET or FTP, create incoming and outgoing TCP connections to the multihomed host from inside and outside the subnet. Verify that these connections are established, using the following commands:

```
$ @sys$manager:tcpip$define_commands
$ ifconfig -a  ! Check the interface addresses
$ netstat -nr  ! Check the routing table
$ netstat -n   ! identify which interface(s) are being used
```

2. *Simulate a failure and observe*

   Simulate a failure and observe OPCOM and log file messages. The failure may be simulated with:

```
$ define/system tcpip$failsafe_failed_<ifname> 1  ! or disconnect the cable
```

   Wait long enough for failover to occur, which will be signaled by OPCOM messages. Now observe the effects of failover and verify TCP connections are still established and can transfer data. For example, TELNET sessions should respond to keyboard input.

```
$ ifconfig -a  ! Observe how the addresses have migrated
$ netstat -nr  ! Observe how the routing table has changed
```

3. *Recover and observe*

   Recover from the simulated failure and observe the OPCOM messages.

```
$ deassign/system tcpip$failsafe_failed_<ifname>  ! or reconnect the cable
$ ifconfig –a ! Observe how the addresses have migrated
$ netstat –nr ! Observe how the routing table has changed
```

Once again, ensure TCP connections are still established and can transfer data

Be aware that simulating a failure with the logical *TCPIP$FAILSAFE_FAILED_<ifname>* does not disrupt physical connections to the machine, and as such is not a true indicator of whether the services will survive a real failover situation. Consequently, this procedure should be repeated by physically removing a network cable from one or more of the interfaces. Since this may potentially be disruptive to network services, this operation should be scheduled into a maintenance period where a disruption may be tolerated.

Configuring failSAFE IP Service

The key concern for configuring the failSAFE IP service is the time it takes to detect a failure and for the standby IP address to become active. One goal of a failSAFE IP configuration is to avoid disrupting existing connections, so the failover time must be within the connection timeout.

The failover time is calculated as:

```
INFO_POLL + (WARN_POLL * RETRY) < failover time < (2 * INFO_POLL) + (WARN_POLL * RETRY)
```

Refer to Figure 1 for an explanation of the variables. The default values (INFO_POLL=3, WARN_POLL=2, RETRY=1) result in a failover interval range of between 5 and 8 seconds. Note that this does not take into account the system load.

The recovery time will be less than the ERROR_POLL period, which has a default of 30 seconds. See Configuring failSAFE IP for more information about the failSAFE IP configuration parameters.

Avoiding Phantom Failures

The health of a NIC is determined by monitoring the NIC's *"Bytes received"* counter. This provides a protocol-independent view of the NIC counters. However, in a quiet network, there may be insufficient traffic to keep the *"Bytes received"* counter changing within the *failover detection time*, thus causing a *phantom failure.* To counteract this, the failSAFE service attempts to generate MAC-layer broadcast messages, which are received on every interface on the LAN *except for the sending interface.*

Consequently, in a quiet network with just two interfaces being monitored by the failSAFE service, a single NIC failure may also result in a phantom failure of the other NIC, since the surviving NIC is not able to increase its own "*Bytes received*" counter.

You can reduce phantom failures in a quiet network by configuring the failSAFE IP service for at least three interfaces on the LAN. In the event that one interface fails, the surviving interfaces will continue to maintain each others "*Bytes received*" counter.

Creating IP Addresses with Home Interfaces

By default, the interface on which a primary IP address is created is its home interface, while an IP *alias* address is created without a home interface. To create an alias address with a home interface, use the *ifconfig* command, which should be added to the SYS$STARTUP:TCPIP$SYSTARTUP.COM procedure. For example to create an alias address of 10.10.10.3 on interface IE0 and designate IE0 as its home interface, the following command could be used:

```
$ ifconfig ie0 home alias 10.10.10.3/24
```

Private Addresses Should Not Have Clusterwide Standbys

For the purpose of this discussion, private addresses are those used for network administration and not published as well-known addresses for well-known services. A standby interface for a private address should be configured on the same node as the home interface. This avoids the situation where a node cannot assign any addresses to its interfaces if they have active connections on another node in the cluster. This is further illustrated in Example 2.

If it is desirable to associate the list of private addresses with a public DNS alias name, then it is recommended that the Load Broker be used to provide high availability of the DNS Alias. The Load Broker is described in DNS Alias with Load Broker and Metric Server.

Examples

Example 1 – Single node configured with two interfaces

Consider a node named *HUFFLE* with two interfaces *IE0* and *IE1*. Each interface is configured with a unique primary IP address and each interface is also configured as a standby for each other. The addresses are *10.10.10.1/24* and *10.10.10.2/24*. These addresses and the failSAFE IP standby aliases are easily created using the TCPIP$CONFIG *Core Environment* menu.

For the purpose of clarification, the commands that TCPIP$CONFIG uses are shown in the table below. The identical *ifconfig* commands are shown for comparison.

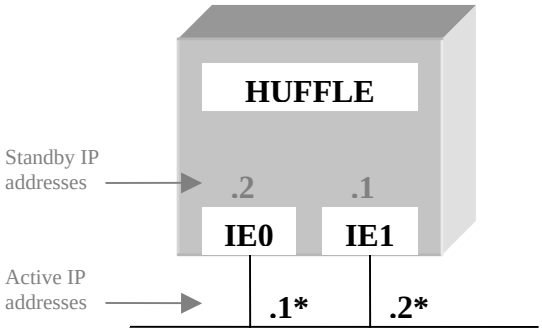
Configure IP addresses:

Action	TCP/IP Command	*ifconfig* command

Create Primary Addresses	`$ tcpip set interface ie0 –`
	` /host=10.10.10.1 -`
	` /net=255.255.255.0 -`
	` /broad=10.10.10.255`
	`$ tcpip set interface ie1 -`
	` /host=10.10.10.2 -`
	` /net=255.255.255.0 -`
	` /broad=10.10.10.255`
		`$ ifconfig ie0 10.10.10.1/24`
		`$ ifconfig ie1 10.10.10.2/24`

	`$ tcpip set interface iea0 –`
	` /host=10.10.10.1 -`
	` /net=255.255.255.0 -`
	` /broad=10.10.10.255`
	`$ tcpip set interface ieb0 -`
	` /host=10.10.10.2 -`
	` /net=255.255.255.0 -`
	` /broad=10.10.10.255`
		`$ ifconfig ie0 alias`
		`10.10.10.2/24`
		`$ ifconfig ie1 alias`
		`10.10.10.1/24`

At this point, the node will be configured as shown in Figure 2.



Network: **10.10.10/24**

Figure 2 Simple failSAFE IP Configuration

Examining the configuration with *ifconfig* reveals how each interface is configured with an active primary address as well as a standby failSAFE IP address. The asterisk beside the address denotes that address's home interface. In the sample output below, note that the standby failSAFE IP addresses also describe where the IP address is active. For example, for interface IE0, the standby address 10.10.10.2 is active on node *HUFFLE*, interface *IE1.* The asterisk before the address indicates that the respective interface is its home interface. Home interfaces are described in more detail in Home Interfaces.

```
$ ifconfig ie0
IE0: flags=8000c43<UP,BROADCAST,RUNNING,MULTICAST,SIMPLEX>
     failSAFE IP Addresses:
        inet 10.10.10.2 netmask ffffff00 broadcast 10.10.10.255 (on HUFFLE IE1)
   *inet 10.10.10.1 netmask ffffff00 broadcast 10.10.10.255

$ ifconfig ie1
IE1: flags=c43<UP,BROADCAST,RUNNING,MULTICAST,SIMPLEX>
     failSAFE IP Addresses:
        inet 10.10.10.1 netmask ffffff00 broadcast 10.10.10.255 (on HUFFLE IE0)
   *inet 10.10.10.2 netmask ffffff00 broadcast 10.10.10.255
```

In the event of an interface failure (for example, IE0 fails), the failSAFE IP service marks the interface as failed, using the following command:

```
$ ifconfig ie0 fail
```

Figure 3 shows the state of the node after IE0 has failed. The *ifconfig* commands are also shown below. Note that interface IE1 is now configured with both addresses and the output from *ifconfig ie0* shows that the interface is in a failed state.

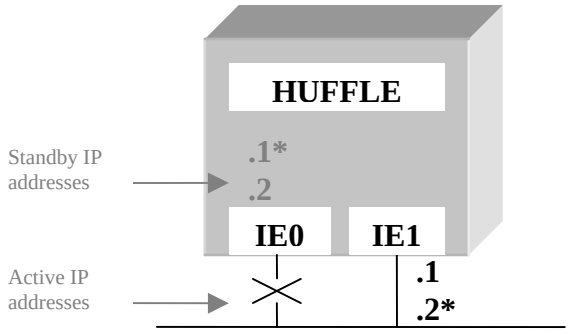
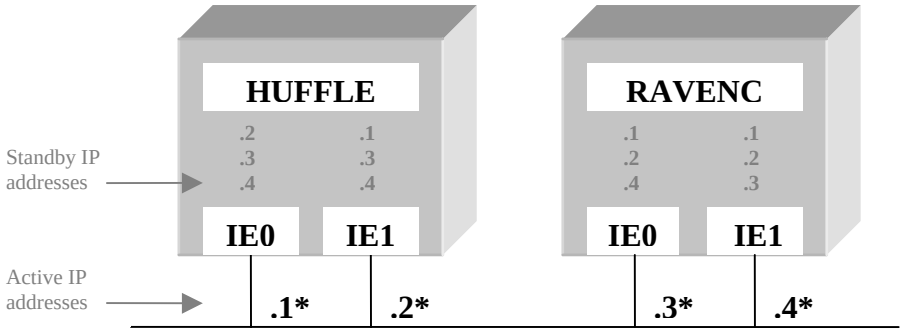


Figure 3 Interface IE0 has failed

```
$ ifconfig ie0
IE0: flags=8000c43<UP,BROADCAST,RUNNING,MULTICAST,SIMPLEX>
    *failSAFE IP - interface is in a failed state.
     failSAFE IP Addresses:
        inet 10.10.10.2 netmask ffffff00 broadcast 10.10.10.255 (on HUFFLE IE1)
       *inet 10.10.10.1 netmask ffffff00 broadcast 10.10.10.255 (on HUFFLE IE1)

$ ifconfig ie1
IE1: flags=c43<UP,BROADCAST,RUNNING,MULTICAST,SIMPLEX>
    inet 10.10.10.1 netmask ffffff00 broadcast 10.10.10.255
   *inet 10.10.10.2 netmask ffffff00 broadcast 10.10.10.255
```

Example 2 – Clustered Nodes configured with Two Interfaces

Extending the previous example to two similarly configured nodes in an OpenVMS cluster:

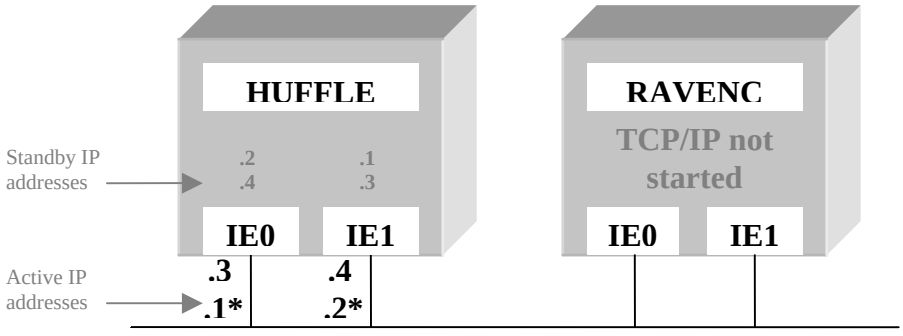


Network: **10.10.10/24**

Executing an *ifconfig* command on node HUFFLE reveals information about all failSAFE IP addresses that are configured on HUFFLE. Note that IE0 is the home interface for address 10.10.10.1, as indicated by the asterisk in the diagram and the asterisk in the output below.

```
$ ifconfig ie0
IE0: flags=8000c43<UP,BROADCAST,RUNNING,MULTICAST,SIMPLEX>
     failSAFE IP Addresses:
        inet 10.10.10.2 netmask ffffff00 broadcast 10.10.10.255 (on HUFFLE IE1)
        inet 10.10.10.3 netmask ffffff00 broadcast 10.10.10.255 (on SLYTHE IE0)
        inet 10.10.10.4 netmask ffffff00 broadcast 10.10.10.255 (on SLYTHE IE1)
   *inet 10.10.10.1 netmask ffffff00 broadcast 10.10.10.255
```

Consider the situation where RAVENC is booted after HUFFLE starts TCP/IP Services. Before TCP/IP Services is started on RAVENC, all addresses are active on node HUFFLE. For instance, the figure above may become:



Network: **10.10.10/24**

Note that this figure shows the *.3* and *.4* addresses being distributed among the interfaces on HUFFLE. In practice, this is indeterminate.

When TCP/IP Services is started on node RAVENC, it requests that its home addresses be returned. In this example, the *.3* and *.4* addresses have their home interface on RAVENC, so RAVENC requests that HUFFLE release the *.3* and *.4* addresses so that RAVENC can assign them.

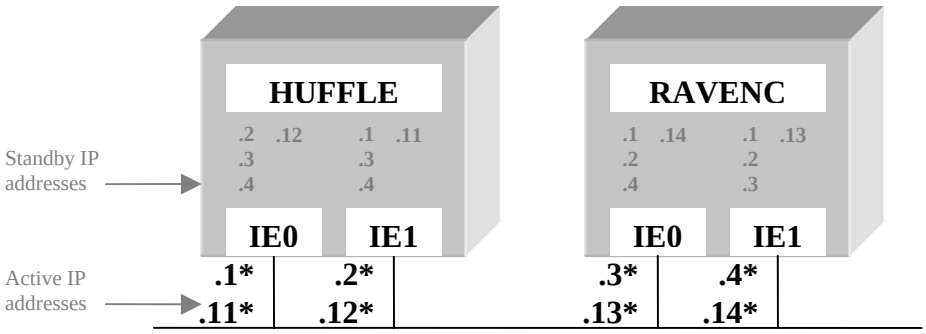
However, node HUFFLE will only release these addresses provided it does not have any outstanding connections to them. This situation could result in node RAVENC being started without any IP addresses being configured. To avoid this, only configure primary addresses with standby interfaces on the *same* node. See Example 3 for an alternate configuration that avoids this problem.

Example 3 – Preferred failSAFE IP Configuration – Putting it all together

Greater demands in availability of IP addresses require reconsideration of how addresses are assigned. One possibility is to create private maintenance IP addresses as primary addresses, and public IP address as aliases[7]. The disadvantage of this is that more IP addresses are required for the dedicated maintenance addresses. The advantage is that there is greater control and flexibility over address assignment. The concept of tying an address to a specific interface becomes less of a concern.

For instance, to build upon the previous example, private maintenance addresses could be assigned as primary addresses, and the public addresses would be configured as aliases, where each alias has at most one home interface. Consider the private primary addresses to be *.11, .12, .13,* and *.14*, and the public aliases to be *.1, .2, .3, and .4,* as shown below.

Note that the private primary addresses have standby addresses configured on the same node only. For example, on node HUFFLE, *.11* and *.12* are configured on both interfaces, but they are not configured on node RAVENC. The public alias addresses (*.1, .2, .3, and .4*) have addresses configured on each interface across both nodes. The asterisk beside these denotes the address's home interface. Thus, IE0 on node HUFFLE is the home interface for the alias 10.10.10.1.



Network: **10.10.10/24**

[7] For the purpose of this example, consider the private maintenance addresses to be known only to the network administrator, whereas the public addresses are well-known and provide connectivity to well-known services.

To configure this, create the primary IP addresses using the TCPIP$CONFIG procedure, and create the alias addresses using *ifconfig*. For example, add the following lines to the TCPIP$SYSTARTUP.COM procedure:

On node HUFFLE:

```
$! Configure home aliases
$ ifconfig ie0 home alias 10.10.10.1/24
$ ifconfig ie1 home alias 10.10.10.2/24

$! Configure IE0 aliases (presumes .12 primary was created via TCPIP$CONFIG)
$ ifconfig ie0 alias aliaslist 10.10.10.2-4,12/24

                                                    created via TCPIP$CONFIG)
$ ifconfig ie1 alias aliaslist 10.10.10.1,3,4,11/24
```

On node RAVENC:

```
$! Configure home aliases
$ ifconfig ie0 home alias 10.10.10.3/24
$ ifconfig ie1 home alias 10.10.10.4/24

$! Configure IE0 aliases (presumes .14 primary was created via TCPIP$CONFIG)
$ ifconfig ie0 alias aliaslist 10.10.10.1,2,4,14/24

                                                     reated via TCPIP$CONFIG)
$ ifconfig ie1 alias aliaslist 10.10.10.1-3,13/24
```

IP Cluster Alias

The IP Cluster Alias provides a subset of the functionality provided by failSAFE IP (see failSAFE IP), and as such has been superseded by failSAFE IP. However, failSAFE IP is introduced in TCP/IP Services Version 5.4, whereas the IP Cluster Alias was introduced in UCX Version1.0. It is recommended that existing users of the IP Cluster Alias update their configuration to use failSAFE IP.

Introduction to IP Cluster Alias

In an OpenVMS cluster, there may be a single IP address designated to represent selected cluster members. This address is known as the IP Cluster Alias address. Each interface still has its own unique IP address while the IP Cluster Alias is an additional address that can be active on only one interface in the cluster at a time. The node holding the address is designated as the *Cluster Impersonator* and as such will field all connections to the Alias address. In the event of a failure, the IP Cluster Alias address will be reassigned to one of the remaining cluster members interfaces.

Note that the functionality provided by the IP Cluster Alias is a subset of that provided by failSAFE IP. IP Cluster Alias is supported for compatibility.

IP Cluster Alias Configuration Requirements

Configuring the IP Cluster Alias requires that the node be an active member of an OpenVMS cluster at the time TCP/IP Services is configured using the TCPIP$CONFIG procedure. The configuration procedure will detect this scenario and when configuring the interfaces, will ask if a cluster address should be assigned.

Only one interface in the cluster can hold the cluster alias address at any time. It is recommended that the cluster alias address be configured in the same subnet as the unique interface addresses. This ensures broadcast traffic to the subnet containing the interfaces will also appear on the IP Cluster Alias address.

Detectable Failures

The types of failures that are detected include:

- Shutting the interface down using TCP/IP management commands
- Shutting down TCP/IP Services
- Shutting down the node

Application

This form of failover provides high availability for incoming connections. In the event of a failure the IP Cluster Alias migrates across nodes. Existing TCP connections will abort and need to be reestablished. Connectionless protocols, such as UDP, will be unaffected.

There is no load-balancing across nodes with this mechanism. The cluster impersonator fields all incoming connections. Outgoing connections do not make use of the IP Cluster Alias. It is best suited to UDP applications like NFS, or for maintaining a high availability IP address where load-balancing of incoming connections is not a priority.

Management Utilities

To identify the node currently acting as the impersonator, enter the following command:

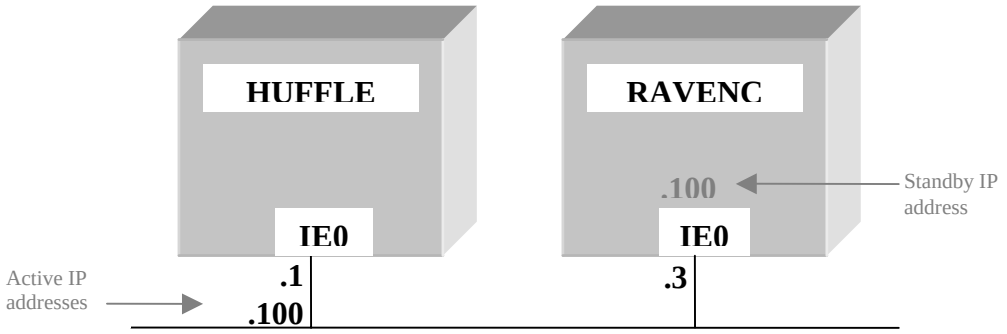
```
$ TCPIP SHOW INTERFACE/CLUSTER
```

The node acting as the impersonator will be labeled as "*Cluster Impersonator*". This needs to be performed on each node in the cluster. An easy way to do this is with the SYSMAN utility:

```
$ MCR SYSMAN
SYSMAN> SET ENVIRONMENT/CLUSTER
SYSMAN> DO PIPE TCPIP SHOW INTERFACE/CLUSTER | SEARCH SYS$PIPE IMPERSONATOR
```

Example

In Figure 3, the IP Cluster Alias address is designated as, 10.10.10.100. Node HUFFLE is currently the impersonator, and RAVENC is the standby node. In the event HUFFLE is shut down, then RAVENC will assume the IP Cluster Alias address and become known as the impersonator. New traffic to the IP Cluster Alias will go through RAVENC. Note that each node still has its own unique interface address, 10.10.10.1 and 10.10.10.3. The alias address 10.10.10.100 can be active on only one of the nodes at any time.



Network: **10.10.10/24**

Figure 3 - IP Cluster Alias Configuration

Replacing the IP Cluster Alias with the failSAFE IP mechanism requires that the IP Cluster Alias first be deleted and the failSAFE IP address created. For instance, executing the following command on each node would delete the IP Cluster Alias:

```
$ tcpip set configuration interface ie0/nocluster  ! remove from config file
$ tcpip set interface ie0/nocluster                ! remove from active system
```

Now configure the 10.10.10.100 address on each node with a command similar to:

```
$ tcpip set configuration interface iea0 /host=10.10.10.100 /net=255.255.255.0
$ tcpip set interface iea0 /host=10.10.10.100 /net=255.255.255.0
```

DNS Alias with Load Broker and Metric Server

This solution provides high availability of the DNS alias by dynamically updating the alias name with the list of most-available IP addresses associated with that alias name. It requires a name server that supports dynamic updates, and the cooperation of DNS administrators to allow your Load Broker to dynamically update their databases[8].

Introduction to DNS Alias

The Load Broker polls nodes for their metric values and dynamically updates the DNS alias with the list of least loaded IP addresses. In this way, whenever a remote host requests a DNS name lookup, it will be presented with the list of IP addresses associated with the least loaded addresses. If a node does not respond with a metric value after 3 attempts, the Load Broker will remove that node's IP addresses from the DNS alias.

[8] Rather than convincing your central DNS administrators to allow you to dynamically update their DNS repository, it may be easier to have them delegate a sub-domain that you administer.

DNS Alias Configuration Requirements

The DNS Alias configuration requires a DNS server that supports dynamic updates and the cooperation of DNS administrators to allow your Load Broker to send dynamic updates to their database. The Load Broker is typically configured on a separate host and located in the network path used by the clients. The DNS Alias is comprised of a list of IP addresses. There is no requirement for these addresses to appear on OpenVMS clustered nodes.

The DNS Alias with the Load Broker and Metric Server are described in detail in the following subsections.

DNS Alias

The DNS Alias, by itself, does not provide high availability. This section provides an overview of the DNS alias and how it is updated by the Load Broker.

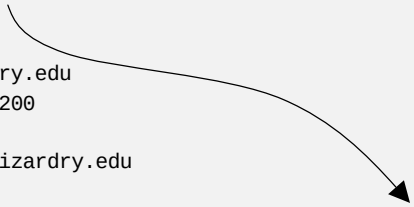
The Domain Name System provides a distributed repository for mapping between DNS alias names and IP addresses. In the repository, a single DNS Alias name may be associated with multiple IP addresses. Each time the DNS server is queried for an alias name, the list of IP addresses associated with that name is returned. That list is rotated in a round-robin fashion for each request, so that subsequent requests will return the list with a different IP address at the top of the list. Since applications typically choose the first IP address in the list, the round-robin feature provides load sharing, but not load balancing, across the list of IP addresses.

For example, consider the DNS Alias name *"hogwarts"* with four IP addresses associated with it. The IP addresses may all be associated with the same node, or the addresses may be spread across multiple nodes. The DNS entry in the forward lookup database may be:

```
hogwarts        IN      A       10.10.10.1
                IN      A       10.10.10.2
                IN      A       10.10.10.3
                IN      A       10.10.10.4
```

The first time a client queries the DNS server for the name *"hogwarts"*, the address list returned will be ordered as *(10.10.10.1, 10.10.10.2, 10.10.10.3, 10.10.10.4).* The client will use the first address in the list, and so connect to the 10.10.10.1 address. The next DNS query for "*hogwarts"* will result in the list being returned to the client as *(10.10.10.2, 10.10.10.3, 10.10.10.4, 10.10.10.1)*. This client will once again use the first address in the list and so connect to the 10.10.10.2 address. The pattern will continue for subsequent DNS requests. This round-robin effect can be observed with repeated queries using the *nslookup* utility. For example, notice the IP address list is rotated in the second query:

```
$ nslookup hogwarts
Server:  ns1.wizardry.edu
Address:  10.10.10.200

Name:    hogwarts.wizardry.edu

Addresses:  10.10.10.1, 10.10.10.2, 10.10.10.3, 10.10.10.4

$ nslookup hogwarts
Server:  ns1.wizardry.edu
Address:  10.10.10.200

Name:    hogwarts.wizardry.edu

Addresses:  10.10.10.2, 10.10.10.3, 10.10.10.4, 10.10.10.1
```



In the event of a failure where an IP address becomes unavailable, DNS will continue to dutifully answer queries and rotate the list of IP addresses. Each time the failed IP address is at the top of the list, a client application will not be able to connect, and as a result there will appear to be intermittent connection failures. To guard against this type of failure, the Load Broker and Metric Server may be used[9]. The Load Broker will remove any unresponsive IP addresses from the DNS repository and so provide high availability of the DNS alias name. The round-robin function will continue to share the load across each of the available IP addresses. The Load Broker may be further configured to maintain a maximum number of IP addresses in the DNS alias and it will update DNS with the IP addresses that return the more favorable metric.

The Load Broker and Metric server are discussed in the next section.

Load Broker and Metric Server

The Load Broker is configured to monitor selected IP addresses on hosts where the Metric Server is enabled. The Metric Server responds with a metric value, indicating the load of that machine. If there is no response from a Metric Server after 3 attempts, then the Load Broker will dynamically update the DNS repository excluding the unresponsive IP address. In addition, when a node becomes heavily loaded, it may be replaced in the list by a node with a more favorable metric.

Detectable Failures

The types of failures that are detected include:

- Shutting the interface down using TCP/IP management commands
- Shutting down TCP/IP Services
- Shutting down the node
- Path lost between Load Broker and Metric Server

Application

The Load Broker benefits incoming connections only. It has the additional benefit that the IP addresses with the more favorable metric will be associated with the DNS Alias. This configuration is suited to maintaining high availability and optimum performance for a well-known service that is

[9] failSAFE IP may also be used to provide high availability of IP addresses across clustered nodes. Use the Load Broker in situations where it is not desirable for an IP address to failover across clustered nodes. Load Broker does not require IP addresses to be within an OpenVMS cluster.

distributed across multiple nodes. There is no requirement for the nodes to be part of an OpenVMS Cluster. In order to be effective, the client application must retranslate the host name of the server following a failure. Applications that do not repeat the DNS query (such as many NFS clients) will never see the updated list of alias addresses.

Management Utilities

The metric value for any host on a LAN can be displayed with the *metricview* utility. For example:

```
$ @tcpip$define_commands
$ metricview
Host                                                    Rating
----                                                    ------
10.10.10.1      huffle-e0                                 136
10.10.10.3      ravenc-e0                                  51
```

The more favorable node is represented by a larger metric value.

Example

Placement of the Load Broker node is important when configuring the network. Since it can detect connectivity between the Load Broker and Metric Server, it is best placed in the same network path used by the clients that access the services. Similarly, the Load Broker should not be configured on a machine running the Metric Server, since it will always report full connectivity to that Metric Server, regardless of the state of the network paths to the clients. An example Load Broker configuration file for is shown below:

```
cluster “hogwarts.wizardy.edu”
{
        dns-ttl 45;
        dns-refresh 31;
        masters { 10.10.10.200; };
        polling-interval 10;
        max-members 3;
        members { 10.10.10.1; 10.10.10.2; 10.10.10.3; 10.10.10.4; };
        failover 10.10.0.150;
};
```

This configuration file indicates that four IP addresses participate in the load-balancing, but only three of these addresses, (*max-members*), will participate in the DNS alias, thus excluding the node with the least favored metric from the DNS alias. Note that Load Broker will only dynamically update the DNS alias if the alias must be modified with a different set of addresses. Load Broker does not compare the order of the DNS alias list with its current metric order because DNS will continue to adjust the order, providing load sharing amongst the DNS alias members.

The Load Broker will poll each Metric Server every 10 seconds (*polling-interval*). If a Metric Server does not respond after 3 polling intervals (30 seconds), then on the next *dns-refresh* timeout (31 seconds) the bad IP address will be excluded from the next dynamic update. The *dns-ttl* will force intermediate name-servers that cache the results of a DNS-query to time out this entry every 45

seconds. In this way, if a failure occurs, a client will take 45 seconds at most to retry a connection before DNS queries the primary server for the new DNS alias list.

This form of high availability is applicable to new incoming connections to a well-known service distributed amongst participating nodes. If a failure occurs during a connection, that connection will need to close and a new connection be established.

Summary

High availability of the network requires careful consideration of the network environment and understanding of the failures that must be protected against. As a result, one or more high availability solutions may be required. failSAFE IP provides high availability of IP addresses for both incoming and outgoing new connections as well as existing traffic flow. The DNS Alias with Load Broker and Metric Server provides high availability of a DNS Alias name and so benefits incoming connections only. The IP Cluster Alias has been superseded by failSAFE IP. LAN Failover provides high availability of a hardware MAC address and benefits all LAN protocols. LAN Failover is required for LAN protocols that do not provide a failover solution, such as LAT. Protocols such as DECnet-Plus, SCS, and IP implement a failover solution.

The various high availability solutions described in this paper require minimal configuration and management. However, since they are protecting vital parts of the network, any solution must be validated prior to being relied upon in a production environment.

For more information

For more information, contact HP OpenVMS products at http://www.hp.com/go/openvms.

DCPI for OpenVMS
a Technical Introduction to a "System Microscope "

By Anders Johansson

Anders is a principal software engineer in the kernel tools team of the OpenVMS development engineering group and the project leader for DCPI on OpenVMS. Other projects currently include development work for OpenVMS I64 in the areas of LIBRTL and SDA. Anders, who has been with the company for 16 years and is based in Stockholm, Sweden, is also an OpenVMS Ambassador for Sweden.

Introduction

How many times have you wondered how your application executes, which parts of the system are used most often, or where you might have bottlenecks in the code? Several products are available to measure performance on OpenVMS systems. Most of these products use software performance counters, which have certain limitations.

HP (Digital) Continuous Profiling Infrastructure, DCPI for OpenVMS, uses the hardware performance counters of the Alpha chip to overcome these limitations. DCPI provides a "fine-grained" view of the system. During the analysis of data, DCPI produces information ranging from the time spent in individual executable images to which instructions are executed within an executable image. It also provides insight into where stalls and instruction or data cache misses occur, and so on. A normal sampling frequency for DCPI on a 600MHz Alpha is 10000 samples per second on every CPU in the system. DCPI does this with a minimum CPU overhead -- usually below 5% of the total available CPU time.

DCPI for OpenVMS, Some Background Information

DCPI began as a research project to find out how programs could be optimized to run faster on the Alpha processor. This project, called "Where have all the cycles gone?" resulted in the first version of DCPI (available on Tru64 Unix), and "SRC Technical Note 1997-016A," which is available at the following web site:

http://gatekeeper.research.compaq.com/pub/DEC/SRC/technical-notes/SRC-1997-016a-html/

An investigation into the feasibility of porting DCPI to OpenVMS started in late 1999; most of the porting work was completed during 2000 and early 2001. DCPI was then used within OpenVMS engineering to pinpoint performance problems. Early in 2002, a version of DCPI for OpenVMS became available externally; it is downloadable from the following OpenVMS web site as an "advanced development kit" under field test license terms:

http://h71000.www7.hp.com/openvms/products/dcpi/

DCPI provides the fundamentals for instruction-level system profiling. In general, DCPI does not require any modifications to the code being profiled, because it is driven by the hardware performance counters on the Alpha chip itself. Data is collected on the entire system, including user and third-party applications, runtime libraries, device drivers, the VMS executive itself, and so on.

The collected data can then be used for program, routine, and instruction-level profiling of the environment.

DCPI for OpenVMS. How Does It Work?

DCPI consists of the following major components:

1. A data collection subsystem, which includes a device driver, a "daemon program," and a daemon control program (dcpictl) for user intervention with the DCPI daemon.

2. Data analysis tools that are used to break down the collected data into image/routine/code-line/instruction profiles.

3. A special version of the OpenVMS debugger shareable image.

4. A shareable image containing the API for data collection of dynamic code.

The role of the DCPI device driver (DCPI$DRIVER) is to interface with the Alpha chip registers that control the hardware performance monitoring of the chip and to handle the performance monitor interrupts generated when the Alpha chip performance counters overflow. The interrupt handler stores the data acquired at each interrupt into resident memory. The data stored by the driver consists of type of event, PC, and PID.

The DCPI Daemon, which runs as an interactive process on OpenVMS, controls the kind of monitoring that is performed and also starts and stops the data collection. However, the main task of the DCPI daemon during the data collection is to:

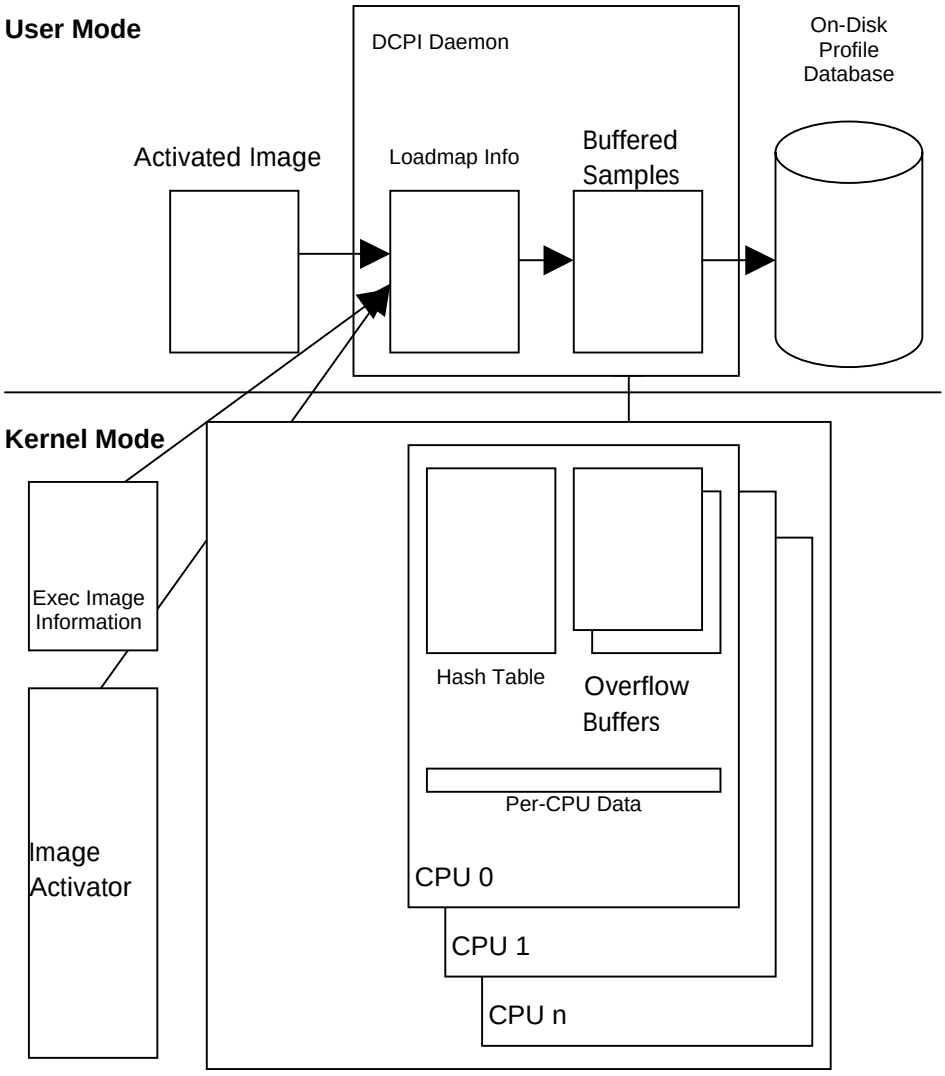
- Read the data out of the driver buffers
- Map the event/PC/PID into an Imagename/Image-offset pair
- Store it into on-disk profiles for later analysis

To do this mapping, the DCPI daemon must have an in-memory map of the activated images within every process on the system and also a map of the exec loaded image list. The DCPI daemon builds a "loadmap" during its initialization, including the exec loaded image list and the activated images in all the processes currently running on the system. Furthermore, to do the mapping on a running system correctly, the DCPI daemon must track all the subsequent image activations in all the active processes on the system. This image activation tracking is done using "image activator hooks" that are available in the OpenVMS operating system starting with OpenVMS V7.3. This tracking is implemented by means of a mailbox interface between the OpenVMS image activator and the DCPI daemon. In this interface, the image activator provides detailed information to the DCPI daemon about all image activations on the system.

The data analysis tools provide various views of the collected data. For these tools to provide routine names and source code correlations to the profile data, DCPI uses its own version of the OpenVMS debugger shareable image (DCPI$DBGSHR.EXE). To perform routine/source correlations, DCPI also needs images with debug information (LINK/DEBUG) or debug symbol files (.DSF files) for the running images (LINK/DSF). Therefore, while the data collection subsystem

collects data on ALL images running on the system, the analysis tools require debug symbol information to perform in-depth analysis of the collected data.

The following figure illustrates DCPI data collection principles.



DCPI Data Collection Principles

Alpha Chip Performance Monitoring

Two different methods exist for collecting performance data using the hardware performance counters of the Alpha chip:

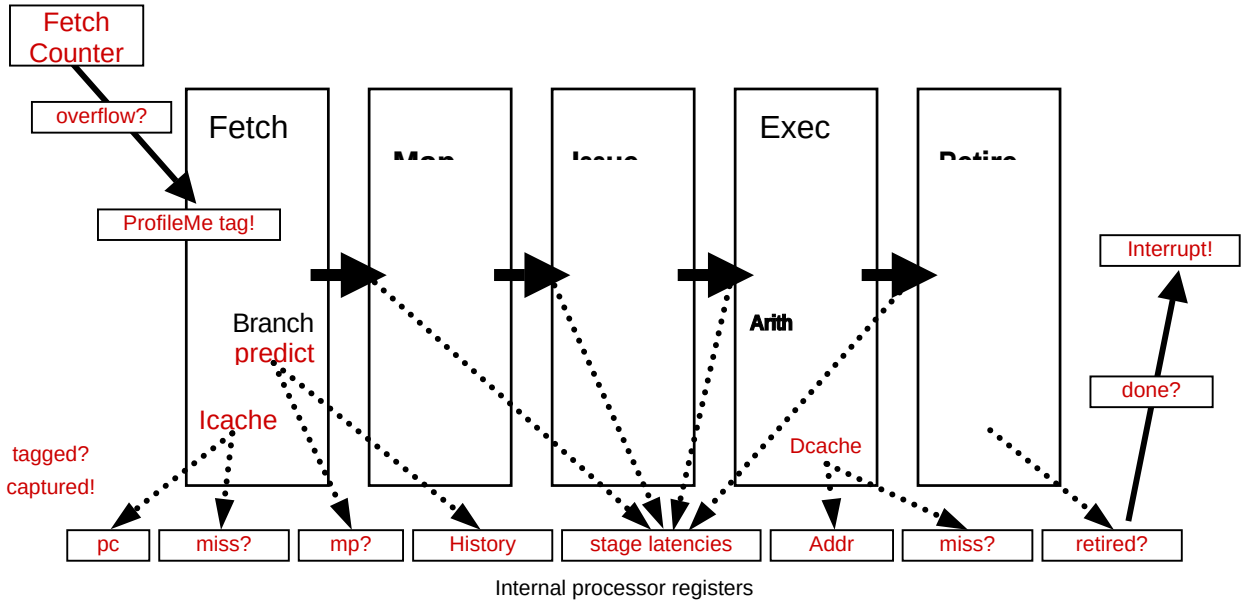
- Aggregate events
  This method is available to a varying degree on all existing Alpha chips. Using this method, DCPI sets a value (sampling period) in the performance monitor control register on the Alpha processor and also specifies which event to sample. Then, each time the event occurs, for example, a "CPU cycle has executed," this event will be counted. When the number of events has reached the specified sampling period, an interrupt is generated, and DCPI stores the PC, PID and event type. For example: If DCPI is sampling the CPU cycles event with a period of 63488, the Alpha chip generates an interrupt every 63488 cycles.

  Although collecting aggregate events is generally a far better method for obtaining reliable profiling data than collecting software performance counters, collecting aggregate events has certain disadvantages This method relies on the instruction that is active at the time of the interrupt being, in fact, the instruction that generated it -- in other words, it counts on the performance monitor interrupts being precise. This is, however, not always the case. Only a few of the performance monitor events produce a precise interrupt. Also, on recent processors -- EV6 and later -- none of these interrupts are precise. This might appear to be an important problem. However, even though the interrupts are imprecise, they are fairly predictable, which DCPI takes into account.

  Another problem with this method is that it allows for "blind spots" -- for example, any code executing at or above IPL29 will not be profiled, because the performance monitor interrupts are at IPL29. Such blind spots also include all the PAL code on Alpha, since the PAL code runs with interrupts turned off. In those cases, the performance monitor interrupt takes place on the first instruction after the PAL call, or when IPL drops below 29.

- ProfileMe
  This method, which is available on EV67 and upward, is in many ways superior to "aggregate events" ProfileMe uses some specific ProfileMe registers on the Alpha chip. When DCPI sets the period for ProfileMe, the CPU counts instruction fetches. When the period has passed (that is, when the instruction fetch counter overflows), the fetched instruction is tagged as an instruction to profile. Information about this instruction is then recorded into the ProfileMe registers throughout the execution phases of the instruction. When the instruction retires, the interrupt is generated. At this point, DCPI reads all the information out of the on-chip ProfileMe registers. This method of collecting performance data is much more reliable, and also provides a much more complete picture of how the different instructions perform.

The following figure shows the flow of events during ProfileMe sampling.



Data Analysis Tools on DCPI for OpenVMS

For the analysis tools to work, they require access to the exact images that were used when the data was collected. The reason is that the tools read the instructions directly from the image files, and the analysis becomes meaningless if the instructions read are not the instructions that were executed. A test is performed that verifies that the image analyzed is the same image as the one being profiled. This verification is performed by checking various fields in the image header and comparing them to what was stored in the DCPI profile database (dcpidb) during the data collection.

The analysis tools can perform breakdown analysis by image or by routine name. To do this successfully, the analysis tools require debug symbol information for the image analyzed. This requirement can be met in two ways:

- The image is linked /DEBUG, which might not be practical, because the image might be INSTALLed on the running system, which requires the image to be linked /NODEBUG

- The image is linked /NODEBUG/DSF, which creates a separate file (*imagename*.DSF) that contains all the debug symbol information for the image. Place this debug symbol file in the same directory as the image file itself. Another alternative is to place all debug symbol files in a separate location and define the logical name DBG$IMAGE_DSF_PATH to point to that directory.

The analysis tools provide slightly different views of the collected data. To perform complete profiling, you must use several of the analysis tools, which are described in the following table:

Tool	Description
DCPIPROF	The top-level analysis tool. It provides either a system-to-image breakdown of the collected samples, or an image-to-procedure breakdown of the collected samples. DCPIPROF is usually the first tool used to obtain an initial idea of the images in which the system is spending time. DCPIPROF is then used to obtain an initial idea of which routines within an image are spending the most time. DCPIPROF analysis works on data collected via the ProfileMe method and via aggregate events.
DCPILIST	The tool to use for detailed analysis within a procedure in a specified image. DCPILIST can correlate the collected samples to either source lines in the code, executed Alpha instructions or both. To do source code correlation, it also needs the actual source file of the analyzed routine. DCPILIST analysis works on data collected via the ProfileMe method and via aggregate events.
DCPICALC	Generates a control flow graph of the procedure or procedures specified for a given image. DCPICALC augments the graph with estimated execution frequencies of the basic blocks, cycle-per-instruction counts, and so on. DCPICALC works only on data collected via aggregate events.
DCPITOPSTALLS	Can be used to identify the instructions within the specified image or images that account for the most stalls. DCPITOPSTALLS only works on data collected via aggregate events
DCPIWHATCG	Generates the same type of control flow graph as DCPICALC, but instead of producing at the procedure level, it looks at different images. DCPIWHATCG works only on data collected via aggregate events.
DCPITOPS	Takes the output from a DCPICALC run and generates a PostScript™ representation of the execution of the image, for example using different font sizes to visualize the execution frequencies of the basic blocks. DCPITOPS only works on data collected via aggregate events.
DCPICAT	Presents the raw profile data in a human readable format. It is normally only used by the DCPI developer, but is included in the DCPI for OpenVMS kit for convenience. DCPICAT can handle all types of DCPI events.
DCPIDIFF	Compares a set of profiles and lists the differences between them. This can be very useful when looking at different test cases.

As indicated in the preceding table, several tools operate only on data that is sampled using aggregate events This somewhat limits the ease of analysis for ProfileMe data, but the richness of the ProfileMe data is sufficient to find all the causes without those tools. DCPICALC, DCPIWHATCG and DCPITOPSTALLS all use intimate knowledge of the Alpha CPU execution characteristics to apply qualified guesswork to find out where problems such as stalls occur. With ProfileMe, all data

is collected on each profiled instruction; therefore, the rationale behind not supporting the tools on ProfileMe is that because the data is collected by the hardware itself, it is much more reliable.

Also note that all the DCPI tools use a UNIX-style command syntax. Because adding DCL syntax to the tools would not add anything to the functionality of DCPI, the decision was made to omit the time-consuming effort of providing a DCL interface to the tools.

Profiling of Code Generated "on the Fly"

Some applications such as Java™ generate code on the fly and then execute it. Standard DCPI needs a persistent on-disk image for the analysis because it reads the instructions from that image during the analysis. When building its profile during the data collection, DCPI must also build a loaded image map to calculate image offsets, and so on. None of these exist for dynamically generated code.

DCPI for OpenVMS includes an API for informing the DCPI daemon about the generated code. This API also provides a way to generate Debug Symbol Table (DST) entries for the generated code. The generated code and its associated DSTs are then written to a persistent on-disk "pseudo image," which is used during the analysis. This is an area where DCPI for OpenVMS has evolved beyond the DCPI version available on Tru64 UNIX.

DCPI Usage

A typical sequence of commands to run DCPI data collection is the following:

dcpid, one or more *dcpictl* commands, and finally *dcpictl quit* to stop the data collection.

By using the DCPI daemon in conjunction with the DCPI driver, you first collect data into on-disk profiles, which are stored into epochs on disk. The epochs are the only means of applying a time-line to the DCPI data. This is very important because it is absolutely impossible to see which data in an epoch were collected during, for example, a peak period of the load. The typical recommendation of a way to obtain good results when using DCPI is to keep the load stable within an epoch, because this is the only way to know what is being profiled. Divide any run that includes ramp-up, ramp-down, peak, and low activity on the system into epochs to correctly determine which profiles came from which test case. The names of the profiles come from the GMT time when they were created.

Commands to manipulate epochs during the data collection are:

- *Dcpid* by default creates a new epoch in the current DCPI database. Using the switch *–epoch* on the command line while starting *dcpid* does not create a new epoch, but rather uses the most recent one in the DCPI database.

- *Dcpictl* is an interface to the DCPI daemon during the data collection. Ways to manipulate epochs include the following:

    - *Dcpictl flush,* which performs a user-initiated flush of the in-memory profile data of the DCPI daemon and DCPI driver, into the current epoch in the DCPI database (the logical

name DCPIDB points to the DCPI database). Flushing also occurs automatically throughout the data collection and into the current epoch.

- *Dcpictl epoch*, which flushes the in-memory profile data of the DCPI daemon and the DCPI driver into the current epoch, and then starts a new epoch.

Running the Data Collection

A typical way of starting the data collection is:

```
$ dcpid cmoveq$dka100:[dcpi.test]
dcpid: monitoring cycles
dcpid: monitoring imiss
dcpid: logging to comveq$dka100:[dcpi.test]dcpid-COMVEQ.log
```

Because the DCPI daemon runs as an interactive process on OpenVMS, you might want to use the following command to avoid locking up the terminal where *dcpid* is run:

```
$ spawn/nowait/input=nl: dcpid cmoveq$dka100:[dcpi.test]
%DCL-S-SPAWNED, process SYSTEM_187 spawned
dcpid: monitoring cycles
dcpid: monitoring imiss
dcpid: logging to comveq$dka100:[dcpi.test]dcpid-COMVEQ.log
```

On pre-EV67 processors, the default events to collect are *cycles* and *imis.* On EV67 and newer processors, the default events are *pm* (ProfileMe) and *cycles.*

To end the data collection, type the following command:

```
$ dcpictl quit
```

Analyzing the Data

After the data collection is completed (or during the data collection, if data has been flushed) you can then use *dcpiprof* to take an initial look at the collected profile data:

```
$ dcpiprof
dcpiprof: no images specified.  Printing totals for all images.
Column             Total  Period (for events)
------             -----  ------
cycles           1755906   65536
imiss              41991    4096

The numbers given below are the number of samples for each
listed event type or, for the ratio of two event types,
the ratio of the number of samples for the two event types.
============================================================
 cycles       %    cum% imiss       % image
1349002  76.83%  76.83%  8154  19.42% DISK$CMOVEQ_SYS:[VMS$COMMON.SYSLIB]DECC$SHR.EXE
 176821  10.07%  86.90%   919   2.19% DISK$CMOVEQ_SYS:[VMS$COMMON.SYSLIB]LIBRTL.EXE
  65432   3.73%  90.62%   426   1.01%
DISK$ALPHADEBUG1:[DEBUG.EVMSDEV.TST.TST]LOOPER.EXE;1
  45788   2.61%  93.23%  8651  20.60% SYS$SYSROOT:[SYS$LDR]SYSTEM_SYNCHRONIZATI
  27039   1.54%  94.77%  4598  10.95% SYS$SYSROOT:[SYS$LDR]SYSTEM_PRIMITIVES.EX
  16045   0.91%  95.68%   844   2.01%
DISK$CMOVEQ_SYS:[VMS$COMMON.SYSEXE]DCPI$DAEMON.EXE;2
   7727   0.44%  96.12%  1969   4.69% SYS$SYSROOT:[SYS$LDR]RMS.EXE;
   6993   0.40%  96.52%  2102   5.01% SYS$SYSROOT:[SYS$LDR]SYS$PEDRIVER.EXE;
   6741   0.38%  96.91%  1762   4.20% SYS$SYSROOT:[SYS$LDR]PROCESS_MANAGEMENT_M
   6587   0.38%  97.28%  1215   2.89% SYS$SYSROOT:[SYS$LDR]F11BXQP.EXE;
   5742   0.33%  97.61%  1079   2.57% SYS$SYSROOT:[SYS$LDR]SYS$BASE_IMAGE.EXE;
   5385   0.31%  97.92%  1434   3.42% SYS$SYSROOT:[SYS$LDR]SYS$EWDRIVER.EXE;
   5344   0.30%  98.22%  1371   3.26% SYS$SYSROOT:[SYS$LDR]IO_ROUTINES_MON.EXE;
```

```
   5015   0.29%  98.51%  1024   2.44% unknown$MYNODE
```

This first example shows the output of the top-level *dcpiprof* run. The next step is to decide which image is interesting, and use *dcpiprof* to look into that image.

```
$ dcpiprof DISK$ALPHADEBUG1:[DEBUG.EVMSDEV.TST.TST]LOOPER.EXE;1
Column             Total  Period (for events)
------             -----  ------
cycles             84210   65536
imiss                540    4096

The numbers shown below are the number of samples for each
listed event type or, for the ratio of two event types, the
ratio of the number of samples for the two event types.
============================================================
cycles       %    cum% imiss       % procedure                image
 65774  78.11%  78.11%   334  61.85% get_next_random          disk$alphadebug1..
 16892  20.06%  98.17%    93  17.22% analyze_samples          disk$alphadebug1..
  1543   1.83% 100.00%   112  20.74% collect_samples          disk$alphadebug1..
     1   0.00% 100.00%     1   0.19% main                     disk$alphadebug1.
```

Usually, you perform the next level of analysis by using *dcpilist* to look at the actual code lines / Alpha instructions that the samples are attributed to:

```
$ dcpilist –both -f dbg$tstevmsdev:[tst]looper.c get_next_random -
disk$alphadebug1:[debug.evmsdev.tst.tst]looper.exe
cycles imiss
     0     0           static int get_next_random (void)
     0     0           /*
     0     0           ** We want to get the next random number sample.
     0     0           ** The samples are SAMPLE_ITERATIONS calls apart.
     0     0           */
     0     0           {
     0     0            long int   i;
     0     0            int        sample;
    21     0
    21     0  0x2045c         STL             R31,#X000C(FP)
 42355   174             i = 0;
  3875    27  0x20460         LDL             R1,#X000C(FP)
 11536    31  0x20464         LDA             R1,#XFF9C(R1)
  3923    19  0x20468         LDL             R0,#X000C(FP)
 12001    50  0x2046c         ADDL            R0,#X01,R0
  3764    13  0x20470         STL             R0,#X000C(FP)
  3772    21  0x20474         BGE             R1,#X000006
                          . . . .
  3484    13  0x2048c         BR              R31,#XFFFFF4
 22013                   while (i++ < SAMPLE_ITERATIONS)
  4248    19  0x20478         BIS             R31,R31,R25
    37     0  0x2047c         LDQ             R26,#X0028(R2)
  3689    17  0x20480         LDQ             R27,#X0030(R2)
  7325    28  0x20484         JSR             R26,(R26)
  6714    38  0x20488         STL             R0,#X0008(FP)
     0     0             sample = rand ();
   195     2  0x20490         LDL             R0,#X0008(FP)
     0     0  0x20494         BIS             R31,FP,SP
    40     0  0x20498         LDQ             R26,#X0010(FP)
    44     1  0x2049c         LDQ             R2,#X0018(FP)
    85    37  0x204a0         LDQ             FP,#X0020(FP)
     0     0  0x204a4         LDA             SP,#X0030(SP)
     0     0            return (sample);
$
```

The real challenge with this detailed information is to understand why the system executes as it does, and why certain routines are used as often as they are. Then you need to look into the routines that need to be used frequently if they appear to have performance problems.

In the preceding examples, the top image is not LOOPER.EXE -- which might be the obvious guess, because a *monitor system* would show that the process running LOOPER.EXE is the top CPU consumer. The top image is, rather, the DECC runtime library. The *rand()* call in the *get_next_random()* routine calls into the DECC runtime library, which most likely also uses one or more routines in LIBRTL, thus having those two as top images. This example shows, in a fairly

simple way, one reason why getting to the root cause of a problem might be a challenge. The cause of the "problems" seen here is LOOPER.EXE, because it makes excessive calls into the DECC runtime library. From the initial *dcpiprof,* believing that the DECC runtime library has problems is easy. Although this example is simplistic, it demonstrates that further analysis is often needed to find the root cause of the system behavior.

Some Basic Hints for DCPI Analysis

A few hints for DCPI analysis follow. To perform a full analysis of an application requires a very good understanding of the application itself.

- As indicated above, the DCPI notion of a time-line is called an epoch. No way of "time-stamping" the individual samples exists, other than creating a new epoch. To obtain predictable results when using DCPI, you must have as a goal a stable load throughout an epoch. No way exists to analyze less than an epoch afterwards. Creation of epochs is done during data collection.

- DCPI samples the whole OpenVMS system, not just the "interesting" program. Calls that are made into shareable images reflect the sum of all the calls made by ALL programs currently running on the system. A way to break this up into images run within different processes is to start the data collection with *$dcpid –bypid 'imagename'.*

- Be careful when drawing conclusions. As the above example illustrates, drawing incorrect conclusions is quite easy. Also, a high number of samples in an image/routine might not mean that this image/routine has any performance problems. A high number of samples means only that the image/routine was used frequently. You need to analyze further to find the root cause of the sample count.

- The DCPI daemon, which is a central piece of the DCPI data collector, is a normal user process. On a heavily loaded system, the DCPI daemon could be starved for CPU time, which might be seen as "dropped samples" in the DCPI daemon log file (dcpid-*nodename*.log) in the DCPI database. The DCPI database is defined by the logical name *dcpidb.* In some cases, it is important to start the data collection with a *dcpid –nice 'priority'* to increase the priority of the DCPI daemon process.

- The DCPI daemon scans all available processes during its initialization, to build an image map for the images in each process. On a system with many processes and a high load, this initial scan can take a considerable amount of time. If the system is running OpenVMS V7.3 or higher, starting the DCPI daemon ahead of time, when the system is relatively idle greatly reduces the time of the DCPI daemon initialization.

- All activity on the system, or lack of activity, is reflected in the collected data. If the system is idle, nearly 100% of the time will be attributed to SCH$IDLE() in PROCESS_MANAGEMENT.EXE. This exec loaded image contains other important code; therefore, a high number of samples might indicate something else. If the percentage of samples in PROCESS_MANAGEMENT.EXE is similar to the percentage of idle time, it is fairly safe to make this assumption.

- Try to minimize the noise level during the data collection by stopping unused CPUs or unneeded software, or both. Do this only during a second-level data collection, when you are narrowing down the causes of a problem.

A high number of cycles might be normal for the images or routines seen during analysis. When using ProfileMe, to find images or routines with possible problems, look at the RETIRED/CYCLES ratio of the routine. Ideally, the Alpha chip is capable of sustaining 4 instructions per cycle. Any routine with a ratio of 3 is probably impossible to improve, while a routine with a ratio below 1 is a good suspect for a routine with performance problems.

RMS Performance: Duplicate key chains

Hein van den Heuvel

Overview

Does your application use RMS Indexed files? Do you know what a SIDR is? Do you know what a duplicate key chain is? You probably should, since SIDRs and duplicate key chains can cause thousands of read I/Os as the result of a single record insert. With that, they can have a tremendous impact not just on the application doing the insert, but also on total system performance. Application managers, of course, notice a slowdown over time, and all too often they solve that by throwing more hardware at the problem. But what if you already have the biggest box on the market? Modest file tuning and a convert can help avoid all those read I/Os and restore performance. I have yet to investigate an RMS application that did not have this duplicate key chain problem. Maybe this is because I get called in only for bad cases, or because indeed so many applications have this problem, at least to some degree.

Problem statement

In recent years, HP systems engineers have investigated and improved several applications in large commercial systems where more than half of the resources for an entire system were wasted by updating duplicate key chains. In one case, a simple CONVERT of a single indexed file changed application end-user response time from several minutes to subseconds. In another case, the total system I/O rate was reduced from 1500 I/Os per second to 200 I/Os per second, all without changing application functionality. Why did this happen? Because of duplicate key chains.

Duplicate key chains are a (long) series of (single) linked RMS data buckets containing records all with the same key value and identified only by a single index entry. (The next section clarifies this definition.) Although the problems described in this paper can occur with primary keys, in real-life applications they typically occur with secondary keys. Therefore, the illustrations in this paper show duplicate key chains in secondary keys. It may be that application designers tend to pick unique or nearly unique keys for the primary key.

To establish a frame of reference, the following section describes the internals of an RMS indexed file. Subsequent sections describe how to identify the problem and suggest a number of possible solutions. You will see that some solutions are very easy to implement and can be very rewarding.

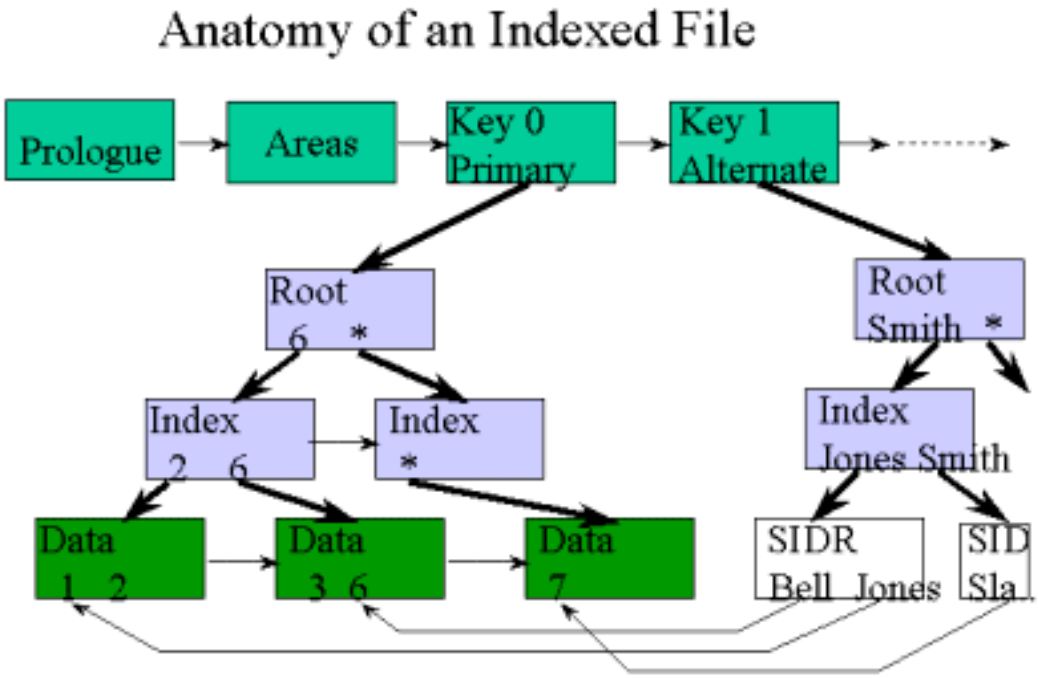
For additional information about indexed files and tuning, refer to the *Guide to OpenVMS File Applications* in the OpenVMS documentation set.

Overview of RMS Indexed File Internals

RMS stores user data records (UDRs) in primary-key order in buckets. Buckets are the unit of I/O to and from the file. Typically, a bucket contains 5 to 50 records. Records cannot cross bucket boundaries. If an entire record does not fit in a bucket, then a new bucket is added to hold the record. This process is called bucket split. You can identify records both by their key and by a record file address (RFA). The latter consists of the starting virtual block number (4-byte VBN) of the bucket in which the record is stored and the record's ID (2 bytes). The index structure is a balanced b-tree with pairs of key values and VBN addresses.

For secondary keys (referred to in this article as alternate keys), the data records pointed to by their key structure are called secondary index data records (SIDRs). A SIDR consists of a key value (optionally compressed) and an array of one or more record retrieval vectors (RRVs). If your application allows duplicates for the key in question, then there will be one RRV for each duplicate value that a key has. Each RRV is 7 bytes in size and consists of a flag byte plus an RFA pointing to the UDR.

The following figure illustrates an indexed file, a number as primary key, and a name as first alternate key.



Here's where the trouble starts

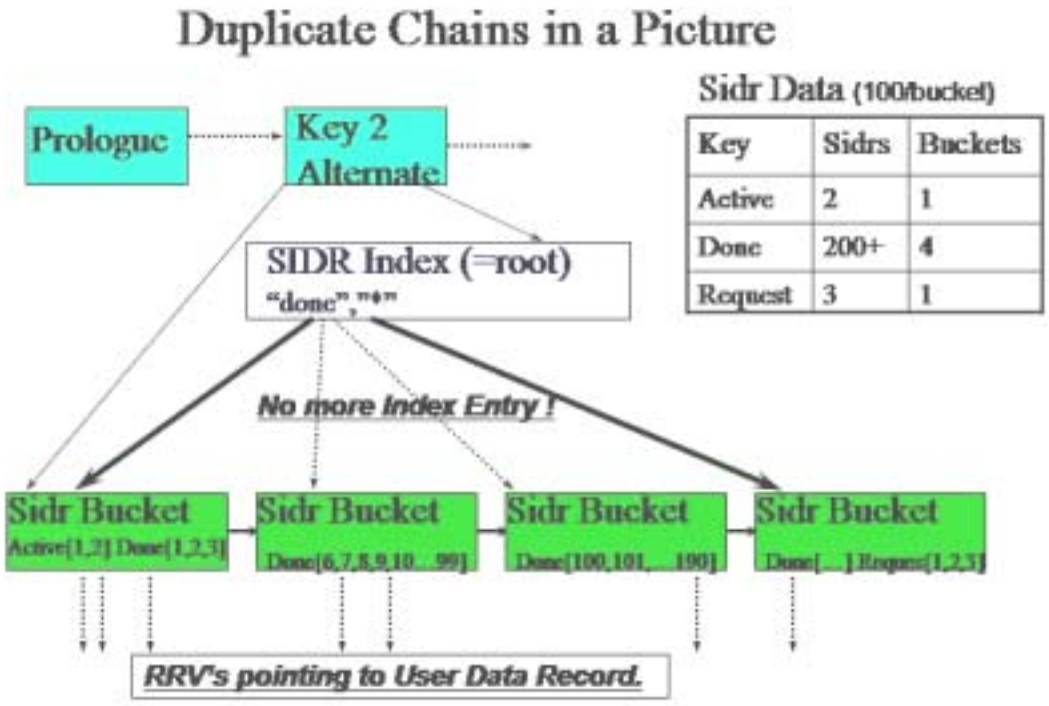
In addition to the file internals described here, RMS follows three rules that work very well in general but that can add up to serious performance problems in certain situations.

- Duplicate key values are to be added in order of arrival.
- There is only one index entry for a given key value that points to the first bucket that contains a record with that key value.
- If a new, duplicate value does not fit in the target bucket, then a new record is created in a new bucket. That bucket is pointed to by the old target bucket by using the next VBN field in the old bucket header.

What are the implications? Suppose you have an alternate key on an item file to indicate a status. That status can be 'Request', 'Active', or 'Done'. Every new item is inserted into the file with a status of 'Request'. Appropriately, according to rule 1, the item is added to the end of the array of 'Request' item. Next, the item gets processed, and the status key is updated to 'Active'. Eventually, all items are updated to the status 'Done'. Still, according to rule 1, as an item status is updated to 'Done', it is placed at the end of the 'Done' array, keeping the first item that was ever 'Done' as the first RRV in the SIDR.

Now, for the sake of the illustration, assume a SIDR can hold up to 100 RRVs. (In actual files, this is likely to be in the 150 – 1500 range.) The SIDR with the RRVs for the first few 'Done records fits in the same bucket as the SIDR for the 'Active' key. When the number of RRVs reaches more than a few hundred, multiple buckets are needed. When the status for item 200 is updated to 'Done', RMS walks down the index to find the first 'Done' record. RMS determines that this is the first, but not the last, SIDR record for the target key, and it reads the next bucket. RMS continues to read buckets until it reaches the final bucket. It then adds the RRV for item 200 to that SIDR and writes that bucket out to the file. This is the crux of the problem. The series of linked buckets, all with SIDRs for the same key value, is called a duplicate key chain. The system will need lots of read I/Os to perform the single write that it sets out to do. The following figure summarizes this layout.

As long as RMS needs to read just a few more buckets to find the last SIDR for a key, any additional I/Os don't cause a problem. However, when there are millions of Done records with thousands of continuation buckets to store their pointers, it starts to hurt. These thousands of I/Os will bring any system to its knees, no matter how big the box.



Detecting a duplicate key chain problem

Monitor I/O rates

The biggest indicator that an application may have a problem with duplicate key chains is excessive I/O rates for seemingly basic functions. For a multiple-key indexed file insert, you can expect 2 – 4 reads per key and 1 or 2 writes per key, for a total of 10 – 20 I/Os for a typical file. If you observe an average of 100 I/Os or more per insert, then you need an explanation and a fix, preferably an easy fix.

A hot-file tool, combined with the SET FILE /STAT command and the standard MONITOR RMS command can help identify the files to analyze. You should also check out the rms_stats freeware in the RMS tools directory with the OpenVMS Freeware. The rms_stats software reports I/Os per record operation for files with RMS statistics enabled.

Analyze files

A tell-tale sign for the duplicate key chain performance problem is the presence of very short alternate keys (1- 5 bytes) in files with large numbers of records. For example, for a 1-byte field, there can be 256 distinct values in a single byte (0 – 255). Practically speaking, a single-byte key has just two key values; for example: M(ale)/F(emale) or Y(es)/N(o). If a file has a million records and just two key values for a specific index, then there will be at least a half million duplicates on one of those values. Even a 5-byte key (such as a zip code, an item code, or a date) often has but a few hundred frequently used values; again, with a million records, several values will have tens of thousands of duplicates.

The standard tool ANALYZE/RMS/FDL can help identify the problem, but it can also be misleading. Its DUPLICATES_PER_SIDR counter is reset for every new bucket, treating a continuation SIDR just like a new SIDR. When ANALYZE reports DUPLICATES_PER_SIDR=500, this is an average that, to the casual observer, suggests a flat distribution. In reality, though, a single chain of 1000 buckets each with 1000 duplicates each for each single value, and 1000 more SIDRs with a single entry, averages out to 500 but is more accurately represented by a duplicate count of 1,000,000.

A better indication within the ANALYZE stats is a large difference between the number of level 1 index records and the number of SIDR buckets. The difference indicates the number of buckets without an index, that is, those in use by duplicate key chains. The following example shows part of the analysis output for a file with a bucket size of 12:

```
ANALYSIS_OF_KEY 1
        :
        DATA_SPACE_OCCUPIED     1968
        DUPLICATES_PER_SIDR     969
        LEVEL1_RECORD_COUNT     9
```

For 1968/12 = 143 SIDR buckets, there are just 9 index pointers. This difference suggests an average of 13 continuation buckets, or an average duplicate key chain of 12000 records. More likely, there was a single duplicate key chain of 100,000 records spanning 100 or more bucket.

RMS Tune Check Tool

A powerful alternative to the standard ANALYZE is the rms_tune_check tool. This tool is available on the recent RMS tools directory with the OpenVMS Freeware. (An older tool called SIDR, which is similar to rms_tune_check, is available on earlier Freeware CD-ROMs.)

The rms_tune_check tool scans specifically for duplicate key chains larger then a specified threshold. The help text for the tool shows an example script that will help analyze all large indexed files.

The following is sample output from a single real file:

```
$1$DGA3014:[xxxxxxxxx.DATA]xxxxxx.IDX;2
- SIDR: Key 2,   203801 Dups in 704 Buckets for value "11"
- SIDR: Key 3,    99252 Dups in 332 Buckets for value "902       "
- SIDR: Key 4,    24648 Dups in 88 Buckets for value "                "
- SIDR: Key 5,   462729 Dups in 1580 Buckets for value "01"
```

The same program can also report a "top ten" list of duplicate values.

That SIDR data, together with a minimal understanding of the application, makes fixing the performance relatively easy. The following two examples demonstrate a problem situation as found in a real file earlier this year. Once you have read the next section I believe the solution for both cases will become obvious. (Hint for later: think null keys and adding segments.)

```
  Duplicate count, Buckets, Key value
-------------------------------------------------------
   1759748         4045     000000000
        46           1      292164044
        27           1      211941745
        25           1      211147595
        22           1      220995050

  Duplicate count, Buckets, Key value
-------------------------------------------------------
    220461         189      CA
    182738         156      NY
    167123         143      TX
    104023          89      FL
     86792          73      PA
     85524          72      MA
```

How to Solve a Duplicate Key Chain Problem

It is not always possible to solve this performance problem entirely in all cases but more often than not we *can* optimize the performance to a large extent. Here are a few techniques to consider.

Drop the Key

The easiest and most effective solution for this duplicate key chain problem is to drop the key altogether. You laugh; but it might just work for you.

- Maybe you have a key on a stray field in a file where some data (perhaps a back reference or additional date stamp) was going to be stored. However, that functionality in your application

was not implemented and the field was left filled with blanks all along. The blank key never seemed to cause problems when the application was tested with a few thousand records, but now that the file has grown over time to contain millions of record, it is slowing the system down.

- How about that key in the Country or State field of an address? Already the set of values to choose from is limited, and maybe not all are used yet. For example, a company in the United States might do business with 30 out of 50 states, but in reality the bulk of the records are from only a handful of states. Perhaps this key is used only by a weekly batch job that reports business across the states or for a particular state. Consider changing that job to read the whole file by primary key and to filter for the selected state. Alternatively, you could have it to pass records to (callable) sort. Consider putting a process in place to convert the file to add a key with the state field just before it is needed, instead of maintaining it for each record inserted. In all likelihood, there is very little business value in an online lookup (such as, "Find the first customer in California.")

Use a NULL KEY value

The null key is a mechanism RMS has always provided specifically to avoid duplicate key chain problems. Although it is restrictive, it is frequently useful.

The null key value is a double-barreled key attribute you can define with FDL (or XABs for the diehard programmers). First specify NULL_KEY yes, then specify a single null key byte. For example, use NULL_VALUE ' ' for a single space. ANAL/RMS/FDL will report this as follows:

```
KEY X
  CHANGES                 yes
  DUPLICATES              yes
  :
  NULL_KEY                yes
  NULL_VALUE              32
  :
  SEG0_LENGTH             LL
  SEG0_POSITION           PP
```

What's the consequence for RMS? *If* a new record is inserted into the file (via the RMS $PUT operation) *and* all bytes of this key's value are identical, *and* this byte value is that of the one defined as the NULL KEY value (=SPACE=32 decimal), then no alternate index entry is made for that record. This solution works immediately for cases with a single-byte key. It also works for the unimplemented field (as in the stray field example), since such fields often contain a string of space (or null) characters. This solution does not work directly for the Status=Done example or for the State=CA scenario, described earlier. For those cases, you need to adapt the application to replace a single, frequently recurring word by a special, reserved series of repeating characters; for example, DDDDDD instead of Done or XX for a state. This works around the single-byte restriction.

For more information, refer to the EDIT/FDL section on null values in the *OpenVMS Record Management Utilities Reference* manual.

Increase the Bucket Size

Strictly speaking, increasing the bucket size is not a good solution. Rather, it is only an effective workaround that hides the underlying problem. However, it might provide enough time for you to implement a real solution. By making the buckets larger, RMS needs far fewer I/Os to find where to insert a new RRV. There are just as many kilobytes of SIDR data to wade through, but it can be done with much less overhead. A small alternate-key bucket size of just 2 blocks (found in some legacy applications) holds fewer than150 RRVs in a SIDR. A typical (and more appropriate) bucket size of 12 blocks holds almost 900 RRVs. Remember that the maximum bucket size is 63 blocks. Each such bucket can hold about 4600 pointers.

Note that this should be only a temporary solution, since the system is still doing excess work.

Deduplicate the Key Values

First, let's remember that a duplicate key value is not a bad thing in itself. Duplicate keys are, in fact, relatively efficient storage. They tend to be shorter than unique keys and, since all key values are identical, they allow for 100% key compression. As long as all duplicate pointers fit in a single bucket, there is no problem using them. Also, many applications can easily tolerate a chain that spans a few buckets. Only when an application frequently adds duplicate values to an already long list that spans dozens or hundreds of buckets will duplicates cost too much to update.

Key values are deduplicated by adding additional, changing, bytes to a key field. Sometimes this is done simply by increasing the key length. For example, suppose a State field is followed by an adjacent zip code field. By adding a 5-character (or 9-character) zip code to the 2-byte state field key, the combined key clearly does not become unique. And our goal is not to make them unique. The millions of duplicates for California will be reduced to a few thousand and will become manageable.

If no useful adjacent field is available, a key segment can be added. (See the description of xab$w_pos in the *OpenVMS Record Management Services Reference Manual*). In the Status example, you might want to add a Done date or an *MMDD* from a date field to the Status key, thereby reducing the number of duplicates from 99% of the file to the number of records processed every day. (This example assumes that records are purged yearly. If not, a year indicator also might be needed.)

Please note that no data is added to the record; the Status field in the application is not extended. Only the definition of the key that used to map directly, and only onto the state field, changes to point to more data. No change in application code is required.

For other applications, you might be able to add a frequently changing single byte from an unrelated binary field. With a perfect distribution, this divides the number of duplicates by a factor of 256. Even with a skewed distribution, you can still expect an improvement of two orders of magnitude. If only ASCII/decimal bytes are added, then each byte will give only a factor of 10, and you will need 3 or 4 bytes to sufficiently reduce duplicates. Again, in the Status example, you can add all or part of an item code (or similar) field as an additional segment.

Adding a segment can be entirely transparent to the application accessing the file by that key as RMS allows for partial or generic key lookup. The specified key length does not have to match the full key size; rather, it can be equal to the original key size. (See the description of rab$b_ksz in the *OpenVMS Record Management Services Reference Manual*.)

Possible snags associated with adding key segments:

- RMS now honors the order of arrival within the new key definition. It takes both the original field as well as the added segments into account, which can result in a new sort order. This may or may not be relevant for the application.

- Some languages verify that the key specification in the program exactly matches the definition in the file itself when opening an existing file. This verification creates no problem in MACRO, C, or BASIC. Programs written in other languages might need to be adjusted and recompiled.
- Some languages do not support segmented keys.

Reminder: The goal is not to make unique keys. To have some duplicates, even hundreds, is fine.

Why Me, Why Now?

Because it's your turn to be a hero!

There are three main reasons for the occurrence of the duplicate key problem: Neglect, Time, and Fear of All Things New.

As time passes, applications scale to unimagined sizes, and they run with millions of records although they were designed and tested only for thousands. Brute-force hardware solutions can ease the pain, but at a price, and eventually hard limits will be reached. Perhaps it will be an IOLOCK8 VMS internal bottleneck after doing too many I/Os per second. Perhaps the duplicate key chain used to fit completely in the disk controller cache and now no longer fits. You can buy still more cache, which would be the "trusted" solution. However, it is far more advisable and rewarding to change the application so that RMS no longer does all those read I/Os.

Many believe that RMS tuning is "black magic," but it is not. In a few days of effort, most of us can pick up the essentials. If you don't make this effort, then tuning is done only once, shortly after implementation. Others believe that all that's required for RMS tuning is a modest automated procedure with ANAL/RMS … EDIT/FDL/NOINTERACTIVE … CONVERT. Such procedures are great, but they are not sufficient over years of change.

You will have to make changes to get changes, there is always risk involved with that. You will also need buy-in from operations, development, and management. But if you find that missing null-key bit, you could save your company millions of dollars. So use the tools, analyze the data, and make the change. What a nice change it will be!

Notes

1. The RMS tools directory with the OpenVMS Freeware contains several tools that may be useful for RMS work, as well as a PowerPoint presentation about RMS tuning and an Excel spreadsheet to review file design.
2. When dropping an unselective key, forcing an application to read all records by primary key can help avoid I/Os. Unless there is an associated primary-key order to the alternate key, you get 1 I/O per record read by an alternate key. Reading a file by primary key, each single I/O will return a bucket full of records, 5 – 50 records depending on the bucket and record sizes. Thus, even if an alternate key selects only10% of all records, it may still be faster to read all records, since more than 10 records fit in a bucket.
3. Duplicate key chains tend *not* to be cached by global buffers. The subtle reason for this is that RMS targets buckets to the local cache, if they are not in the global buffer cache yet, and are requested with write intent. When the duplicate key problem occurs, there is write intent. On one hand, this is unfortunate, since it would provide a seemingly easy workaround. On the other hand, it would only hide a problem that ultimately needs to be fixed. Furthermore, adding thousands of duplicate key buckets to the global buffer cache can easily exceed the capacity of that cache and make matters much worse for other users of that cache (that is, thrashing can occur).

4. Duplicate key chains are not restricted to alternate keys; they can also happen with the primary key. But they don't, because application designers tend to pick unique or near-unique primary keys.

A Customer Case Study of Oracle Rdb Database Consolidation

Authors:
Vivi Maurizio, SIAER, System & Development Technical Director
Turrini Giorgio, SIAER, Software Solution Architect, R&D Senior Consultant
Ghedini Vanna, SIAER, Data Base Administrator
Lotti Ermanno, SIAER, System Manager Supervisor
Guerri Stefano, SIAER, System Manager
Rosani Ornella, SIAER, Networking & Database Manager
Revisited by:
Vischio Giovanni, HP, OpenVMS Ambassador, Pre-Sales Technology & Solution Consultant

Overview

Early last year, SIAER, a long-time Italian OpenVMS customer, began investigating the feasibility of a project to consolidate server, storage, software applications, and Rdb databases along with the renewal of its network IT infrastructure. In order to provide the customer with the proof of concept that the global consolidation was possible with the technology architecture we proposed, HP invited SIAER to the OpenVMS Solution Center in Nashua, NH, to test its applications and database in a new environment, representing a significant subset of the global consolidation.

After the visit, SIAER reported success from its tests and great benefits from Oracle Rdb's Row Cache technology. SIAER provided us with an exhaustive report of the trials and tests performed. This article details the experiences of this customer in making use of an OpenVMS cluster and Rdb's Row Cache feature to achieve great improvements in performance after the consolidation process. This article is another positive result of the HP and SIAER partnership.

Company Overview

S.I.A.E.R — Sistema Informativo Aziende Emilia Romagna (Information System of Emilia Romagna Companies) — is a company founded in 1981 (SIAER scarl, Via Malavolti, 5, 41100 Modena, Italy www.siaer.it). It has 60 employees and approximately 20 independent consultants. It plans and develops software applications for financial and administrative functions of companies (payroll, financial accounting, management control, business relations with local and public government, bank, commercial institution, and others) in an integrated environment of services, which provides a high level of efficiency and achieves a solid and wide database of craftsmanship and small-medium business companies. SIAER works in a B2B environment and currently serves 15 associations (provinces), mostly based in north central Italy; those associations belong to CNA, a primary trade association agency (www.cna.it - www.er.cna.it).

SIAER customers, in turn, provide services to approximately 80,000 firms and employ more than 3,000 operators at local agencies (provinces). Furthermore, SIAER recently implemented SIR, Sportello Istruttore di Rete, which allows users to interact within local and central administrations;

moreover, it put "On-line Services" in place for its customers that allow companies to access web-based applications of the current integrated environment of services. SIAER developed a software application for benchmarking services for Ecipar Emilia Romagna and started the implementation of EKO (Ecipar Knowledge Organization), an ERP system for global management of educational services.

In 2001, SIAER, with the contribution of a major Italian telecommunication carrier, built one of the wider broadband networks in Italy. In January 2003, with the HP and TelecomItalia partnership, SIAER started the implementation of the IT infrastructure renewal project: near its main office in Modena, SIAER built the Data Center where all hardware infrastructure and software applications were brought together through a process of server, storage, database and application consolidation. During 2003, SIAER will provide its customers with desktop management service and will assume the shape of one of the most important ASP (Application Service Provider) in Italy.

SIAER's President is Mr. Giorgio Allari; SIAER CEO is Mr. Lauro Venturi.

Acknowledgements

We would like to thank the following people who spent time and effort to make this document possible:

Aldo Priora, HP Italy: global organization
Vittorio Mezzano, HP Corp.: global organization
Giovanni Vischio, HP Italy: technical and logistic support
Craig Showers & OpenVMS Solutions Center Team in Nashua, HP Corp.: systems configuration and support
Bill Gettys, Oracle Corp.: Rdb support
Carlton Davis, HP Corp.: Rdb support

The Server Consolidation Project: "Golem" — The Middleware to Support Applications

From the functional point of view, all services supplied by SIAER work together; SiDist, a middleware component developed in-house, provides the overall integration. SiDist, based on a semantic model, maps object representation, consistency rules, and logic of the distributed model on a wide number of approximately 200 Rdb/Oracle databases: SiDist manages the distributed model based on a partial data replication. The project to consolidate hardware infrastructure and software applications required changing the SiDist distributed model to a centralized model: the Golem.

Introduction

To evaluate the feasibility of the server consolidation project, we conducted performance and workload benchmark tests on an OpenVMS AlphaServer cluster (2 x ES45) at the OpenVMS Solution Center in Nashua, NH (USA).

> This testing was part of a wider collaboration with OpenVMS Engineering started in March 2002 when the SIAER CEO visited Nashua. Italian sales and technical account managers later provided a preliminary global outsourcing proposal in response to SIAER's request. This document and the related results of the benchmark testing were prerequisites for a successful server consolidation.

The scope of the benchmark testing was to:

- Validate configuration and structural changes to the main application program (SiDist) and to the Rdb database
- Check and validate the new environment (Golem) with an up-to-date hardware and software system architecture
- Verify concurrent access to the unique Rdb consolidated database by a large number of users.

Systems and storage were available between July 8th and August 16th. Prior to July 28 we used systems for experiments, unofficial tests, and preparing scripts and test data. We performed official tests between July 29th and August 16th when SIAER personnel were available.

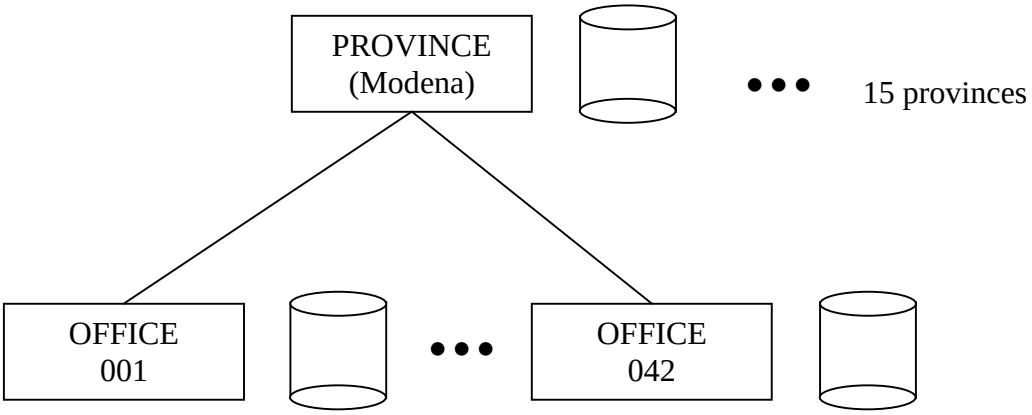
This document contains two sections and an Appendix:

- The first section describes the benchmark setup, preparation, and results
- The second section describes the tests performed, system tools, system architecture, data collections, and results. The contents of this section are fundamental to planning the future Golem implementation.
- The Appendix gives details of each test.

Benchmark Setup and Preparation

The Environment

The driving force behind this test was the server consolidation project that SIAER planned for 2003. The current configuration is as follows:



Currently, each location has a system and a database running locally; the software application, SiDist, manages data replication and coherence. The future configuration will be located in the "Datacenter," where a single system will handle the workload of all the offices and the main province office; the consolidation project foresees a single database instance called "Golem".

In the current configuration, we have multiple databases with three kinds of data:

- GLOBAL: shared between province and office and between offices (the percentage of sharing between offices is around 10% — 336 tables)
- LOCAL: each record in these tables is exclusively owned by one office; local data exists also in the province office (228 tables)
- BROADCAST: the same data is everywhere (147 tables)

Golem will consolidate all the data on a single database. This change leads to the "Visibility problem"! Golem solves the problem by introducing a table, called the "Visibility table", which correlates the user with the data that can be viewed; this table allows more than one user to view the same data. In the actual design, a single visibility table is used to manage the visibility for a set of related tables; currently 17 tables are used to manage visibility for 336 global tables.

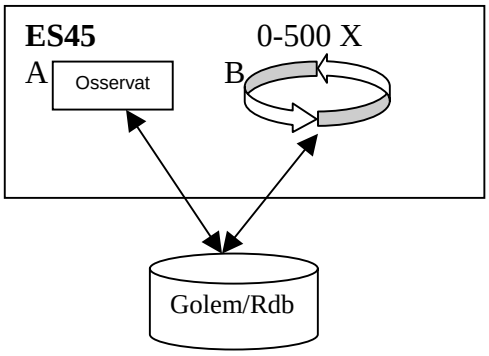
Test Structure

A general overview of the benchmark shows two kind of process to be run:

- Main interactive processes (one or two processes called A or Osservato) monitored by SIAER personnel using general system and Rdb-specific tools.
- Many (0 to 500) workload processes (called B or Disturb) in order to simulate an effective workload against the systems.

We ran both A Osservato processes and B Disturb processes and collected data about system parameters and elapsed time; that data has been compared to results from a test system (a DS20E) located in Italy at SIAER.
The target of the test plan is to analyze the A Osservato process when we increase the number of B Disturb processes.

A: Osservato process: an interactive process simulating user activities at the office. B: Disturb processes: workload processes simulating overall activities in a typical office (usually batch processes about report and printing).
We collected system data and analyzed log files for each A Osservato process running different workload conditions.



B Disturb process, coded with specific SIAER language (SIC), the implementation language of the SiDist main SAIER application, simulates back-office activities typical at either local or main offices: read, modify, add, and delete of records in the Golem database. To accurately simulate back-office tasks, we also introduced "idle time" that emulates human activities (average idle time was calculated from DTM (DEC Test Manager) sessions recorded at typical local and main offices in Italy).

Back-office tasks run against the Golem database to a limited extent (local office data), but batch tasks (report, printing, and so forth) usually run against the whole Golem database.

The A Osservato process runs at the same time with different workloads (0 to 500 Disturb processes): we collected elapsed time from all sessions.

From the system log, we estimated transactions of the systems, and from Rdb monitor and log files we estimated transactions of the database. In both cases we determined an average number of transactions per second calculated on a timeframe of 90 seconds while A Osservato process and B Disturb processes were running.

History Reports

July 15th to 26th

During these two weeks we gained access to systems in Nashua and refined systems and executables as follows:

- Built up and adjusted batch procedures and control statements.
- Made changes to the main program (SiDist) to remove a record update used specifically by the distributed model of SiDist.
- Improved the algorithm for report generation.
- Developed a detailed plan and schedule for the next two weeks, officially allocated for test.

July 29th to August 2nd

Monday, Tuesday, Wednesday:
A first step was performed with strange results: we experienced several problems that badly altered results and the assessment between simulated and real environment: priority, idle time, high collision rate of database access. We made adjustments and procedure changes and planned a new series of tests.
Thursday:
Due to network problems, the system was inaccessible from Italy. Support teams in Nashua and HP and SIAER personnel worked together on the problems. The problems were fixed Thursday evening, and we planned new tests and batch sessions for that night.
Friday:
The DTM recorded session did not work properly. A new session was recorded and we ran tests TP1, TP2, TP3, TP4, TP5 Rdb: 80000 global buffers, no global buffers in VLM. (See the appendix for all references to tests named TPx, TZx, T…)

Summary after the first week:

Many problems have been discovered: DTM recorded sessions are not reliable due to many failures during execution.

A new strategy has been implemented.

We created two new processes, OSSERVATO and OSSERVATO_L, which simulate human behavior; in detail:

- OSSERVATO: user session on global data (company and related database structures)
- OSSERVATO_L: user session on local data (local financial accounting)

Both processes execute standard operations on specific database instances (updates, insert, delete) with idle time between operations in order to simulate human activities.

Those changes do not alter the tests: we still have A Osservato processes and B Disturb processes, and we still consider it to be a new session when we increase B Disturb processes. The main change is the Osservato processes: they are now executable and not recorded sessions.

August 5th to 9th

Monday:

Hardware and software were both reconfigured for further test sessions (database and application server, cluster configuration, and so forth); changes were made to batch procedures to have a heavier workload (from 230 to 500 users — disturb processes).

Tuesday:

Ran tests TZ2, TZ3, TZ4, TZ5. Rdb: global buffers in VLM 524000, 700 per user

Wednesday:

Ran tests TZ1 and TZ6.

Ran tests TDB1, TDB5, TDB7 with database machine + global buffers in VLM 524000, 700 per user

Thursday:

Used a new software configuration with row cache; made changes to restore database procedures.

Ran tests TRC5, TRC7 database machine + row cache + global buffers in VLM 524000, 700 per user

Friday:

Ran a new software configuration with row cache.

Ran test TRC7N: database machine + row cache + global buffers in VLM 524000, 700 per user

Summary after second week:

The changes made the system more robust, reliable, and stable; performance has been increased. Tests ran as planned without problems in all configurations.

We achieved our planned goals. Due to high loads of work for the systems, and for SIAER and HP people, we closed all sessions and the week after we conducted manual tests and reorganized the data and database, collected log files and results, etc.

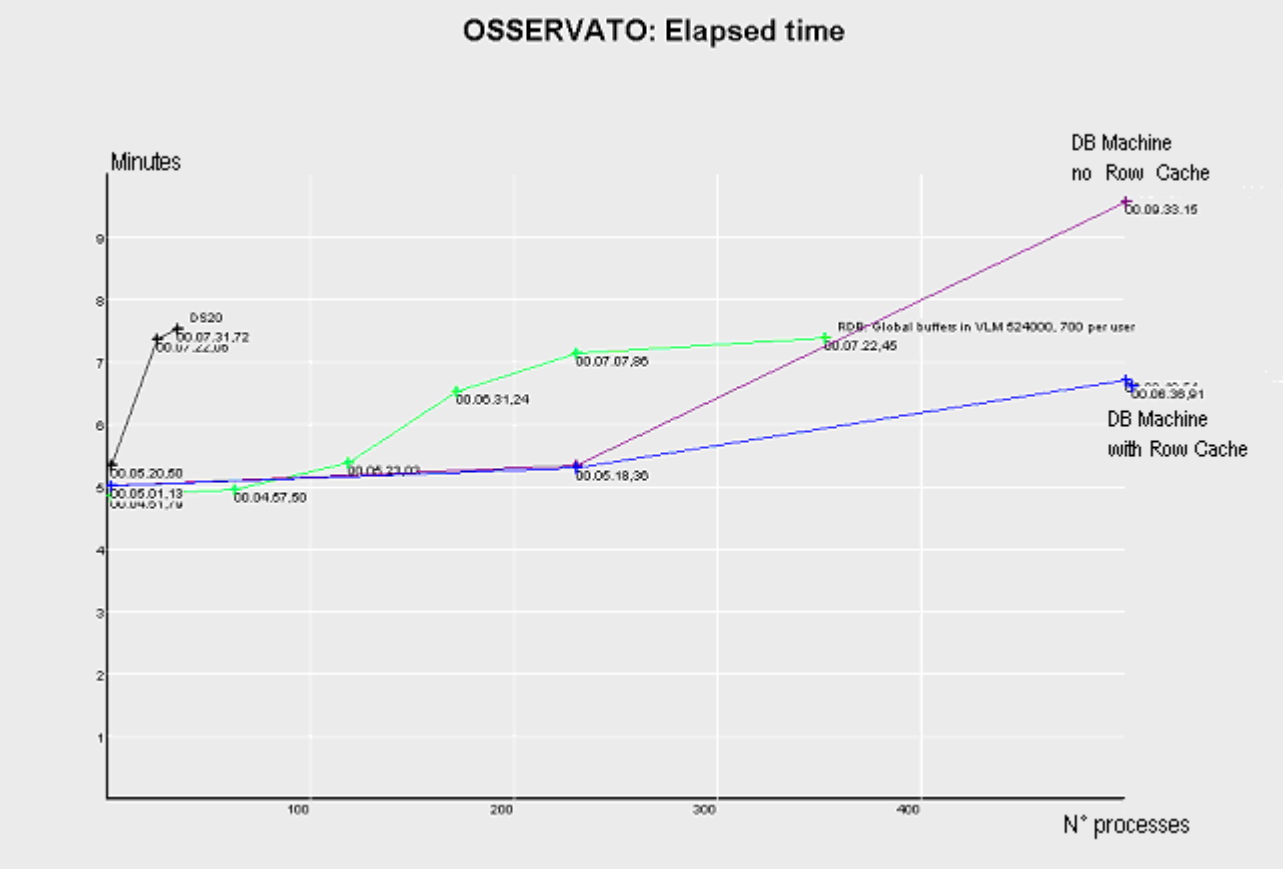
As agreed early in July, systems and storage were released to HP on August 18th.

August 19th to 27th

To compare test results with the current production environment, we organized a test session with an internal system (code name: Foa001). We used the same database and scripts with minor changes due to different hardware (DS20). On August 27th, we ran Osservato processes with a workload of disturb processes that simulated 25 and 35 users.

Comments About Tests

The Technical Analysis of Benchmark (the second section of this document) details processes and results. A more in-depth analysis needs more time and work, but we have comments and information that could be made public, as follows:

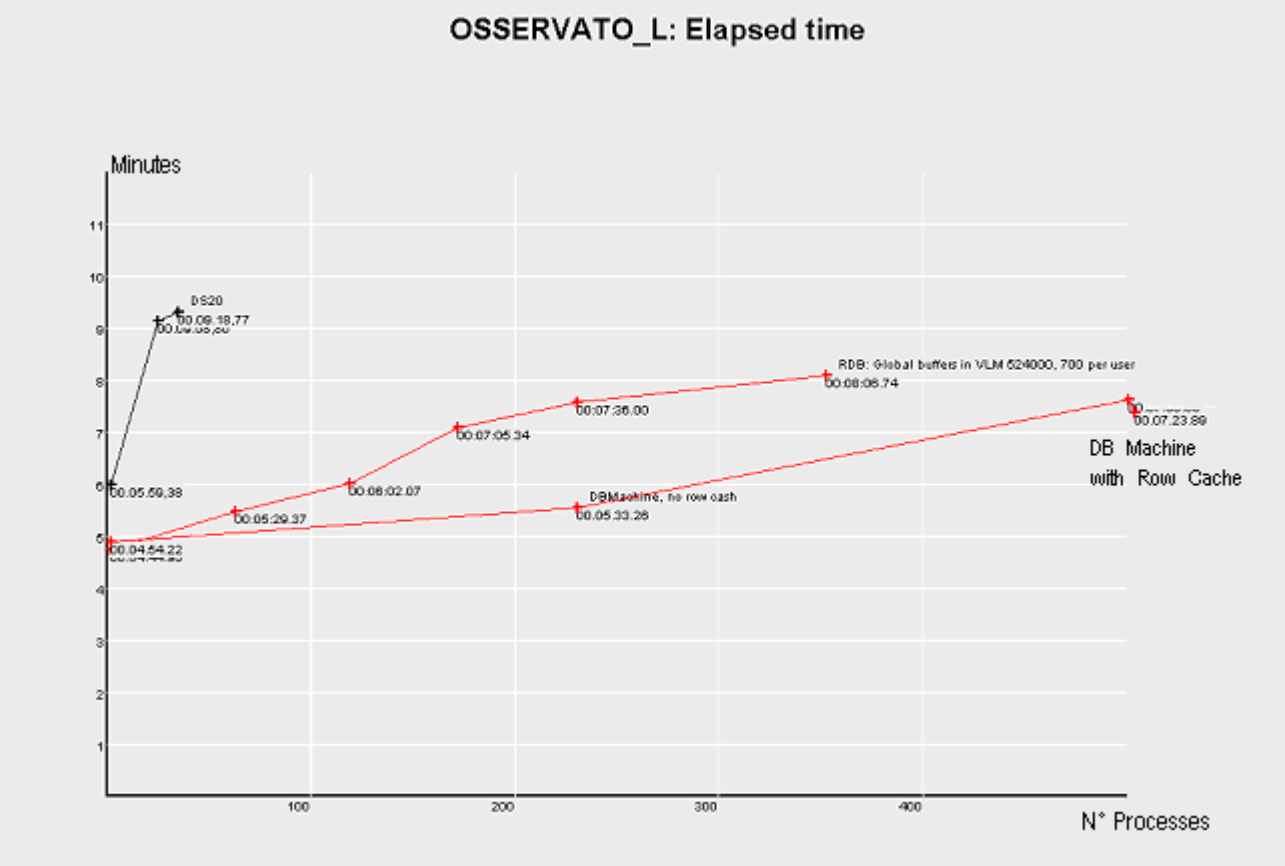


The previous figure shows Osservato process elapsed time (this process run against global data) in different configurations. It may be interesting to look at the left side of the picture where the results of the DS20 used locally in SIAER can be seen (it simulates offices called Foa001).

The less favorable configuration is database machine; there is no row cache and the result shows a breakpoint when we have 350 users (or disturb processes).

The other two configurations are better.

Note that the Osservato process is a heavy workload for the system because it runs on global data.



The previous figure shows Osservato_l process elapsed time (this process run against local data) in different configurations. Look at left side of the picture where the results of the DS20 used locally in SIAER can be seen (it simulate offices called Foa001).

All configurations are better than today.

Final Results

Let's review the target and results of our benchmark:

Validate configuration and structural changes to the main application program (SiDist) and to the Rdb database: this should enable us to implement a unique Rdb database for each single area (province in Italy).

> Functional tests were run before this benchmark, but in our recent tests with new AlphaServer ES45 systems and MA8000 FC storage, the main program (SiDist) was stressed as never before: a heavy workload of more than 500 users was simulated; note that 500 users are 70% of the theoretic overall users of the Modena area. We did not find a single fault or problem in either system architecture or database.

Check and validate the new environment (Golem) with an up-to-date hardware and software system architecture as included in the HP proposal, with the goal of offering performance at least as good as users have today in the distributed environment.

> The Technical Analysis of Benchmark (the second section of this document) fully details this target. Looking ahead, we found that in all configurations we tested, Osservato processes ran faster than on DS20's. We cannot say at this point which is the best system architecture and configuration; it is possible that changes to the database and SiDist code caused the performance improvements.

Verify concurrent access to the unique Rdb consolidated database by a large number of users.

> This was the main concern we had before the benchmark. The tests show that Rdb manages concurrent access in an excellent way, and database-served queues contained few entries (20-30) at the worst workload (>500 users simulated). The database table containing local financial accounting was critical: tests on Osservato_l process do not show any problems on concurrent access.

Summary

Here are some final comments:

- The code and performance of the SiDist main application can be improved.
- The performance of the Rdb database can be improved through reorganization. Mr. Bill Gettys from Oracle Corp. in Nashua, after a quick review of our architecture, said the Golem database could be considered a "very large database"; for those databases, a correct architecture and organization is absolutely necessary in order to gain better performance.
- The Golem database has been robust and reliable in all workloads and conditions. Golem crashed rarely, and the causes were immediately identified.

- The benchmark shows that the application SiDist can be implemented in a consolidated environment and that the HP proposal well matches application requirements. Now that we have proven that the server and Rdb consolidation can be implemented on HP architecture, we have to refine and complete all applications.

Technical Analysis of Benchmark

Methodology

This section describes how we implemented the test strategy. We developed a set of scripts using SiDist (file extension is .SIC) language; those scripts emulate back-office activities inside local and main area CNA facilities based on a set of data significant enough for testing. All test procedures and reports run in batch mode on a specific queue with base priority 4 (the base priority OpenVMS reserves for interactive processes).
Scripts execute typical activities like record modify, add, delete on local data concerning the local office itself. On the other side, we also have scripts and reports running on global data and emulating the main area (province) offices. That is, the first set of scripts run local data against the local database; the second set of scripts and reports run global data against the "whole database"; they run in different configurations in order to generate a workload of from 1 to 500 users.

At the same time, two specific procedures, both named 'osservati', ran on the system and we monitored them with different workloads (users) and collected the corresponding elapsed times. From log files we extracted interesting details concerning overall activities that the system/cluster can perform; also, using Rdb monitor tools, we collected from the whole database an average transactions per second measured on a timeframe of 90 minutes during the execution of the 'osservati' processes.

As a first step, we executed two DTM interactive procedures to test the system and collect details on transactions; unfortunately, the DTM recorded interactive sessions were not reliable due to problems in events synchronization. Therefore, we decided to change our strategy as previously described.

Hardware Configuration

The cluster configuration contained the following hardware:

1. ES45 (sia047), 16 GB memory, 4 CPU 1001MHz
2. ES45 (sia048), 12 GB memory, 4 CPU 1001MHz
3. MA8000 storage array with 5 volumes:
   - 1DGA200: 36 GB (2 x 18 GB disks stripe set)
   - 1DGA300: 36 GB (2 x 18 GB disks stripe set)
   - 1DGA400: 144 GB (8 x 18 GB disks stripe set)
   - 1DGA500: 72 GB (4 x 18 GB disks stripe set)
   - 1DGA800: 36 GB (2 x 18 GB disks stripe set)

We also used the following equipment for communication and services purposes:

1. DS20 (sia049); it operated as a DTM server and to launch interactive sessions
2. XP1000 (isvlab); it ran as a gateway between lab and external internet access
3. CISCO Firewall

From SIAER facilities in Modena, Italy and Nashua Labs we had access via telnet with triple authentication: Cisco, isvlab AlphaServer and ES45's cluster.

Software Configuration

The software configuration was as follows:

1. OpenVMS 7.3 with XFC cache enable
2. Oracle Rdb v 7.1-02
3. Data monitor and collector PSDC
4. SiDist application with Golem support (global database):
   - Local and global data access
   - Query optimization (for report generation only)
5. DCL procedures to start test execution
6. SiDist procedures, as described in Table 1.

– *Table 1: Procedure Descriptions*

Name	Description
Gol_ente_rag.sic	Add and update of ENTE (company) record and related structures
Gol_ente_rag_nonew.sic	Update of ENTE (company) record and related structures
Gol_pers_cognomi.sic	Add and update of PERSONA (person) record and related structures
Gol_pers_nonew.sic	update of PERSONA (person) record and related structures
Gol_genmov1.sic, gol_genmov2.sic, gol_genmova.sic, gol_genmovb.sic, gol_genmovc.sic	Different procedures to read and manage financial accounting record and data structure
Gol_crediti.sic	Read CLIENTE_CNA record and add / update credits
Gol_f24.sic	Mining of 'pkey' details for proxy payment
Gol_dett.sic	Add and create invoices
Gol_righe.sic	Reading and updating all financial accounting record for selected ENTE
Report gol_tess.sic	Complex report (iscritto, casind, impresa, albo_costruttore, auto_trasportatore, operatore_estero, esercente_commercio, tessera, artigiana, separata_sezione, commerciale, piccola_impresa)
Report gol_repcon.sic	Simple report
Report gol_reppag.sic	Simple report
Report gol_prato.sic	Complex report (iscritto, cliente_cna, impresa, sede, comuni, casind, contabilità, mia, consulenza, cose, coge, coge_attivita, pa_mensile, paghe, istanza, artigiana, separata_sezione, piccola_impresa, commerciale)

We used the set of scripts, procedures, and records as shown in Table 2.

– *Table 2: Sets of Procedures*

Workload	Set of Scripts
Set ONE (< 350 users, test 1,2,3,4 e 5)	• Ufficio (7 procedures) o 1 x `Gol_ente_rag.sic` o 2 x `Gol_ente_rag_nonew.sic` o 1 x `Gol_pers_cognomi.sic` o 1 x `Gol_pers_nonew.sic` o 2 x `Gol_genmov[1|2].sic` • Provincia (7 procedures) o 1 x `Gol_ente_rag_nonew.sic` o 2 x `Gol_repcon.sic` o 2 x `Gol_reppag.sic` o 2 x `Gol_crediti.sic` • Unici (5 reports and procedures) o 1 x `Gol_f24.sic` o 1 x `Gol_righe.sic` o 1 x `Gol_dett.sic` o 1 x `Gol_prato.sic` o 1 x `Gol_tess.sic` • Osservati (2 procedures) o 1 x `Gol_osservato.sic` o 1 x `Gol_osservato_l.sic`

Set TWO (~ 350 users, test 6)	• Ufficio (9 procedures)
	  o 1 x `Gol_ente_rag.sic`
	  o 3 x `Gol_ente_rag_nonew.sic`
	  o 1 x `Gol_pers_cognomi.sic`
	  o 2 x `Gol_pers_nonew.sic`
	  o 1 x `Gol_genmov[1|2].sic`
	• Provincia (7 procedures)
	  o 1 x `Gol_ente_rag_nonew.sic`
	  o 2 x `Gol_repcon.sic`
	  o 2 x `Gol_reppag.sic`
	  o 2 x `Gol_crediti.sic`
	• Unici (5 reports and procedures)
	  o 1 x `Gol_f24.sic`
	  o 1 x `Gol_righe.sic`
	  o 1 x `Gol_dett.sic`
	  o 1 x `Gol_prato.sic`
	  o 1 x `Gol_tess.sic`
	• Osservati (2 procedures)
	  o 1 x `Gol_osservato.sic`
	  o 1 x `Gol_osservato_l.sic`

Set THREE (~ 500 users, test 7)	• Ufficio (13 procedures)
	  o 1 x Gol_ente_rag.sic
	  o 4 x Gol_ente_rag_nonew.sic
	  o 1 x Gol_pers_cognomi.sic
	  o 3 x Gol_pers_nonew.sic
	  o 3 x Gol_genmov[a|b|c].sic
	  o 1 x Gol_crediti.sic
	• Provincia (7 procedures)
	  o 1 x Gol_ente_rag_nonew.sic
	  o 2 x Gol_repcon.sic
	  o 2 x Gol_reppag.sic
	  o 2 x Gol_crediti.sic
	• Unici (5 reports and procedures)
	  o 1 x Gol_f24.sic
	  o 1 x Gol_righe.sic
	  o 1 x Gol_dett.sic
	  o 1 x Gol_prato.sic
	  o 1 x Gol_tess.sic
	• Osservati (2 procedures)
	  o 1 x Gol_osservato.sic
	  o 1 x Gol_osservato_l.sic

To generate a correct workload, the amount of "set of scripts" was increased. Table 3 shows the different workload types we used to test the whole architecture.

– *Table 3: Workload Types*

Workload Type	Procedures	Processes
1	No workload 2 osservati	2 osservati Total 2
2	7 local offices x 7 procedures Ufficio (49) 1 main offices x 7 procedures Provincia (7) 5 Unici (5) 2 osservati	61 disturb processes 2 osservati Total 63
3	14 local offices x 7 procedures Ufficio (98) 2 main offices x 7 procedures Provincia (14) 5 unici (5) 2 osservati	117 disturb processes 2 osservati Total 119
4	21 local offices x 7 procedures Ufficio (147) 3 main offices x 7 procedures Provincia (21) 5 Unici (5) 2 osservati	173 disturb processes 2 osservati Total 175
5	28 local offices x 7 procedures Ufficio (196) 4 main offices x 7 procedures Provincia (28) 5 Unici (5) 2 osservati	229 disturb processes 2 osservati Total 231
6	38 local offices x 9 procedures Ufficio (342) 1 main offices x 7 procedures Provincia (7) 5 Unici (5) 2 osservati	354 disturb processes 2 osservati Total 356
7	38 local offices x 13 procedures Ufficio (494) 1 main offices x 7 procedures Provincia (7) 5 Unici (5) 2 osservati	506 disturb processes 2 osservati Total 508

Table 4 shows the architecture and system configuration models we used to run workload procedures. The last digit in Test Name indicates the corresponding workload as shown in Table 3; for example, TZ4 means we used workload number 4 from Table 3, which is 175 processes.

– *Table 4: Configuration Models*

Configurations	Description	Test Name
P	Single ES45 and Global Buffer in 32 bit memory (max. 81550 buffers)	Test TP1, TP2, TP3, TP4 e TP5
Z	Single ES45 and Global Buffers in VLM (max. 524000 buffers)	Test TZ2, TZ3, TZ4, TZ5 e TZ6
DB	Application server ES45 and database server ES45 with Global Buffers in VLM (max. 524000 buffers)	Test TDB5 e TDB7
RC	Application server ES45 and database server ES45 with Global Buffers in VLM (max. 524000 buffers) and row cache configuration type 1	Test TRC5 e TRC7
RCN	Application server ES45 and database server ES45 with Global Buffers in VLM (max. 524000 buffers) and row cache configuration type 2	Test TRC7N

The row cache configurations are shown in Table 5.

– *Table 5: Row Cache Configurations*

Configuration type 1: (326,707,172 byte RAM used)

Users = 700	SLOTS	LENGTH	WSSIZE	Tot.Mem	Phy.Mem
SIB003EE70001D04AA_VIS_IDX	13,000	430	10	6,417,828	6,416,416
SIB003EE7001E904AA_VIS_IDX	35,000	430	10	17,173,924	17,172,512
SIB003EE70001D04AA_PK_ENTITA	33,000	120	10	5,918,116	5,916,704
SIB003EE7001E904AA_PK_ENTITA	88,000	120	10	15,420,836	15,419,424
RDB$SYSTEM_AREA_CACHE	200,000	512	10	113,175,972	113,174,560
SIB003459000B404AC_IDX	19,000	450	10	9,743,780	9,742,368
SIB003459000BC04AC_IDX	13,000	450	10	6,679,972	6,678,560
SIB003EE70001D04AA	58,000	600	10	37,957,028	37,955,616
SIB003EE7000DC04AA	30,000	260	10	9,489,828	9,488,416
SIB003EE7001E904AA	130,000	360	10	53,685,668	53,684,256
SIB003EE7002E204AA	57,000	134	10	10,866,084	10,864,672
SIB003EE7002F404AA	50,000	232	10	14,372,260	14,370,848
SIB013A1E0B8E30A4A	14,000	200	10	3,640,740	3,639,328
SIB003EE7000DC04AA_PK_ENTITA	22,000	120	10	3,935,652	3,934,240
SIB003EE7002E204AA_PK_ENTITA	48,000	120	10	8,441,252	8,439,840
SIB003EE7002F404AA_PK_ENTITA	43,000	120	10	7,597,476	7,596,064
SIB013A1E0B8E30A4A_PK_ENTITA	12,000	120	10	2,190,756	2,189,344

Configuration type 2: (396,374,224 byte RAM used)

Users = 700	SLOTS	LENGTH	WSSIZE	Tot.Mem	Phy.Mem
SIB003EE70001D04AA_VIS_IDX	13,000	430	10	6,417,828	6,416,416
SIB003EE7001E904AA_VIS_IDX	35,000	430	10	17,173,924	17,172,512
SIB003EE70001D04AA_PK_ENTITA	33,000	120	10	5,918,116	5,916,704
SIB003EE7001E904AA_PK_ENTITA	88,000	120	10	15,420,836	15,419,424
RDB$SYSTEM_AREA_CACHE	200,000	512	10	113,175,972	113,174,560
SIB003459000B404AC_IDX	19,000	450	10	9,743,780	9,742,368
SIB003459000BC04AC_IDX	13,000	450	10	6,679,972	6,678,560
SIB003EE70001D04AA	58,000	600	10	37,957,028	37,955,616
SIB003EE7000DC04AA	30,000	260	10	9,489,828	9,488,416
SIB003EE7001E904AA	130,000	360	10	53,685,668	53,684,256
SIB003EE7002E204AA	57,000	134	10	10,866,084	10,864,672
SIB003EE7002F404AA	50,000	232	10	14,372,260	14,370,848
SIB013A1E0B8E30A4A	14,000	200	10	3,640,740	3,639,328
SIB003EE7000DC04AA_PK_ENTITA	22,000	120	10	3,935,652	3,934,240
SIB003EE7002E204AA_PK_ENTITA	48,000	120	10	8,441,252	8,439,840
SIB003EE7002F404AA_PK_ENTITA	43,000	120	10	7,597,476	7,596,064
SIB013A1E0B8E30A4A_PK_ENTITA	12,000	120	10	2,190,756	2,189,344

SIDIST_HASH_BIS_CACHE	100,000	120	10	17,444,260	17,442,848
SIDIST_HASH_CACHE	200,000	120	10	34,778,532	34,777,120
SIDIST_HASH_LOCALI_CACHE	100,000	120	10	17,444,260	17,442,848

The following list provides the cache names and related descriptions:

- Physical caches in Table 6 (they include data corresponding to all record types in the database)

– *Table 6: Physical Caches*

RDB$SYSTEM_AREA_CACHE	Physical Rdb system area
SIDIST_HASH_BIS_CACHE	Physical area with HASH indexes
SIDIST_HASH_CACHE	Physical area with HASH indexes
SIDIST_HASH_LOCALI_CACHE	Physical area with HASH indexes

- Logical caches in Table 7 (they include data corresponding to specific record type in the database)

– *Table 7: Table and Index Caches*

SIB003EE70001D04AA	Table ENTE
SIB003459000B404AC_IDX	Index on RAGIONE_SOCIALE of ENTE
SIB003EE70001D04AA_PK_ENTITA	Index PK_ENTITA for ENTE
SIB003EE70001D04AA_VIS_IDX	Visibility index of hierarchy ENTE
SIB003EE7000DC04AA	Table CLIENTE_CNA
SIB003EE7000DC04AA_PK_ENTITA	Index PK_ENTITA of CLIENTE_CNA
SIB003EE7001E904AA	Table PERSONA
SIB003459000BC04AC_IDX	Index on COGNOME of PERSONA
SIB003EE7001E904AA_PK_ENTITA	Index PK_ENTITA for PERSONA
SIB003EE7001E904AA_VIS_IDX	Visibility index of hierarchy PERSONA
SIB003EE7002E204AA	Table CLIENTE_P
SIB003EE7002E204AA_PK_ENTITA	Index PK_ENTITA of CLIENTE_P
SIB003EE7002F404AA	Table DR_PERSONA
SIB003EE7002F404AA_PK_ENTITA	Index PK_ENTITA of DR_PERSONA
SIB013A1E0B8E30A4A	Table CONTABILITA
SIB013A1E0B8E30A4A_PK_ENTITA	Index PK_ENTITA of CONTABILITA

Analysis of Results

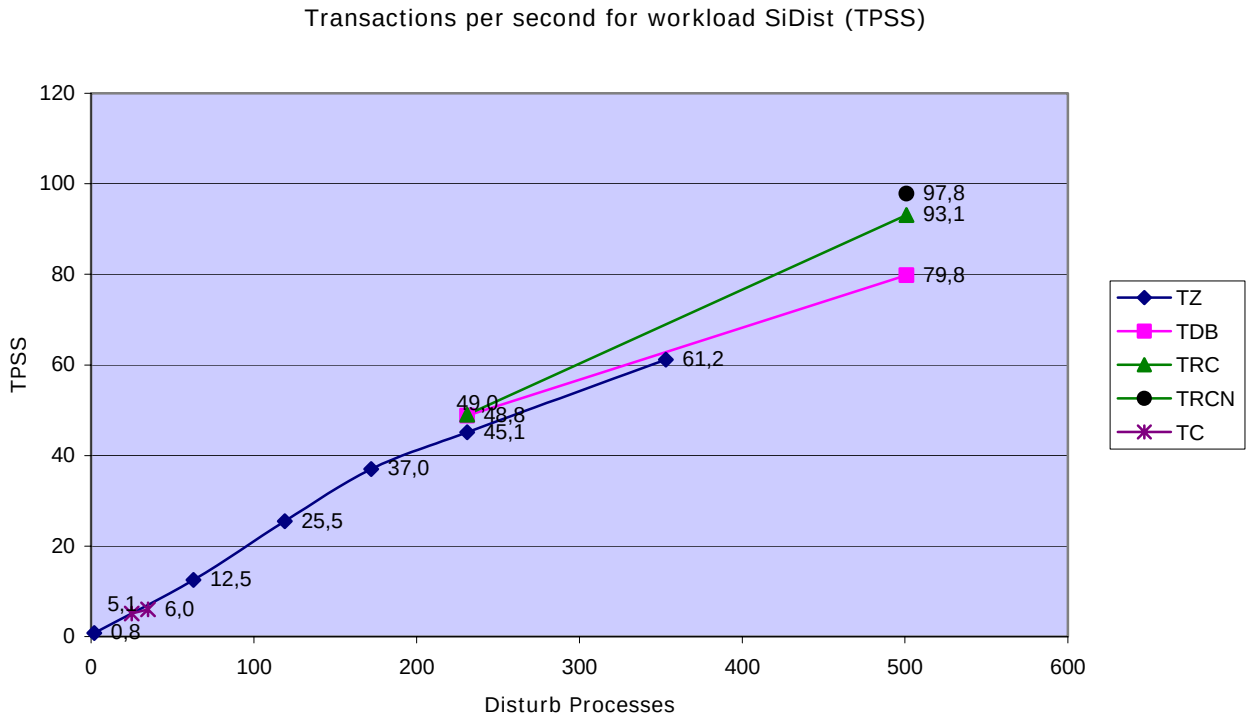
The following figures and tables show all the results we gathered from tests. For detailed information on the tests, see the Appendix.

Figures 1, 2, 3, and 4 show the details collected from the tests.

Figure 1 shows transaction per second, as Oracle Rdb monitor tool reports, for different workloads. Configurations are shown in colors, see legend aside. We reported also the comparison system we used locally: (TC: comparison test).

Note that only the configuration with a global buffer in VLM @ 524000 and single ES45 (TZ) runs tests 2, 3, 4, and 6 (corresponding to 63, 119, 175 e 356 disturb processes); configurations with database server, with or without row cache (TDB e TRC), run tests 5 and 7 (231 and 508 disturb processes). Test 5 (231 disturb processes) has been used against configurations TZ, TDB e TRC. The configurations with database server, with row cache type 2, run test 7 only (508 processes). We'll use TPSS name in order to show TPS (Transactions per second) for workload on SiDist main application.

– *Figure 1: Transaction per Second at a Different Workload (Disturb Processes)*



In Figure 1, note that TPSS has a linear increase on TZ configuration up to test 4 (178 processes); then, it shows a nonlinear curve, meaning that this configuration cannot support a heavier workload. If we highlight the configuration with row cache, it shows a linear growth, like a straight line; if we take only the line portions of tests 1, 2, 3, and 4 for TZ, test 5 for TRCm, and test 7 for TRCN, we will obtain the results in Figure 2.

Figure 2: Trend of Better Results at a Different Workload (Disturb Processes)

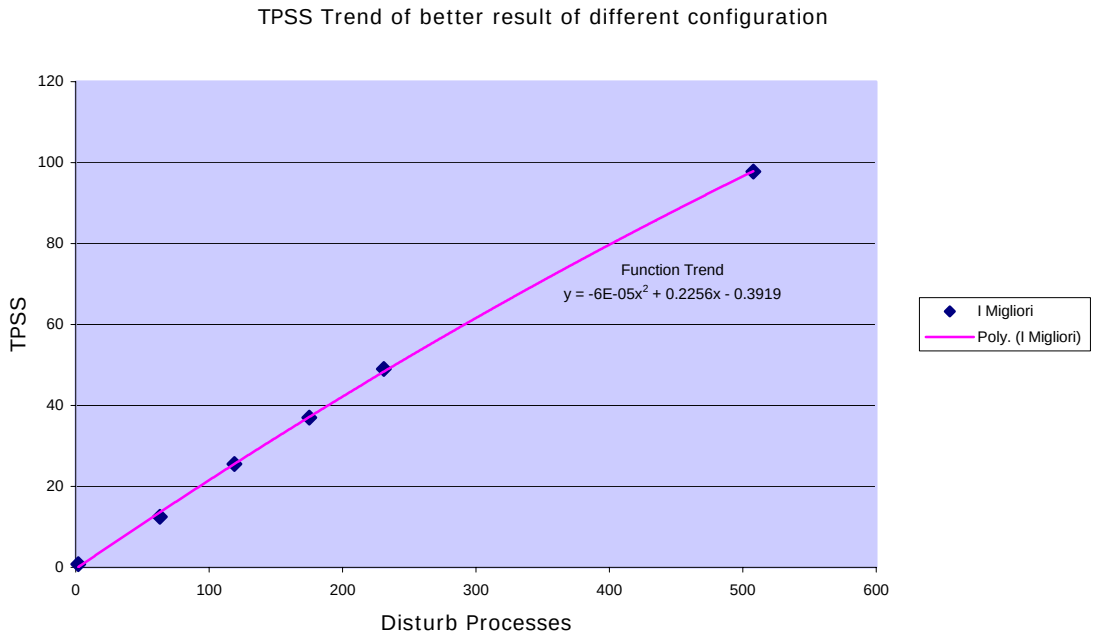


Figure 2 shows a linear curve of TPSS; the configuration with database machine and row cache is the best configuration because it keeps a linear growth at different workload, higher included (508 disturb processes). Also note in this figure that Migliori in the key means "Top of Series."

Figure 3 shows the amount of TPSS per single process (average). We have a very low decline of performances.

Figure 3: TPSS per Process at a Different Workload

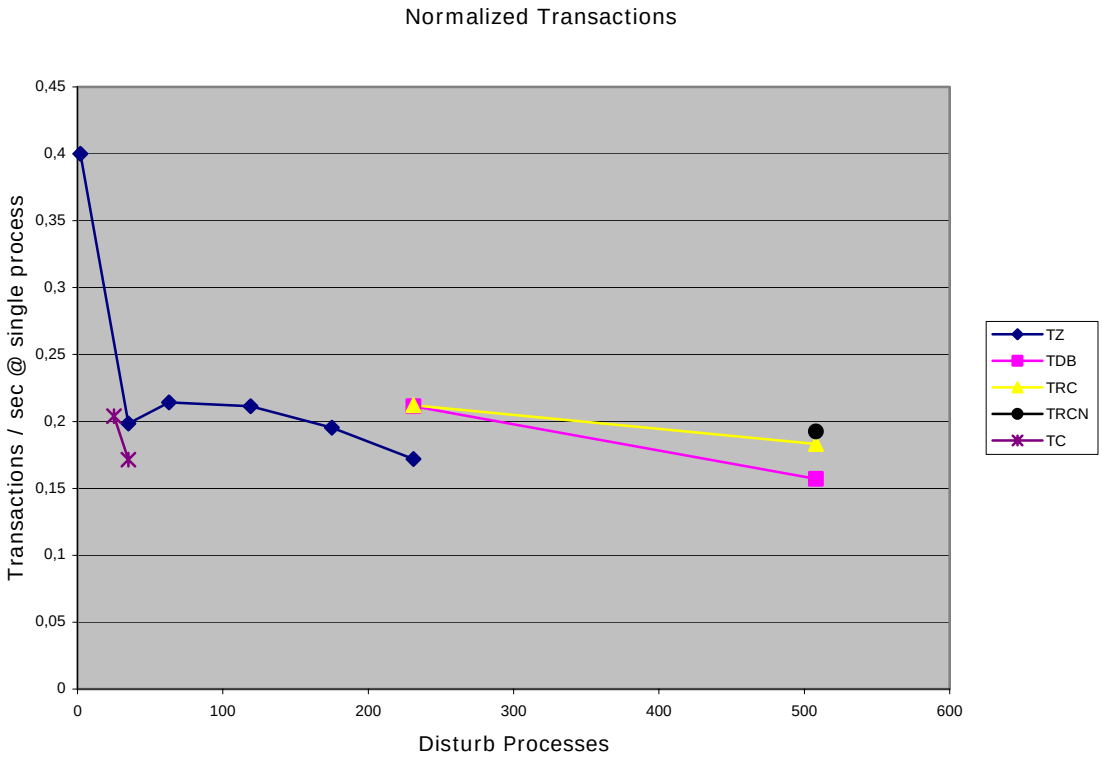


Figure 4: Elapsed time of OSSERVATO_L Process at a Different Workload

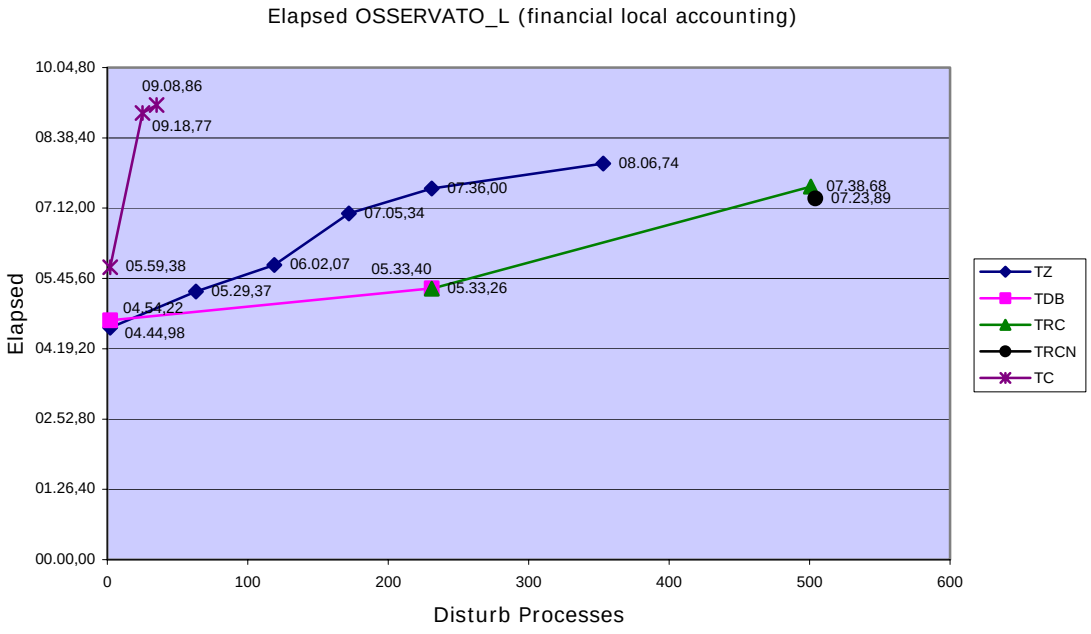
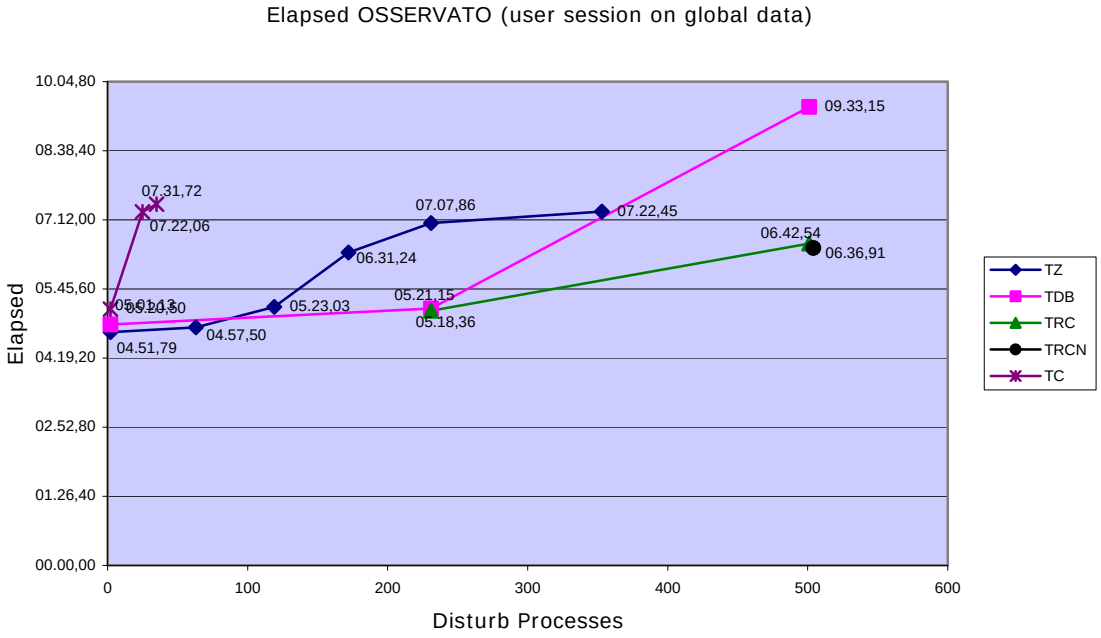


Figure 4 and Figure 5 show the elapsed time for "OSSERVATI" processes on different workloads. Note the elapsed time of "OSSERVATI" processes on the configuration named RCN (the database machine with row cache type 2) and 508 disturb processes. It is clearly less than the elapsed time on the comparison test system (DS20E) with fewer disturb processes (only 25).

OSSERVATO_L: 07:23:89 versus 09:18:77
OSSERVATO: 06:36:91 versus 07:22:06

We may confirm the configuration used for benchmark is better than one we used on our offices; it reacts fine even with higher workload on global data.

Figure 5: Elapsed time of the OSSERVATO Process on a Different Workload



We also collected a number of transactions on different workloads. The data collected were: ENTE creation, update of ENTE and related data structure, PERSONE creation and update, and ACTIVITIES creation.

We normalized transaction data to a one-hour period in order to compare data collected from the benchmark system and comparison test system.

Tests loading and running really penalize the normalization of test results. In fact, tests with a large number of disturb processes have been activated in more than one shot because the server, which handles the generation of disturb processes, did not allow more than 200 disturb processes to start at a time. Therefore, tests with 231 and 356 processes were loaded in two steps and tests with 508 processes were loaded in three steps. In Table 8, start time is related to start time of the last step. We also had difficulties in killing all disturb processes not stopping by themselves as the procedures should do; this may generate some very limited uncertainty in collected results.

Table 8: Transaction Data: Elapsed Time on Different Tests

					ENTE		PERSONE		ACTIVITIES
Test	Processes	Start	End	Elapsed	Creation	Update	Creation	Update	Creation
TP2	61	08:09:43	09:15:43	01:06:00	29	827	26	1319	5704
TP3	117	09:26:18	10:02:31	00:36:13	44	984	34	1454	5874
TP4	173	10:50:20	11:30:38	00:40:18	18	980	32	1602	6150
TP5	229	11:32:10	12:30:00	00:57:50	0	1907	62	3327	13901
TZ2	63	11:06:01	11:41:46	00:35:45	28	501	14	737	3056
TZ3	119	11:45:19	12:22:45	00:37:26	52	1011	33	1484	5982
TZ4	172	08:29:13	09:06:50	00:37:37	70	1377	44	1987	7784
TZ5	233	10:16:43	11:01:35	00:44:52	111	2105	72	2951	11708
TZ6	353	04:17:25	05:06:00	00:48:35	141	3997	92	6475	13834
TDB5	233	07:52:00	08:36:00	00:44:00	95	2224	72	3199	13298
TDB7	501	10:40:00	11:33:00	00:53:00	274	4497	81	7238	19392
TRC5	231	09:27:02	10:07:06	00:40:04	97	2387	71	3574	14514
TRC7	501	11:19:52	12:01:20	00:41:28	136	4866	94	8690	20840
TRC7N	504	08:12:01	08:54:04	00:42:59	150	4826	109	8899	21433
TC2	25	13:41:28	14:30:15	00:48:47	4	166	2	402	1082
TC3	35	16:22:00	17:16:18	00:54:18	4	318	3	741	1092

Table 9: Transaction Data: Elapsed @ different tests and normalized @ 1 hour

		ENTE		PERSONE		ACTIVITIES
Test	Processes	Creation	Update	Creation	Update	Creation
TP2	61	26.4	751.8	23.6	1,199.1	5,185.5
TP3	117	72.9	1,630.2	56.3	2,408.8	9,731.4
TP4	173	26.8	1,459.1	47.6	2,385.1	9,156.3
TP5	229	0.0	1,978.4	64.3	3,451.6	14,421.8
TZ2	63	47.0	840.8	23.5	1,236.9	5,129.0
TZ3	119	83.3	1,620.5	52.9	2,378.6	9,588.2
TZ4	172	111.7	2,196.4	70.2	3,169.3	12,415.8
TZ5	233	148.4	2,815.0	96.3	3,946.4	15,657.1
TZ6	353	174.1	4,936.3	113.6	7,996.6	17,084.9
TDB5	233	129.5	3,032.7	98.2	4,362.3	18,133.6
TDB7	501	310.2	5,090.9	91.7	8,194.0	21,953.2
TRC5	231	145.3	3,574.5	106.3	5,352.1	21,734.8
TRC7	501	196.8	7,040.8	136.0	12,574.0	30,154.3
TRC7N	504	214.0	6,886.1	155.5	12,697.7	30,582.2
TC2	25	4.9	204.2	2.5	494.4	1,330.8
TC3	35	4.4	351.4	3.3	818.8	1,206.6

To compare the results between the benchmark and the comparison test (TC2, TC3), we projected the number of processes to 500, as shown in Table 10.

Table 10: 500 Processes Projection

		ENTE		PERSONE		ACTIVITIES
Test	Processes	Creation	Update	Creation	Update	Creation
TC2	500	98.4	4,083.4	49.2	9,888.6	26,615.6
TC3	500	63.1	5,019.7	47.4	11,696.9	17,237.6

Note that the configuration we tested in Nashua shows better performance at all times with any kind of workload than the comparison test system at the office.

Database Activity

The Rdb database created on-site at SIAER before our visit to Nashua is the full Modena database; Modena (a province inside the Emilia-Romagna region) has the largest and most populated database of the region. We reorganized the database because after the first tests we discovered a

bad organization of the data: excessive fragmented record rate, and a slowness in inserting record operation due to excessive check on database pages. After the reorganization, we experienced better performance. To test different configurations, the Modena database was modified. We changed the parameter related to global buffer allocation (total number of global buffers, global buffers per process, memory location, and so on) and created Row Caches. Row Caches were allocated taking care of two topics:

- The database analysis told us the record sizing and allocation
- The knowledge of applications helped us identify where and how Row Caches have to be applied.

We did not perform any optimization on indexes; we used the original ones at database creation before visiting Nashua.

The entire database was loaded on a single volume of storage array MA8000; it performed fine in any test. The I/O rate of the database has been always lower than the I/O rate of the disks where application data (called KB inside SIAER applications) are stored: the I/O rate was between 800 and 900 I/O per second on the database disk for the test with largest number of disturb processes.

Final Considerations

All tests validated the proposed configuration. This architecture can carry the workload of the biggest province (see database), Modena, performing better than the current smaller databases at local offices.
If we look at the results, the better configuration is to have separate database and application systems – a database machine and an application machine – with row cache: this configuration provided significant performance increases.

The presence of the "visibility table," a new table we included specifically for the database consolidation, adds one more join level for each record (ENTE, PERSONA, ACTIVITIES); that makes the search operations across the overall database more expensive in contrast to performing current search operations on single and smaller databases located at each local offices.

The use of row cache on the "visibility table" makes the difference, because we do not perform any I/O. The usage of PK_ENTITA, as a sorted index, for search operations (RAGIONE_SOCIALE, COGNOME) in the database, is performed without accessing the records and avoids heavy I/O. Otherwise, any search operation could involve a huge number of tables. In those cases, splitting the query into two or more queries and using fewer tables for each query provides better performance.

A simple consideration we did after the benchmark, and one we are investigating now, is to consolidate the 180 single/local databases into 15 databases. The number of provinces (Modena is one of them and has the biggest database after consolidation) requires more attention on how queries are performed because a "not optimized" query may create bad performances. As stated earlier, the consolidated database of the test province of Modena is approximately 100 GB. A specific guideline will be provided to SIAER programmers in order to optimize the code of applications due to huge dimension of many tables inside the consolidated database (Golem).

Looking Forward

We are reviewing the following recommendations to revise the final SiDist application in the new consolidated environment:

- Database configuration and management: In creating a single consolidated database from 15 databases, some of them approximately 80-100 GB, it is extremely important to have the correct configuration of the database as well as to test and validate all applications involved. Our experience showed that an application that runs fine in a distributed environment with many smaller databases may have performance problems running in a single consolidated database.

- Security: Security is more critical for a consolidated database. A security problem on a local database can generate a corruption or loss of data only in a single office; for a consolidated database the consequences can be much greater. We are planning a "security project" in the near future because of this issue and, because we are also developing external Web-based applications running together with existing applications and databases. All the data we manage is sensitive data, confidential and classified, and requires the implementation of a secure environment.

- Applications: We will review the current set of applications in relation to the following criteria when they run with the single large database:
    - Query optimization: prudent and cautious usage of the "visibility table", key factors for faster and wider queries, and security of data management
    - Re-use of compiled queries, dynamic query optimization
    - Assure security on data access and on qualification of database operation.

Appendix

Test Summary

In this appendix you will find details of the tests we performed and discussed in this document. Note that in the beginning tests were performed by some recorded (using DTM) interactive session, but we discontinued these later due to the instability of the DTM recorded operation.

We used the terminology TPSS to mean transactions per second as reported by the database monitor process. The database monitor collects data for a 90-second time frame when "osservati" processes run.

"Launch end" means times when all batch disturb processes are up and running (each batch log reports start-time of processes).

"Test end" means when all disturb processes completed or, for some processes, when they have been killed.

Global Buffer in Standard Memory (32-Bit)

With the single system, named Sia047 (16 GB, 4 CPU), and the database configured with 81550 global buffer in 32-bit system memory, we ran tests with configurations 2, 3, 4 and 5 of Table 3. For each configuration, user buffers were allocated when the database was opened in order to allow connections to "disturb" processes. The results are as follows:

TP2

Date: August 2, 2002
Configuration: 2 (Table 3), P (Table 4)
User DTM1: PPROD01
User DTM2: PPROD50
User "osservati": PPROD04
Launch start: 08:09:43
Launch end: 08:12:05
Launch DTM: 08:13:23
Launch Osservati: 08:13:57
Test end: 09:15:43
TPSS: 14.2 (with 63 active processes)

TP3

Date: August 2, 2002
Configuration: 3 (Table 3), P (Table 4)
User DTM1: PPROD01
User DTM2: PPROD50
User "osservati": PPROD04
Launch start: 09:26:18
Launch end: 09:29:31
Launch DTM: 09:29:44
Launch Osservati: 09:52:33
Test end: 10:02:31

TPSS: 24.1 (with 119 active processes)

TP4

Date: August 2, 2002
Configuration: 4 (Table 3), P (Table 4)
User DTM1: PPROD01
User DTM2: PPROD50
User "osservati": PPROD04
Launch start: 10:50:20
Launch end: 10:55:17
Launch DTM: 10:55:20
Launch osservati: 10:55:20
Fine DTM: 11:18:02
Test end: 11:30:38
TPSS: 32.2 (with 175 active processes)

TP5

Date: August 2, 2002
Configuration: 5 (Table 3), P (Table 4)
User DTM1: PPROD01
User DTM2: PPROD50
User "osservati": PPROD04
Launch start: 11:32:10
Launch end: 11:44:24
Launch DTM: 11:44:44
Launch osservati: 11:44:44
Test end: 12:30:00
TPSS: 39.5 (with 231 active processes)

Global Buffer in VLM (64-Bit)

With the single system Sia047 (16 GB, 4 CPU) and the database configured with 524000 global buffer in 64bit system memory (VLM: Very Large Memory), we ran tests with configuration 1, 2, 3, 4, 5, and 6 of Table 3. For each configuration, we allocated 700 buffers per user. The results are as follows:
TZ1

Date: August 7, 2002
Configuration: 1 (Table 3), Z (Table 4)
Partial collection on "Osservati processes"
Test start: 03:43:01
Test end: 03:47:52

TZ2 (version TZ2B)

Date: August 6, 2002

Configuration: 2 (Table 3), Z (Table 4)
User "osservati": PPROD04
Launch start: 11:06:01
Launch end: 11:07:35
Launch osservati: 11:09:14
Test closing: 11:39:45
Test end: 11:41:46
TPSS: 12.5 (with 63 active processes)

TZ3 (version TZ3B)

Date: August 6, 2002
Configuration: 3 (Table 3), Z (Table 4)
User "osservati": PPROD04
Launch start: 11:45:19
Launch end: 11:46:34
Launch osservati: 11:49:55
Test closing: 12:20:39
Test end: 12:22:45
TPSS: 25.5 (with 119 active processes)

TZ4 (version TZ4B)

Date: August 6, 2002
Configuration: 4 (Table 3), Z (Table 4)
User "osservati": PPROD04
Launch start: 08:29:13
Launch end: 08:31:20
Launch osservati: 08:41:13
Test closing: 09:02:39
Test end: 09:06:47
TPSS: 37.0 (with 175 active processes)

TZ5

Date: August 6, 2002
Configuration: 5 (Table 3), Z (Table 4)
User "osservati": PPROD04
Launch start part 1: 10:16:43
Launch end part 1: 10:20:00
Launch start part 2: 10:24:04
Launch end part 2: 10:24:54
Launch osservati: 10:27:40
Test closing: 10:56:06
Test end: 11:01:40
TPSS: 45.1 (with 231 active processes)

TZ6

Date: August 7, 2002
Configuration: 6 (Table 3), Z (Table 4)
User "osservati": PPROD04
Launch start part 1: 04:11:15
Launch end part 1: 04:13:05
Part 1 processes active at: 04:16:05
Launch start part 2: 04:17:25
Launch end part 2: 04:19:10
Part 2 processes active at: 04:25:27
Launch osservati: 04:26:39
Osservati processes active at: 04:27:55
Test closing: 05:00:00
Test end: 05:06:00
TPSS: 61.2 (with 356 active processes)

Database Machine, Global Buffer in VLM

With the system Sia047 (16 GB, 4 CPU) configured as the application server and the second system, named Sia048 (12 GB, 4 CPU) configured as the database server, and the database configured with 524000 global buffer in 64-bit system memory (VLM: Very Large Memory), we ran tests with configuration 1, 5, and 7 of Table 3. For each configuration, we allocated 700 buffers per user.
TDB1

Date: August 7, 2002
Configuration: 1 (Table 3), Z (Table 4)
Partial collection on "Osservati processes"
Test start: 07:36:48
Test end: 07:41:49

TDB5

Date: August 7, 2002
Configuration: 5 (Table 3), DB (Table 4)
User "osservati": PPROD04
Launch start part 1: 07:52:00
Launch end part 1: 07:54:33
Part 1 processes active at: 07:55:33
Launch start part 2: 07:56:32
Launch end part 2: 07:57:34
Part 2 processes active at: 07:58:40
Launch osservati: 07:59:43
Osservati processes active at: 08:00:58
Test closing: 08:30:50
Test end: 08:35:38
TPSS: 48.8 (with 231 active processes)

TDB7

Date: August 7, 2002
Configuration: 7 (Table 3), DB (Table 4)
User "osservati": PPROD04
Launch start part 1: 10:40:05
Launch end part 1: 10:41:44
Part 1 processes active at: 10:42:56
Launch start part 2: 10:44:18
Launch end part 2: 10:45:47
Part 2 processes active at: 10:47:50
Launch start part 3: 10:49:06
Launch end part 3: 10:51:07
Part 3 processes active at: 10:55:23
Launch osservati: 10:55:27
Osservati processes active at: 10:56:34
Test closing: 11:29:49
Test end: 11:33:53
TPSS: 79.8 (with 508 active processes)

Database Machine, Global Buffer and Row Cache in VLM

With the system Sia047 (16 GB, 4 CPU) configured as the application server and the second system Sia048 (12GB, 4CPU) configured as the database server, and the database configured with 524000 global buffers in 64-bit system memory (VLM: Very Large Memory) and two specific row cache configurations in VLM (see Table 5), we ran tests with configurations 5 and 7 of Table 3. For each configuration, we allocated 700 buffers per user.
TRC5

Date: August 8, 2002
Configuration: 5 (Table 3), RC (Table 4)
User "osservati": PPROD04
Launch start part 1: 09:21:36
Launch end part 1: 09:24:09
Part 1 processes active at: 09:25:56
Launch start part 2: 09:27:02
Launch end part 2: 09:30:04
Part 2 processes active at: 09:30:58
Launch osservati: 09:32:30
Osservati processes active at: 09:33:44
Test closing: 10:04:10
Test end: 10:07:06
TPSS: 49.0 (with 231 active processes)

TRC7

Date: August 8, 2002
Configuration: 7 (Table 3), RC (Table 4)
User "osservati": PPROD04

Launch start part 1: 11:10:00
Launch end part 1: 11:11:38
Part 1 processes active at: 11:14:11
Launch start part 2: 11:15:22
Launch end part 2: 11:16:55
Part 2 processes active at: 11:19:00
Launch start part 3: 11:19:52
Launch end part 3: 11:22:03
Part 3 processes active at: 11:26:46
Launch osservati: 11:27:58
Osservati processes active at: 11:29:33
Test closing: 11:58:25
Test end: 12:01:20
TPSS: 93.1 (with 508 active processes)

TRC7N

Date: August 9, 2002
Configuration: 7 (Table 3), RCN (Table 4)
User "osservati": PPROD04
Launch start part 1: 08:02:31
Launch end part 1: 08:04:16
Part 1 processes active at: 08:06:07
Launch start part 2: 08:07:14
Launch end part 2: 08:08:48
Part 2 processes active at: 08:10:07
Launch start part 3: 08:12:01
Launch end part 3: 08:14:05
Part 3 processes active at: 08:18:58
Launch osservati: 08:20:04
Osservati processes active at: 08:21:16
Test closing: 08:50:59
Test end: 08:54:04
TPSS: 97.8 (with 508 active processes)

Comparison Test

To better evaluate results, we got ready a comparison system that looks like a typical system at local offices (Foa001); it's a DS20 single CPU@500MHz processor and 1GB memory; disks are as follows:

- DKA0 (9GB) OpenVMS operating system and layered products
- DKA100 (9GB) database: user data
- DKA200 (36GB) application: user space - database: root file
- DKA300 (36GB) not used
- DKC0 (18GB) database: rdb$system space, user space
- DKC100 (36GB) database: RUJ
- DKC200 (36GB) database: user space
-

The comparison system refers to a local office with a number of users between 25 and 35; we adjusted the scripts launching "disturb" processes in order to emulate a typical office workload and related number of users. Osservati processes remains as before.

Table 11: Comparison Test Configuration

Configuration TC1 (2 users)	• Only Osservati Processes (2 procedures) o 1 x `Gol_osservato.sic` o 1 x `Gol_osservato_l.sic`
Configuration TC2 (25 users)	• Disturb Processes Ufficio (23 procedures) o 1 x `Gol_ente_rag.sic` o 5 x `Gol_ente_rag_nonew.sic` o 1 x `Gol_pers_cognomi.sic` o 5 x `Gol_pers_nonew.sic` o 5 x `Gol_genmovu[1|2|3|4|5].sic` o 1 x `Gol_repcon.sic` o 1 x `Gol_reppag.sic` o 4 x `Gol_crediti.sic` • Osservati Processes (2 procedures) o 1 x `Gol_osservato.sic` o 1 x `Gol_osservato_l.sic`
Configuration TC3 (35 users)	• Disturb Processes Ufficio (33 procedures) o 1 x `Gol_ente_rag.sic` o 10 x `Gol_ente_rag_nonew.sic` o 1 x `Gol_pers_cognomi.sic` o 10 x `Gol_pers_nonew.sic` o 5 x `Gol_genmovu[1|2|3|4|5].sic` o 1 x `Gol_repcon.sic` o 1 x `Gol_reppag.sic` o 4 x `Gol_crediti.sic` • Osservati Processes (2 procedures) o 1 x `Gol_osservato.sic` o 1 x `Gol_osservato_l.sic`

Comparison tests provided the following results:

				ENTE		PERSONE		ACTIVITIES	OSSERVATO		OSSERVATO_L	
test	date	start	end	creation	update	creation	update	creation	elapsed	CPU	elapsed	CPU
Tc1	27/08								05:20.50	03.90	05:59.38	09.53
Tc2	27/08	13:41:28	14:30:15	4	166	2	402	1082	07:22.06	04.55	09:08.86	10.96
Tc3	27/08	16:22:00	17:16:18	4	318	3	741	1092	07:31.72	04.71	09:18.77	11.45

Test Summary for Comparison

Tests run on local DS20 single CPU processor running at 500 Mhz and 1 GB memory.
TC1

Partial collection on “Osservati processes”
TC2

Date: August 27, 2002
Configuration: 2 (Table 8)
User “osservati”: PPROD04
Launch start: 13:41:28
Launch end: 13:41:37
Processes active at: 13:48:39
Launch osservati: 13:48:42
Osservati processes active at: 13:51:02
Test closing: 14:29:13
Test end: 14:30:15
TPSS: 5.1 (with 25 active processes)
TC3

Date: August 27, 2002
Configuration: 3 (Table 8)
User “osservati”: PPROD04
Launch start: 16:22:00
Launch end: 16:22:04
Processes active at: 16:32:08
Launch osservati: 16:32:25
Osservati processes active at: 16:35:28
Test closing: 17:14:40
Test end: 17:16:18
TPSS: 6.0 (with 35 active processes)

Best of the HP Customer Support Center

Mark ‘Jilly’ Jilson
Service/Support Consultant
Customer Solutions & Support Center
OpenVMS Internals, Drivers & Performance

Overview

One of the most frequent problem statements that the CSC receives is one asking help in finding out why some system behavior changed. These might be questions like “why is a job taking longer to run?” or “why is a system running slower?” or “why is that backup job using more tape?” etc. One of the basic steps in responding to and analyzing such problems is to compare before and after profiles of the involved areas, but the CSC very often finds that there are no profiles to compare. This article describes some of these profiles and methods for collecting them. These profiles should be stored in at least two places outside of the system/cluster they cover and I advocate at least one paper copy so that when you really need them they can be accessed. With before and after profiles, it becomes a lot easier to pinpoint what area has changed and often directly answers the "What Changed?" question.

What Changed?

What devices are there on the system? What are their hardware revision levels and what are their firmware versions? How do the system devices look from the utilities that can see them?

Configuration Profile

Start with the device listing that the console can provide. With the OpenVMS system shut down (in a cluster, preferably have all nodes shut down), capture the output from the available console SHOW commands. Available items will vary from system to system but the following commands should work for most systems. If any of these commands don't work, then check the system's hardware User's Guide or consult with your hardware services representative or the CSC. To capture this information without transcribing it requires the use of a hardcopy console or a console management setup.

For Alpha systems use:

```
>>> SHOW CONFIG
>>> SHOW DEVICE
>>> SHOW MEMORY
>>> WWIDMGR -SHOW WWID -FULL
```

For VAX systems use:

```
>>> SHOW CONFIG
>>> SHOW DEVICE
>>> SHOW MEMORY
>>> SHOW SCSI
>>> SHOW DSSI
>>> SHOW VERSION
```

With OpenVMS booted on all cluster nodes, use the following command to capture configuration information:

$ SHOW DEVICE/FULL/OUTPUT={file.ext}

Editing this output to remove template devices and transient devices, like LTA or TNA, is recommended. But leave in any devices that are specifically created at startup like printer ports.

On OpenVMS Alpha, use the following commands to capture configuration information:

```
$ ANALYZE/SYSTEM
SET OUTPUT {file.ext}
CLUE CONFIG
CLUE SCSI/SUMMARY
EXIT
```

With the help of your hardware services representative and the CSC, annotate these displays with what the devices are, where they live in the system enclosure, card cage, etc., what external port they connect to on the system enclosure, what hardware settings they have (SCSI ID, CI node number, etc.), what hardware revision they are at (if any) and what firmware they are running (if any). If you make use of the Golden Eggs diagrams, available at http://h18000.www1.hp.com/info/golden-eggs/ , then you can annotate these with references to this collected output. If any of the devices have logical names or are used by print queues or execution queues, or you have specific names for a device, note this information also.

For storage subsystems, collect the same type of information. Consult the User's Guide for the storage subsystem to determine what information is available and how to acquire it.

Hardware Setup Profile

This information is stored in the system console that selects boot options, device operating options, reboot options, etc.

For Alpha systems, use the following command at the console to collect this information:

```
>>> SHOW *
```

For VAX systems, use the following commands at the console to collect this information:

```
>>> SHOW BFLG
>>> SHOW BOOT
>>> SHOW ETHERNET
>>> SHOW HALT
>>> SHOW SCSI_ID
```

Annotate these displays with descriptions for what disk device and root the system is booting from, operating options for network cards, operating options for SCSI cards, etc.

System Setup Configuration

This is the information that is used at boot time to configure the devices on the system and to set operating parameters.

For OpenVMS Alpha, collect output from the following commands:

```
$ DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$SYSTEM:ALPHAVMSSYS.PAR
$ DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$SYSTEM:ALPHAVMSSYS.OLD
$ DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$SYSTEM:AUTOGEN.PAR
$ DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$SYSTEM:SETPARAMS.DAT
$ DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$SYSTEM:PARAMS.DAT
$ DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$SYSTEM:MODPARAMS.DAT
```

For OpenVMS VAX, collect output from the following commands:

```
$ DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$SYSTEM:VAXVMSSYS.PAR
$ DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$SYSTEM:VAXVMSSYS.OLD
$ DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$SYSTEM:AUTOGEN.PAR
$ DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$SYSTEM:SETPARAMS.DAT
$ DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$SYSTEM:PARAMS.DAT
$ DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$SYSTEM:MODPARAMS.DAT
```

The dates and ids of these files will show when AUTOGEN was last executed, when parameters might have been changed with SYSBOOT or SYSGEN without an AUTOGEN, or if a parameter change was contemplated but not actually implemented.

Execute the following commands to capture system parameter information in a file:

```
$ MCR SYSGEN
USE CURRENT
SET/OUTPUT {file.ext}
SHOW/ALL
SHOW/SPECIAL
EXIT
```

The output file will now contain the settings of all the system parameters in the on-disk system parameter file.

Save a copy of the system files SYS$SYSTEM:MODPARAMS.DAT, SYS$SYSTEM:SYCONFIG.COM, and SYS$SYSTEM:SYS$DEVICES.DAT and add any other device creation commands that are executed at system startup time like LTA or TNA devices used for printing. With this information it will be very easy to determine if anything in the system setup might have been altered.

System Boot Profile

Save the console output from a normal system startup. If startup logs via STARTUP_P2 are implemented, then also save a copy of a log from a normal system boot. If possible, boot an OpenVMS Alpha system with full verbose boot options at the console by setting the boot flags to 30000 and save the console output from this. If possible boot the system with STARTUP_P2 set to TRUE and save the console output. Annotate these outputs with what portions of startup are executing.

Save the output from the following command:

```
DIRECTORY/DATE=(CREATED,MODIFIED)/FILE_ID SYS$STARTUP:*.COM,
SYS$SYSTEM:*.COM
```

Additionally gather this file information from any other system startup procedure that is executed.

After the system has booted normally, log in and save the output from the following commands:

```
$ ANALYZE/SYSTEM
SET OUTPUT {file.ext}
SHOW SUMMARY/IMAGE
EXIT
```

Annotate this output with descriptions of any application processes.

Performance Profile

A system performance profile, at minimum, should represent significant time periods in the system operation cycle and should cover a maximum 30 minutes of elapsed time. Save both the raw performance data and the performance reports for these time periods. OpenVMS provides the MONITOR utility and the ECP Data Collector and Performance Analyzer is freely available at http://h71000.www7.hp.com/openvms/products/ecp/index.html; either or both of these can be used.

OpenVMS provides examples of using MONITOR to gather performance data and generate a performance report in the following files: SYS$EXAMPLES, SUBMON.COM, MONITOR.COM & MONSUM.COM . These files need very little editing to be usable. The following are the minimum MONITOR commands needed to gather a performance profile that covers 30 minutes of elapsed time; these commands can be submitted via batch.

```
$ MONITOR
ALL_CLASSES/RECORD={recfil.ext}/NODISPLAY/INTERVAL=120/END="+0:30:0.0"
$ MONITOR ALL_CLASSES,DISK/ITEM=ALL,SCS/ITEM=ALL/NODISPLAY -
  /INPUT={recfil.ext}/SUMMARY={file.ext}
```

Both the recording file and the summary file are saved.

If MONITOR data is already being saved, then here is an example of a command for creating a report if the period from 9am to 9:30am is a peak operational time.

```
$ MONITOR
ALL_CLASSES,DISK/ITEM=ALL,SCS/ITEM=ALL/NODISPLAY/SUMMARY={file.ext} -
  /INPUT={recording file that covers the date}/BEGIN=12-MAY-2003:09:00:00.00 -
  /END=12-MAY-2003:09:30:00.00
```

Both the recording file and the summary file are saved.

If ECP is collecting data, then the following command is an example of creating a report if the peak time period of interest was 13:30 to 14:00.

```
$ PLAN ANALYZE/CPC_VMS_FILE=ECP$PERF_DATA:NODE_2003MAY12_1.CPC -
  /BEGIN=12-MAY-2003:13:30:00.00/END=12-MAY-2003:14:00:00.00 -
  /ANALYZE_REPORT_FILE={file.ext}
```

Both the report and the .CPC file are saved.

In addition to overall system performance reports that cover significant operation cycles, collect accounting information from significant jobs that execute on the system. These are jobs that you expect to complete in a certain elapsed time or are usually finished before a certain time. Also the accounting logs from backup jobs should be recorded. If any of these jobs have multiple parts to them, then adding SHOW PROCESS/ACCOUNTING commands to the job's command procedure(s) are beneficial. Then, when needed, different portions of a job can be compared before and after. If you have any application benchmark jobs or can create some simple benchmark jobs, save the accounting information from these. Finally, if you have application-provided performance data (for example, number of queries per minute or number of orders per hour, etc.) save this also.

Managing and Using the Profiles

Profiles should be refreshed periodically and after any changes are made to the system or the system's application load. Integrating these profiles into your system management structure will ensure that when the time comes to answer the question "What Changed?", you will be well prepared to quickly zero in on the area responsible for the change.

Best of Ask the Wizard

Steve Hoffman “Hoff”
HP OpenVMS Consulting Engineer

HP OpenVMS Support Resources and How to Use Them

HP provides a variety of support resources including web sites and customer support centers. In addition, contract support customers have access to associated databases and services. To aid its customers, HP workers also have access to a problem escalation process and problem tracking, discussion forums, and research tools.

Many of the available OpenVMS support resources are accessible to all OpenVMS customers. Among these, the OpenVMS Frequently Asked Questions (FAQ) and the OpenVMS Ask The Wizard area contain answers to many common questions, such as:

- How to reset a forgotten password on the SYSTEM username.
- How to set up or troubleshoot an IP printer.
- How to get support for OpenVMS questions.

The FAQ and the Ask The Wizard areas are both available at the following web site:

http://www.hp.com/products/openvms/

The HP Natural Language Search Assistant (AskQ) area provides direct access into the HP support databases; access to a subset of the source code examples and the support articles that are available to contract support customers. A natural-language search engine is provided at the AskQ web site:

http://www.itrc.hp.com/service/james/CPQhome.do

Major and active OpenVMS discussion forums include the Usenet newsgroup comp.os.vms and the DECUServe notes conferences:

news://comp.os.vms/
telnet://eisner.decus.org/

Search engines are also available for these forums.

Other newsgroups, web sites and discussion forums are available as well. The OpenVMS FAQ has a complete list of these support resources and pointers to common applications and to many of the available Freeware packages. If searching for commercial applications and options, the HP DSPP web site and its search engine are available for locating commercial applications for OpenVMS. The FAQ also has pointers to AskQ and to the online OpenVMS documentation web site. Pointers to example source code are also available within the FAQ.

ECO Kits

Information on the OpenVMS ECO (patch) kits is available by FTP file server and by search engines. The following ECO search engine can acquire lists of ECO kits by installation rating, making it easier for you to keep the current mandatory ECO kits installed for your particular OpenVMS release:

http://ftp.support.compaq.com.au/pub/ecoinfo/ecoinfo/top.htm

Email notifications of new ECO kits are also available. The ECO notification subscription web site is:

http://www.support.compaq.com/patches/mailing-list.shtml

Direct and Formal HP Assistance

For customers wishing to request direct and formal HP assistance, the customer support centers are the best initial contacts. When your request for support is received at the local or regional support center for your geography, the information you need to provide includes an initial description of the particular problem. Specific information on logging calls and on your support center telephone number are all available within your hardware or software support contract documentation.

To speed the resolution of your support call, please follow these tips:

- Be very specific in your problem description. Generic problem statements such as, "It doesn't work" or "It crashed", can and often do cover huge numbers of potential problems and even larger numbers of potential causes.
- Provide the product version and OpenVMS platform information, as many problems can be version- or platform-specific.
- Reference the specific commands or utilities that might provoke the problem, the full text of any error messages displayed, and the expected outcome.
- Provide information on any installed ECO kits.

The more general the problem statement, the longer it often takes to determine the details of the problem and to then provide you with the resolution.

Regardless of the nature of the particular problem, providing HP with a method to easily reproduce the reported problem can be invaluable in providing you with the quickest response. Accordingly, HP will often request a reproducer, a way to trigger and to localize the problem and to subsequently verify the correctness of the resolution. Note that the smaller, simpler, and more targeted the reproducer; the faster HP can obtain a resolution.

Software Code Reproducer

An example of a concise software source code reproducer follows:

```
$ set noon
$ if f$search("sys$share:zzzshr.exe") .nes. ""
$ then
$   known = f$file_att("sys$share:zzzshr.exe","known")
$   if known
$   then
$     install delete sys$share:zzzshr.exe
$   endif
$   delete sys$share:zzzshr.exe;*
$ endif
$ if f$search("sys$scratch:zzz.exe") .nes. ""
$ then
$   known = f$file_att("sys$scratch:zzz.exe","known")
$   if known
$   then
$     install delete sys$scratch:zzz.exe
$   endif
$   delete sys$scratch:zzz.exe;*
$ endif
$ cc zzz/def=ZZZ/obj=sys$scratch:zzz.obj
$ cc zzz/def=ZZZSHR/obj=sys$scratch:zzzshr.obj
$ goto 'f$getsyi("ARCH_NAME")'
$Alpha:
$ link/notrace/nodebug -
sys$scratch:zzzshr/share=sys$share:zzzshr.exe,sys$input/opt
symbol_vector=(ChkPrv=procedure)
gsmatch=lequal,1,0
identification="zzzshr v1.0"
$ goto common
$VAX:
$ macro sys$input/object=sys$scratch:zzzxfr.obj
  .title $$$xfrvec transfer vector(s)
  .ident /zzzxfr v1.0/
  .psect $$$xfrvec,exe,shr,nowrt,rd,pic,quad
  .macro xfrvec entrypoint
  .align quad
  .transfer entrypoint
  .external entrypoint
  .mask entrypoint
  jmp l^entrypoint+2
  .endm
  xfrvec ChkPrv
  .end
```

```
$ link/notrace/nodebug -
sys$scratch:zzzshr,sys$scratch:zzzxfr/share=sys$share:zzzshr.exe,sys$input/opt
cluster=$$$xfrvec
collect=$$$xfrvec,$$$xfrvec
gsmatch=lequal,1,0
identification="zzzshr v1.0"
$ goto common
$Common:
$ link/notrace/nodebug -
/execu=sys$scratch:zzz.exe sys$scratch:zzz,sys$input/opt
sys$share:zzzshr/share
$
$
$! SYSLCK disabled, not installed, not installed with SYSLCK privilege
$
$ set process/privilege=nosyslck
$ run sys$scratch:zzz
$
$! SYSLCK enabled, not installed, not installed with SYSLCK privilege
$
$ set process/privilege=syslck
$ run sys$scratch:zzz
$ set process/privilege=nosyslck
$ install create sys$share:zzzshr
$
$! SYSLCK disabled, installed, not installed with SYSLCK privilege
$
$ run sys$scratch:zzz
$ install create sys$scratch:zzz
$ run sys$scratch:zzz
$
$! SYSLCK disabled, installed, installed with SYSLCK privilege
$
$ install replace sys$scratch:zzz/priv=syslck
$ run sys$scratch:zzz
$
$! clean up...
$
$ install delete sys$scratch:zzz
$ install delete sys$share:zzzshr
$ delete sys$scratch:zzz.exe;*
$ delete sys$share:zzzshr.exe;*
$
$ exit
```

--

```
#include <prvdef.h>
#include <ssdef.h>
#include <starlet.h>
#include <stdio.h>
#include <stdlib.h>
#ifdef ZZZ
main()
  {
  int RetStat, ChkPrv();
  RetStat = ChkPrv();
  return 1;
  }
#endif
#ifdef ZZZSHR
int ChkPrv()
  {
  int RetStat;
  int PrvQWIn[2] = {0,0}, PrvQWOut[2]= {0,0};
  RetStat = sys$setprv( 0, PrvQWIn, 0, PrvQWOut );
  if ( PrvQWOut[0] & PRV$M_SYSLCK )
    printf("SYSLCK enabled\n");
  else
    printf("SYSLCK disabled\n");
  return RetStat;
  }
#endif
```

This example shows a complete and concise problem reproducer that was constructed by HP in response to a problem report reputing errors within the OpenVMS handling of the SYSLCK privilege and installed images. Based on this reproducer, the problem report was shown to be incorrect or incomplete, and there were additional factors involved in the problem trigger.

System Bugchecks

The most serious OpenVMS problems can involve a system bugcheck. When an unrecoverable error is detected within OpenVMS, a bugcheck system crash is triggered. By default, OpenVMS is configured to write the system state to the system dumpfile or potentially to the system pagefile during the bugcheck processing. The contents of this system dump file can be central to the resolution of fatal system failures. Your own applications can utilize similar process-state, logging mechanisms; process-level dumps can be generated upon application failures. For details on configuring and utilizing the process dump mechanism, please see the debugger documentation and the details of the ANALYZE/PROCESS_DUMP command.

If OpenVMS generates a bugcheck, you will want to acquire a synopsis of the crash. The ANALYZE/CRASH system Dump Analyzer (SDA) CLUE CRASH callout easily provides this synopsis of the system dumpfile, and -- when the CLUE CRASH output is written out to a file -- the synopsis can be provided to and examined by HP using HP-internal automated scanning tools. When compared against known crashes, this synopsis can speed the resolution of known problems. Whether the bugcheck is known and an answer is available, or if the bugcheck is a previously-unknown problem, the synopsis typically helps quickly isolate the particular cause and correlate this report with any other similar reports.

Summary

HP offers services that can help you to avoid, or to even weather, the occurrences of many problems, either by the preemptive application of critical ECO fixes, or by correctly configuring your OpenVMS systems and clusters for best reliability. Service offerings ranging from installation assistance, system healthcheck offerings, system management outsourcing, and consulting services such as custom programming and disaster-tolerant cluster configurations are all available.

In addition to the resources and services already mentioned here, additional OpenVMS support information and services are available to you. Further, if you are unsure of where to find the information you need or potentially how to best utilize the resources available to you, please see the OpenVMS Frequently Asked Questions (FAQ) and the HP services web site. If you do decide to utilize formal HP assistance, you can help expedite the response by providing HP with critical information. With access to the appropriate information and to the available HP services, you can speed the resolution and speed your OpenVMS systems back into the business of serving your own customers.

Server-Agnostic Perl/DCL CGI Programming with WASD and OSU

Dick Munroe
Cottage Software Works, Inc

Overview

In January of 2003, a client of mine decided to switch from Purveyor, on which they were running their secure commerce web site, to the WASD web server. The OSU server was also in use at the site, but my clients decided that the features provided by WASD were better than those available in OSU. I was asked to install the WASD server and move the secure portion of their applications from Purveyor to WASD. During this job I suggested that my clients stop using the OSU server as well, purely for support reasons. After all, one web server is less work to support than two. To avoid a "flag day", where we would have to switch an entire application from one web server to another, I designed a general framework that allowed them to use their current DCL/Basic CGIs in either environment. Since the clients followed a consistent pattern for implementing their CGIs, it was also possible to write tools that prepared their applications for execution under either WASD or OSU. In turn, this made the switch from the Purveyor/OSU environment to the WASD/OSU environment quick and simple, while leaving the elimination of OSU a decision that could be made on a case-by-case basis.

I was so impressed with the quality of the WASD server[10], its documentation, the provided debugging tools, and the commitment of the WASD development group that I decided to switch from OSU to WASD at my site.

Hedging My Bets

Having made the decision to switch from OSU to WASD, I wanted to hedge my bets. A fair portion of my site has functionality implemented by CGIs. I didn't want to lock myself irretrievably into WASD (or into any specific server if possible). So I had to look into the same issues that my clients had, i.e., how to build CGIs that run under all servers.

While I do occasionally write CGIs in a compiled language like C, the job can generally be done quite easily with DCL, Perl, or some combination of both. The net result is that most of the CGIs at my site are written in one of these languages.

So, what execution environments does each server offer for DCL and Perl?

- OSU

  DCL can execute only under the "script server," which uses DECnet to create an execution environment and communicate with the CGI.

  Perl also executes through the "script server" environment with some special case code in WWWEXEC.COM to ensure that Perl CGIs have an appropriate execution environment. OSU also provides an execution environment using the FastCGI interface specification.

[10] I believe the WASD and OSU servers to be, more or less, equivalent in terms of performance and overall capabilities. The WASD web server and environment is much better documented than the OSU distribution and is therefore much easier to use and manage, giving WASD the edge.

- WASD

  DCL can execute in one of three environments. The first environment is an OSU emulation. In practice I saw no differences between the OSU server and the WASD OSU emulation. The second environment involves execution of the CGI as a sub-process of the server. The third, CGIplus (analogous to FastCGI), dedicates a process to running a DCL CGI.

  As with DCL, Perl can also execute under the OSU emulation environment, directly as a sub-process, or with a dedicated process. WASD provides one additional execution environment, PerlRTE. PerlRTE is a persistent Perl interpreter, analogous to mod_perl. CGIplus is a persistent Perl "server", analogous to FastCGI: a process dedicated to the repetitive execution of a single Perl script.

PerlRTE and CGIplus are performance optimizations. They address the two principal overhead components in execution of any Perl CGI or program:

1. Creation of the process running Perl and loading of the Perl interpreter (PerlRTE)
2. Loading of the Perl modules necessary for execution of the CGI or program (CGIplus)

PerlRTE creates a process and loads the interpreter but can execute any Perl.CGI. The cost of creating the process and loading the Perl interpreter is amortized across all CGIs using PerlRTE.

CGIplus creates a process and loads the Perl interpreter (when executing a Perl CGI) but then goes on to load and execute the CGI. The Perl CGI is wrapped in a loop and is available for "instant" execution the next time that CGI is invoked. The cost of creating the process, loading the Perl interpreter, and loading all the necessary Perl modules required by the CGI is amortized across the number of invocations of a specific CGI. Each CGIplus-enabled CGI requires a dedicated process for execution. Unfortunately, each of these environments differs in small ways that must be accounted for in a server-agnostic CGI.

Perl CGI Programming in the Different Environments

This document is not a tutorial on how to program CGIs or how to program in Perl. Both of those topics have been covered in the literature more thoroughly and far better than I can do. However, a quick discussion of the basics is in order.

Outside of mod_perl, the most commonly used CGI programming interface is the venerable CGI.pm. It provides access to the CGI environment variables, access to query, form, and multipart form data. CGI.pm can also generate http protocol headers and many of the standard HTML tags, making creation of dynamic content much easier.

Listing 1 shows the basic structure of a simple Perl CGI using CGI.pm.

```
use strict;
use 5.6.1 ;
use CGI ;
my $theCGI = new CGI ;
print $theCGI->header('text/plain') ;
my @theParameterNames = $theCGI->param() ;
print "The Parameter/Value pairs:\n" ;
print "-------------------------\n\n" ;
foreach (sort @theParameterNames)
{
    my @theValue = $theCGI->param($_) ;
    if ($#theValue)
    {
        for (my $i = 0; $i <= $#theValue; $i++)
```

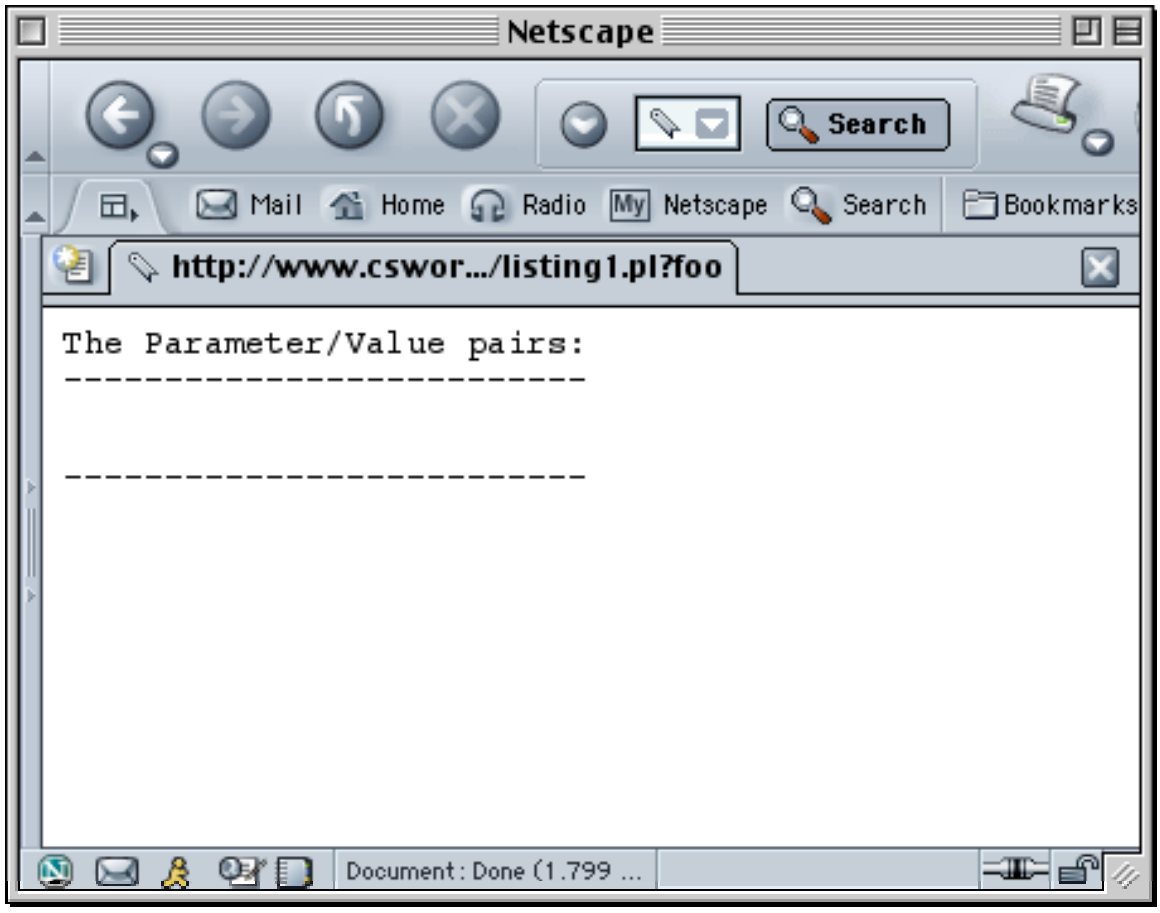
```
        {
            print $_,"[$i] = ",$theValue[$i],"\n" ;
        }
    }
    else
    {
        print $_," = ",$theValue[0],"\n" ;
    }
}
print "\n--------------------------\n" ;
```

Listing 1
CGI to list the names/values of query/form parameters.

As can be seen, the CGI is pretty straightforward. The CGI is invoked, goes through the hash of parameter names in alphabetical order and prints the values.

You can see the CGI in action at http://www.csworks.com/cgi-bin/vtj/listing1.pl?foo=1. If you download the source for this article and put the listing1.pl CGI in CGI-BIN:[000000] so your WASD server can find it, fire up a browser and try http://your.server.name/cgi-bin/listing1.pl?foo=1 you will, likely, see something like Figure 1. What happened to the value of foo? Why didn't it print?

Figure 1
What happened to the Parameters?



If you didn't get a display at all or got an error message, you probably have a simple configuration issue. You need to tell WASD how to execute Perl code. Full details are available in the WASD documentation, but you need to make sure that there is a mapping from the .pl suffix to a bit of DCL that executes the Perl procedure in the right environment. The mapping is kept in HT_AUTH:HTTPD$CONFIG.CONF and at my site looks like this:

```
[DclScriptRunTime]
.pl @cgi-bin:[000000]perl.com
```

CGI-BIN:[000000]PERL.COM is:

```
$ define /user perl_env_tables clisym_global,lnm$process
$ perl "''p1'"
$ exit
```

and is provided with the WASD distribution in the right place for use. Once the linkage between file type (.pl) and execution environment (CGI-BIN:[000000]PERL.COM) is properly set up, you can move on.

The second possibility is the first lesson in server agnosticism. Both WASD and OSU prefix their CGI environment variables (those defined by the CGI Interface Specification) with "WWW_". This is intended to do two things:

1. Identify the CGI variables as belonging to the World Wide Web environment and
2. Prevent the standard CGI variable names from "masking" the equivalent DCL symbols or OpenVMS logical names, both of which are provided to Perl programs as part of the general environment available via the %ENV hash.

Unfortunately, we are dealing with Perl and CGI.pm whose roots are deep in the Unix world. The Unix CGI environment doesn't prefix its CGI environment variables with "WWW_". Rather the CGI environment variables are provided unmodified as per the CGI Interface Specification. Since WASD is prefixing the CGI environment variables with "WWW_", CGI.pm and any other Perl code following the CGI Interface Specification won't see the CGI environment variables.

In order to bolt Perl CGIs using CGI.pm to WASD you must add to your mapping files something like the following:

```
set /cgi-bin/vtj/*            cgiprefix=
```

before the mapping of the CGI via the exec occurs. Once these lines have been added, WASD will apply the specified prefix (none at all) to all CGIs in the cgi-bin/vtj directory. Of course you can make this as specific as you want, e.g.:

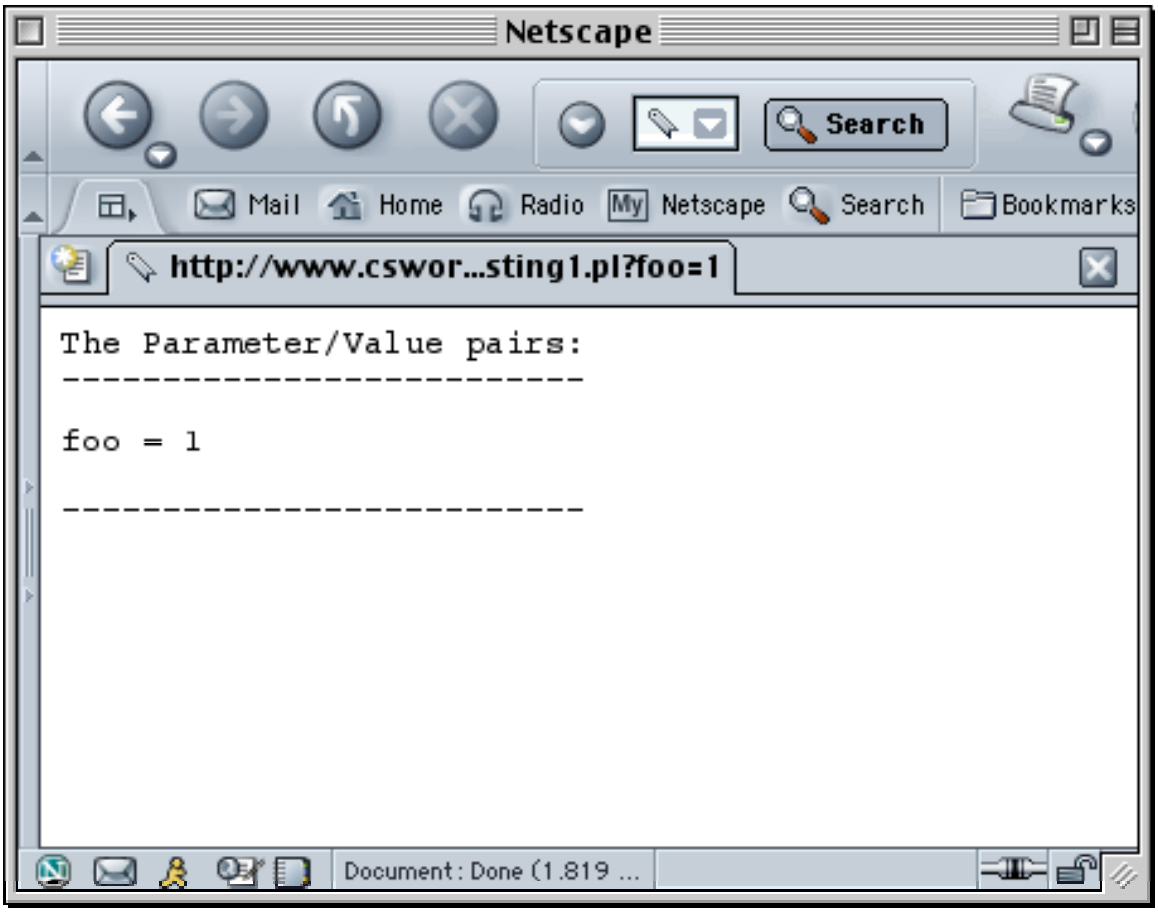
```
set /cgi-bin/vtj/listing1.pl        cgiprefix=
```

provides an empty (zero length) CGI prefix for only listing1.pl.

Once you've done this, reloaded the WASD server's mappings[11], and executed listing1.pl again on your server, you should see something like Figure 2.

[3] For the WASD server, HTTPD/DO=MAP reloads the mapping files, HTTPD/DO=RESTART restarts the server reloading the entire configuration. Both have the same effect on mappings, but the RESTART may be visible by your users.

Figure 2
Parameters with cgiprefix set appropriately.



The OSU server has functionally equivalent magic to avoid breaking CGI.pm and other U*x oriented CGI code. It's built into WWWEXEC.COM and is not configurable short of modifying WWWEXEC.COM. However, WWWEXEC.COM makes reasonable assumptions about the Perl CGI execution environment.

If you run listing1.pl under OSU or under WASD's OSU emulation, you'll find that the same code runs identically under OSU, WASD with OSU emulation and WASD sub-processes.

The next CGI execution environment provided by WASD is CGIplus. The WASD kit includes a Perl module, CGIplus.pm, which provides CGIplus support. CGIplus.pm provides a number of useful interfaces, including a test to see if CGIplus mode is active, a usage counter, writing to the standard output stream in binary mode, access to CGI environment variables, etc. Unfortunately, the interfaces are provided without formal documentation[12]. You have to figure out what's there by inspection of the various examples provided in the WASD distribution or the CGIplus.pm code itself.

[12] One of the things that attracted me to the WASD web server was the thoroughness and high quality of the user documentation produced by the author. Finding a spot where the documentation was less than thorough was quite a shock.

The basic mechanisms for using CGIplus are simple. First, wrap what used to be your CGI in a subroutine, load CGIplus[13], and call the subroutine using the CGIplus process manager, CGIplus::process. Listing 2 shows the "obvious" port to use CGIplus.

```
unshift @INC,"HT_ROOT:[SRC.PERL]" ;

use strict;
use 5.6.1 ;

use CGI ;
require CGIplus ;

CGIplus::process(\&doit) ;

sub doit
{
    my $theCGI = new CGI ;
    print $theCGI->header('text/plain') ;
    my @theParameterNames = $theCGI->param() ;
    print "The Parameter/Value pairs (Usage: ",
        CGIplus::usageCount(),"):\n"  ;
    print "-------------------------\n\n" ;
    foreach (sort @theParameterNames)
    {
        my @theValue = $theCGI->param($_) ;
        if ($#theValue)
        {
            for (my $i = 0; $i <= $#theValue; $i++)
            {
                print $_,"[$i] = ",$theValue[$i],"\n" ;
            }
        }
        else
        {
            print $_," = ",$theValue[0],"\n" ;
        }
    }

    print "\n-------------------------\n" ;
    if ($ENV{'QUERY_STRING') eq "eoj")
    {
        exit ;
    }
}
```

Listing 2
Port to CGIplus

If you run http://www.csworks.com/cgiplus-bin/vtj/listing2.pl?foo=1 repeatedly, changing the value of the foo query parameter, you'll notice two things. First, the format of the output changes abruptly. All of a sudden there are a few extra new lines in the output. Second, and much more

[13] Since CGIplus isn't part of the standard Perl distribution, the unshift/require pair is necessary to get CGIplus loaded and ready for use. You could copy CGIplus.pm into your Perl library tree (at my site, I would put it in PERL_ROOT:[LIB.VMS_AXP.5_6_1]) and then use either a "require" or "use" statement.

important, the value of foo doesn't change from invocation to invocation, although the displayed usage count tells you that the CGI is, indeed, getting invoked. What's wrong?

This is the second lesson in server agnosticism. It's related to the first insofar as the problem is with CGI.pm, but the causes are completely different. CGI.pm predates CGIplus.pm by quite some time and CGI.pm has never been modified to take the WASD specific CGIplus environment into consideration. CGI.pm has been modified to look for Active States PerlEx and CGIplus.pm attempts to take advantage of that fact by asserting that CGI.pm is running in a PerlEx environment. By asserting "PerlEx" mode CGIplus.pm causes CGI.pm to reset its internal persistent state every time a new CGI object is created. All well and good, but listing2.pl still doesn't work!

What emerges is an ordering problem. CGIplus.pm asserts "PerlEx" mode when CGIplus::process is run. CGI.pm checks for the PerlEx environment when it loads (at the time the require or use statement loading CGI.pm is executed). As written, the CGI.pm loaded in listing 2 believes that it's running in a standard CGI environment and it never finds out about the persistent CGIplus environment.

So the lesson is to make sure that your CGI interface library stays sane across all environments. It may be necessary to build the occasional shim to make things work. If things get too complicated (and that is a judgement call), it's probably time to consider modifying the standard distribution of your CGI interface library and offer the modifications for general use.

Listing 3 shows the correct way to use CGIplus and fully implement server agnosticism as understood so far.

```
unshift @INC,"HT_ROOT:[SRC.PERL]" ;

use strict;
use 5.6.1 ;

my $useCGIplus = ($ENV{'CGIPLUSEOF'} ne undef)  &&
    ($ENV{'SCRIPT_RTE'} eq undef) ;
eval { require CGIplus ; } || die "Can't find CGIplus.pm" ;
if ($useCGIplus)
{
    #CGIplus::stripWWW(1);
    CGIplus::process(\&doit) ;
}
else
{
    doit()
}

sub doit
{
    require CGI if (!defined(&CGI::new)) ;
    my $theCGI = new CGI ;
    print $theCGI->header('text/plain') ;
    my @theParameterNames = $theCGI->param() ;
    my $theString = "The Parameter/Value pairs" ;
    $theString .= " (Usage: " . CGIplus::usageCount() . ")"
        if (CGIplus::isCGIplus()) ;
    $theString .= ":\n" ;
    print $theString ;
    print "-------------------------\n\n" ;
```

```
    foreach (sort @theParameterNames)
    {
        my @theValue = $theCGI->param($_) ;
        if ($#theValue)
        {
            for (my $i = 0; $i <= $#theValue; $i++)
            {
                $theString = $_ . "[$i] = " . $theValue[$i] . "\n" ;
                print $theString ;
            }
        }
        else
        {
            $theString = $_ . " = " . $theValue[0] . "\n" ;
            print $theString ;
        }
    }
    print "\n-------------------------\n" ;
    if ($ENV{'QUERY_STRING'} eq "eoj")
    {
        exit ;
    }
}
```

Listing 3
Fully Agnostic CGI

The third lesson in building server-agnostic CGIs is to ensure that the output of your CGI is constant across all environments. When http://www.csworks.com/cgiplus-bin/vtj/listing3.pl?foo=1 is run repeatedly, and the value of the query parameter varied, the display of the CGI output remains constant, the display of the usage counter comes and goes depending on the environment (CGI or CGIplus), and those pesky extra new lines have been eliminated.

However, those pesky new lines are important if CGIs that output binary data such as graphics are written. When the standard output data stream is opened for the second and later instances of a CGI executing in the CGIplus execution environment, the standard output stream is opened in `record` mode. This means that for every I/O operation presented to the standard output stream a record delimiter is added. In this case, an extra new line (or carriage return/line feed pair, I haven't verified which), is inserted at the end of each I/O operation. Since the standard output stream isn't buffering data, each collection of data gets written in a new operation, and each operation has a record delimiter, therefore the extra lines. By collecting the data to be presented into strings, and writing each fully formatted string in a single I/O operation, formatting is preserved.

We need to briefly address providing binary data output in a server-agnostic fashion. CGIplus.pm provides interfaces that write binary data to the standard output stream. These work for all WASD Perl execution environments `except` OSU emulation, or within OSU.

As was seen above, if the standard output stream is in record mode, spurious data is introduced into the CGI output stream. For HTML data this is largely harmless, but for binary data of any kind, it's fatal. The data stream is corrupted by the spurious carriage control. The necessary fix for the OSU server and WASD OSU emulation is to remove the change to record mode for the standard output stream. The following shows the portion of WWWEXEC.COM that is changed.

```
$ perl_script:
$   tfile = "sys$scratch:perlcgi_" + f$string(f$getjpi("0","PID")) + ".tmp"
$!  write_net "<DNETRECMODE>"
$   on warning then goto perl_done
```

Once this is done, the binary output interfaces provided by WASD's CGIplus.pm can be used. Note that a "typical" Perl CGI using CGI.pm and using a standard output stream that is not in record mode will break. CGI.pm is written based on record mode being OpenVMS's default behavior for the standard output stream. In a production environment WWWEXEC.COM should be modified so that record mode is the default except for some class of CGIs that could be detected by directory, file name or file extension. Implementation of this patch at your site is left as an exercise to the reader.

For the interested reader a server-agnostic CGI that does binary output has been implemented and can be downloaded from http://www.csworks.com/download/modularian.html.

The last execution environment available to Perl using the WASD server is PerlRTE. PerlRTE is essentially a persistent Perl Interpreter capable of running any Perl CGI. The state of the CGI is discarded upon CGI exit, so PerlRTE is similar to the standard CGI execution environment without the overhead of sub-process creation and loading of the Perl interpreter. A CGIplus script cannot be activated (the module detects and prevents it) using the PerlRTE in RTE mode (a seemingly subtle but very real distinction with WASD). As seen in listing 3, CGIplus::process is only executed if CGIPLUSEOF is defined in the Perl environment and the SCRIPT_RTE environment variable is undefined. Since the SCRIPT_RTE environment variable is always defined by PerlRTE then CGIplus::process will never get executed and the CGI in listing 3 is ready to execute correctly in the PerlRTE environment.

Apache for OpenVMS

One server has gone unmentioned in this article, not because I wished to ignore it but because I can't run a copy easily (my systems are still running 7.1) without breaking out an old box and spending a couple of days building up a new system. However since these server-agnostic CGIs rely upon CGI.pm and CGI.pm is well known to function in a mod_cgi or mod_perl environment, then I believe that server-agnostic CGIs developed for WASD and OSU should also run "out of the box" on Apache as well. Mark Daniels (the author of WASD) tested listing3.pl under Apache for OpenVMS[14] for me. The script runs unmodified. This makes server-agnostic programming even more useful since by being able to run under OSU, WASD, and Apache for OpenVMS, the large majority of OpenVMS web server installations are accounted for. Platform agnostic CGIs, those written in a "portable" language (Perl, Python, PHP, et al.) and using nothing but commonly available interfaces, can move from platform to platform without change, and can thus easily be made truly agnostic, caring about neither server nor platform.

Summary

Using two of the web servers commonly available for OpenVMS systems and a simple CGI, we've deduced three guiding principles for developing server-agnostic CGIs. These principles are:

1. Make sure that the execution environment meets the expectations of your CGI interface library
2. Make sure that the innards of your CGI interface library stay sane independent of the execution environment
3. Make sure the output of the CGI stays constant independent of the execution environment

The mechanisms used to build server-agnostic CGIs are not expensive (in terms of performance) or difficult (in terms of programming); all it takes is a little care.

[14] The test configuration was OpenVMS 7.3-1, CSWS 1.3, and CPQ Perl 5.6.1 or CSWS Perl 1.1.

The benefits are that your CGIs can be used "anywhere," allowing you and your customers more flexibility in terms of development, debugging, and deployment.

I'm interested in seeing server agnostic CGIs in as many environments as possible. So far I have accommodated WASD, OSU, and Apache for OpenVMS. If there are others in active use on OpenVMS, I'd love to know what has to change to make listing3.pl work on your server. In addition, please send be the results of listing3.pl on platforms other than OpenVMS. I can be reached at munroe@csworks.com.

Thanks

I'd like to thank my reviewers and editors, Mark Daniels, author of WASD, Jennifer Cole Ripman, my wife, and Ben Ripman, my son. Their contributions to this article made it significantly better than it would otherwise have been. Any errors remaining at this point are mine and not theirs.

For more information

Modularian – A server agnostic CGI that produces binary output.

Framework – A Perl script I used to convert a client's DCL OSU CGIs to simple server agnostic CGIs.

ServerAgnosticPerl.zip – sources of the listings for this article.

FastCGI Interface Specification – The FastCGI protocol and interface specification.

CGI Interface Specification – The Common Gatewate Interface specification.

PerlEx – A Perl performance enhancement for Windows.

WASD - A web server implemented using multiple processes.

OSU – A web server implemented using DECThreads.

Afterword

Perl makes, more or less, anything possible. The listings shown in this article present an "under the hood" view of server agnostic CGIs. It is possible to package these requirements and simplify the process enormously. As an exercise I have done so. Included in ServerAgnosticPerl.zip are two additional files, listing4.pl and saperlcgi.pm which demonstrate how well hidden the details of server agnostic CGI programming can be made in Perl.