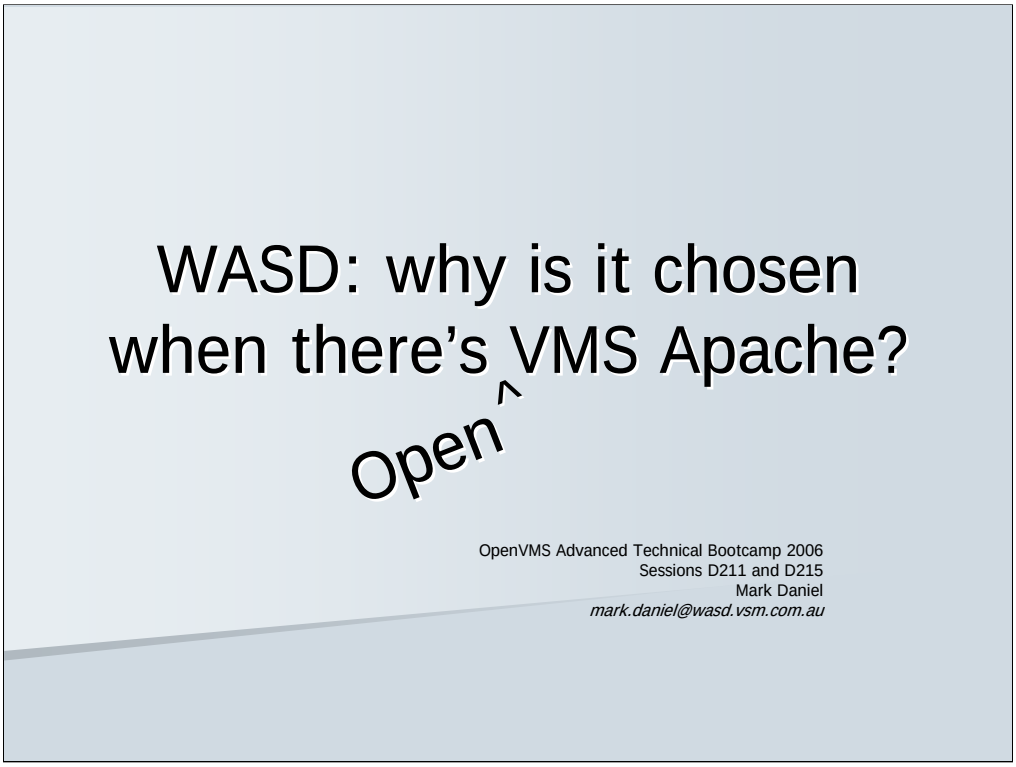
WASD: why is it chosen when there's VMS Apache?

OpenVMS Advanced Technical Bootcamp 2006
Sessions D211 and D215
Mark Daniel
mark.daniel@wasd.vsm.com.au

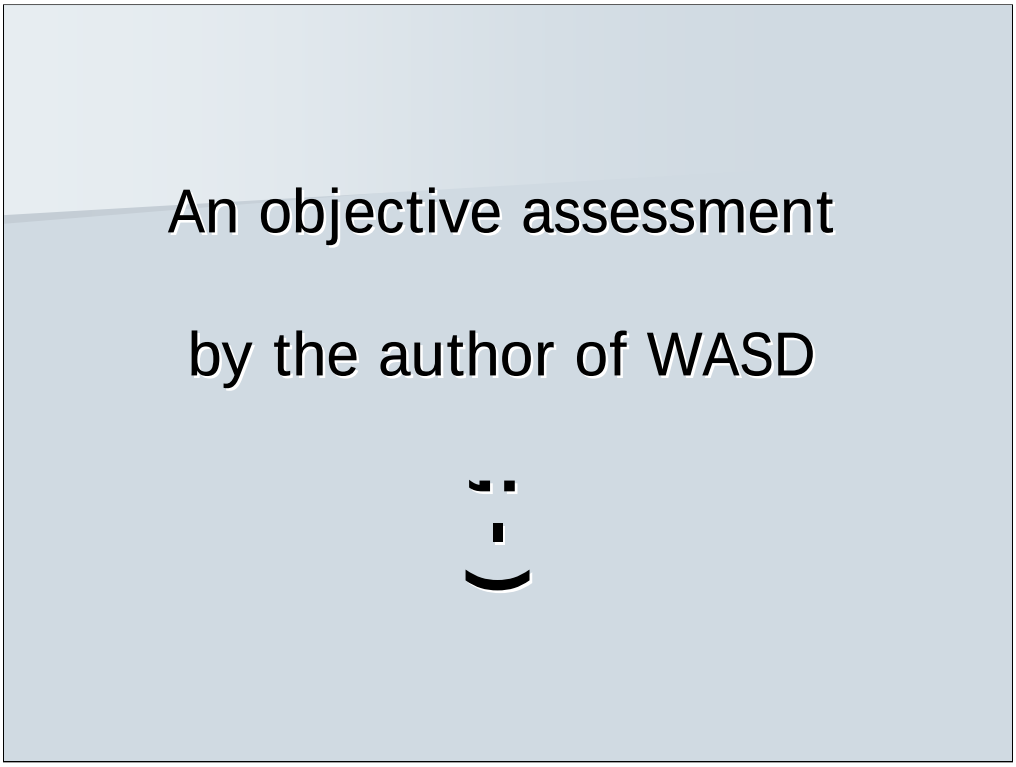


Hmmm. Session title is accepted using "VMS" then a missive arrives requesting the use of a written "Open" before "VMS" even if verbally it is made silent. Ok! I can live with that.

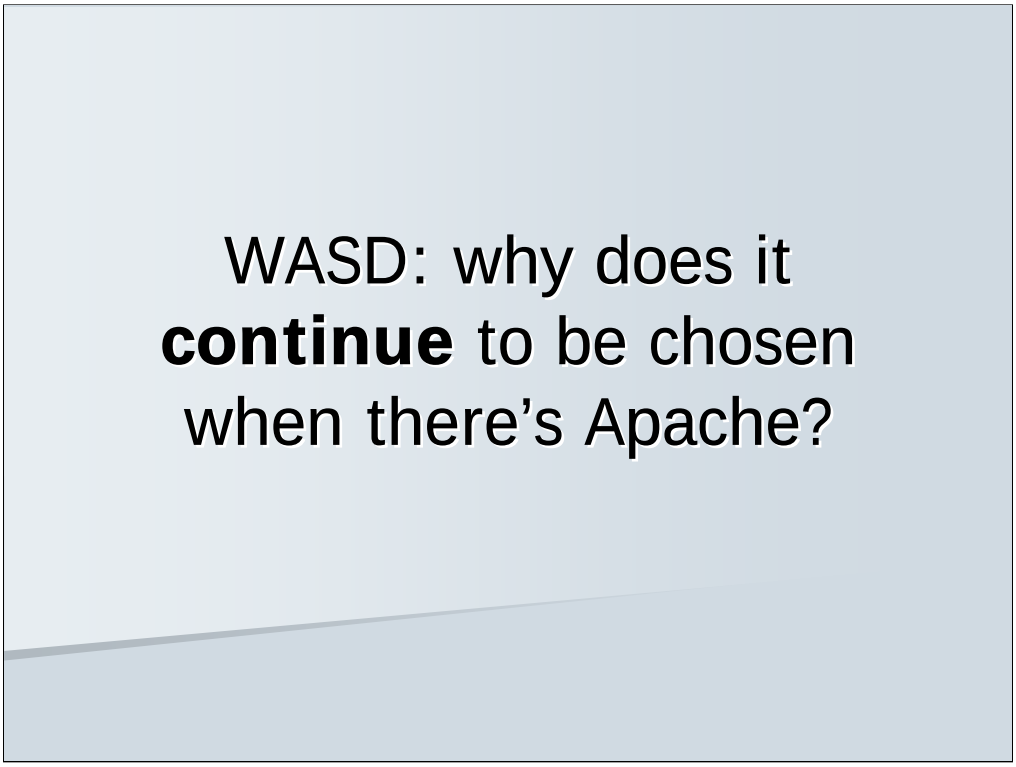
An objective assessment

An objective assessment

by the author of WASD



A bit like Bill Gates introducing the next generation of some Windows application (only he’s probably not got holes in his socks).



You can understand WASD being selected when OSU was the only other viable alternative but from the times when OpenVMS Apache 1.3 and 2.1 have been release why do sites continue to select WASD over this ported 'industry standard'?

Apache: why is it chosen when there's WASD?

Of course the converse question could be asked! With WASD being stable for a decade and OpenVMS Apache a bit of a moving target.

Ask me!! I have made successive modifications to the WASD CGILIB library that attempts to smooth out the differences between OpenVMS-based web servers catering for changes:

```
04-MAY-2005  MGD  v1.8.0, support SWS 2.0 along with CSWS 1.2/1.3,
                         CgiLib__ApacheSetSockOpt() for SWS 2.0 BG control
22-APR-2002  MGD  v1.6.3, CSWS APACHE_SHARED_SOCKET to APACHE$SHARED_SOCKET
08-JAN-2001  MGD  v1.5.2, CSWS V1.0-1 (VMS Apache) "fixbg" support (see note
                         above), APACHE_INPUT becomes APACHE$INPUT
09-APR-2000  MGD  v1.4.0, VMS Apache environment (1.3.9 BETA)
```

WASD behaviours, while receiving additional functionality over the last decade, have continued to transparently provide backward compatibility.

Rationale

- Why would any developer choose something not ‘industry standard’, with a smaller user base, fewer tools and seemingly more tenuous support?

The obvious riposte to a choice of something like WASD.

Rationale

Why would any developer choose something not ‘industry standard’, with a smaller user base, fewer tools and seemingly more tenuous support?

Same might be asked of OpenVMS!

Sometimes for very good (and familiar) reasons.

Rationale

- Purpose of the session is not (necessarily) to proselytize WASD but to explain how it might end up the preferred option for given projects and sites.

If attending the session or reading the notes inclines you to download and evaluate WASD all the better but it's not the primary objective.

The session is intended to familiarize you with those WASD capabilities that WASD users think are important differentiators for the product, in a structured presentation that attempts to tie the various elements together.

Thanks to Sponsors

Hewlett Packard

John Gillings and OpenVMS Engineering for the ‘admit one’

VSM Software Services

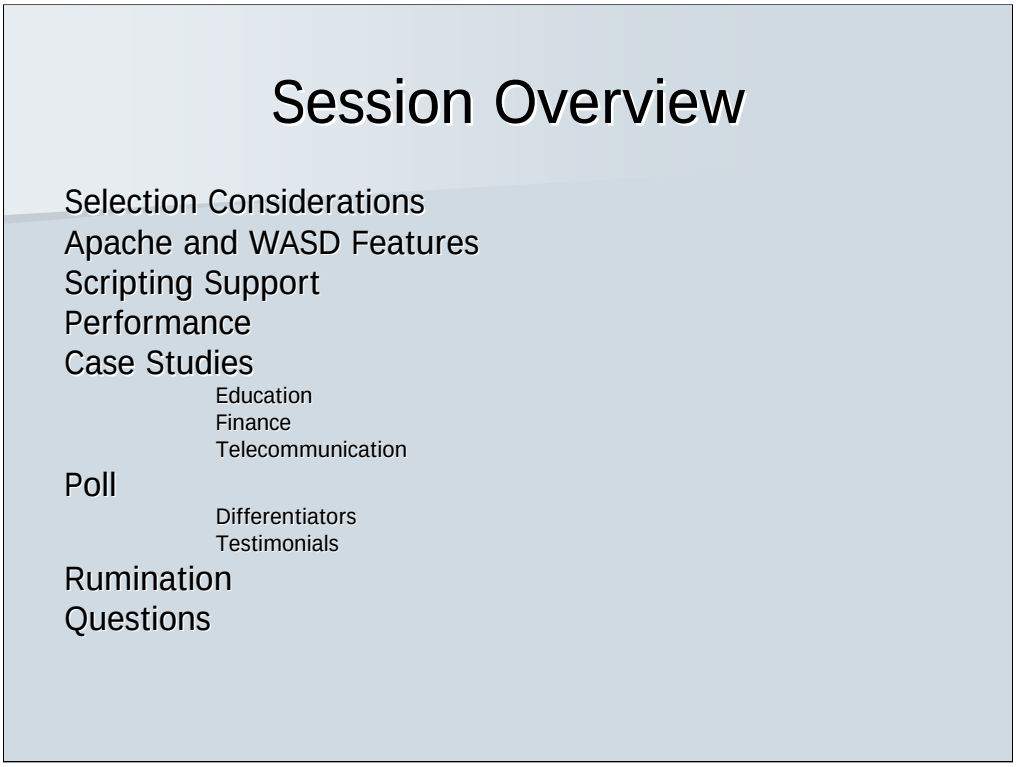
Jeremy Begg for not needing to swim here

Defence Science and Technology

Paul Amey for ‘board and lodging’

My wonderful spouse

Robyn for a long-*ish* leash and some pocket-money



There are four main sections to the session (now we have got this far).

1. Criteria that contribute to choosing a particular environment.
2. A review and comparison of Apache and WASD features.
3. An examination of four WASD sites.

   These have been chosen to represent something other the usual deployment of a web environment – for the publication of 'documents'.
4. A brief review of those differentiators and question-time.

It is assumed that the audience is familiar with web terminology (e.g. CGI, SSL) and OpenVMS technology (e.g. ACME, AST).

Testimonials and quotations are not attributed. It did this for 'privacy' reasons, not wishing to scatter contributors name and contact details around the Web. If you want the author detail for any particular case-study, comment or testimonial then please contact me directly at the email address on the title page and I will provide these privately for you.

This is just a quick mnemonic to the basic criteria involved when selecting any application, including Web-based ones.

Selection Considerations

- Purpose
  - Document publication
  - Data connectivity
  - Web ‘applications’
- Security
  - Authentication sources
  - Access control
  - Privacy (e.g. SSL)
- Content
  - Static
  - Dynamic
    - Scripting
    - *Pages (e.g. JSP, PHP)
- Scripting
  - CGI
  - Perl, PHP, Python, etc.

Selection Considerations

- Load
  - Peak
  - Average
- Platform
  - Alpha, Itanium, VAX
  - OpenVMS V*n.n*
  - Hobbyist, SOHO, Enterprise
- Other
  - Policy
  - 'Industry standard'
  - Documentation
  - Community
  - Contractual
  - Skills base
  - Comfort zone

Features

Apache and WASD Toe-to-Toe

Macro Comparison

	Apache	WASD
HTTP/1.1	Yes	Yes
Alpha/Itanium	Yes	Yes
Secure Sockets	Yes	Yes
IPv4 & IPv6	Yes	Yes
Persistent Scripting	Yes	Yes
Access Control	Yes	Yes
Request 'Rewrite'	Yes	Yes
Proxy	Yes	Yes
Logging	Yes	Yes
Perl, PHP, Python, etc.	Yes	Yes
License	GPL	GPL

Apache and WASD both offer all the features expected in a modern Web package. Of course the implementation and detail may differ but each is highly competitive in offering a 'full solution'.

Request 'rewrite', an Apache description for being able to modify request handling and characteristics during processing, is paralleled in WASD's conditional mapping and internal redirection facilities.

WASD offers full proxy serving (HTTP, SSL, CONNECT), with on-disk caching, protocol tunneling and protocol gatewaying.

Both WASD and Apache are available for free download and available for use under the GNU General Public License (GPL)

http://www.gnu.org/licenses/gpl.txt

Platform Support

	Apache	WASD
Alpha	Yes	Yes
Itanium	Yes	Yes
VAX	No	Yes
V6.0	No	Yes
V6.1	No	Yes
V6.2	No	Yes
V7.1	No	Yes
V7.2	V1.3	Yes
V7.3	V2.1	Yes
V8.2	V2.1	Yes
F8.3	V2.1(?)	Yes
Install ODS-2	V1.3	Yes
Install ODS-5	Mandatory V2.1	Yes

All of WASD features and facilities are available on all OpenVMS platforms for all OpenVMS versions from V6.0 onwards.

The latest release of Apache tends to rely on recent tweaks and accommodations to underlying run-time libraries, TCP/IP packages and even (so I'm led to believe) kernel tweaks.

This obviously lends some weight to WASD's selection where sites cannot or will not update hardware or O/S versions for whatever reasons.

Concurrent Serving

	Apache	WASD
Server	Child Processes	Single Process
Concurrency	Per-Process*	VMS AST
Multi-CPU	Per-Process	Multiple Instances**
Scripting	Per-Process or Subprocess	Detached Process

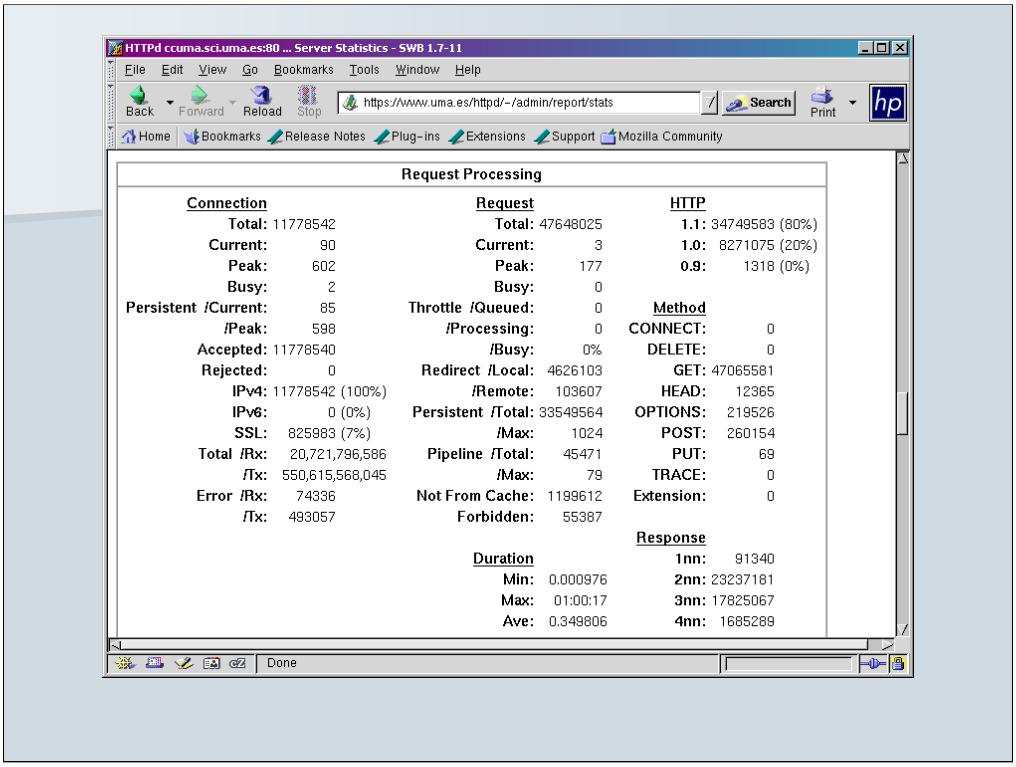
* To support 100 concurrent requests Apache requires a minimum of 101 processes.
** Multiple, per-CPU processes, cooperating via mutex and the DLM.

Apache, at least most of the current deployments, and certainly the OpenVMS release, uses a request-per-process model. A supervisory process creates child processes that are then managed through idle and request processing queues. Each child process handles a single request. Multiple child processes are used for request serving concurrency. This model is sometimes referred to as heavyweight-threading.

Experimentation by the author has demonstrated something like a 10-15% overhead in OpenVMS Apache process requirements under load. This presumably is managing a process into and out-of the idle and processing queues, and associated overheads, within Apache. Hence to support 100 concurrent requests Apache would require something like 115 instantiated processes (even to serve static pages).

WASD uses a single process and ASTs to enable an event-driven (mainly I/O but with some timer queue) multiple request concurrency. This model could be referred to as lightweight-threading. These are very lightweight in the sense they are VMS' native threading model, almost negligible servicing cost and certainly containing none of the thread-management overhead of something like POSIX Threads or a process context.

The WASD conservative approach to resource consumption in this respect is often a significant factor in preference over other approaches.



To sustain the 177 requests experienced by this site Apache (according to observations by the author described in the previous slide) would have required a minimum of approximately 200 Apache processes active on the system. This is the case even if all requests were for static pages, while in such a case WASD would have required just the single server process.

This image shows a portion of the *Server Statistics* report from the University of Malaga site described in the Case Study section. Statistics are accumulated in a permanent global section. A system startup or explicit action by the site administrator is required to reset these.

The data are reasonably self-explanatory and can just be browsed.

Items of possible interest:

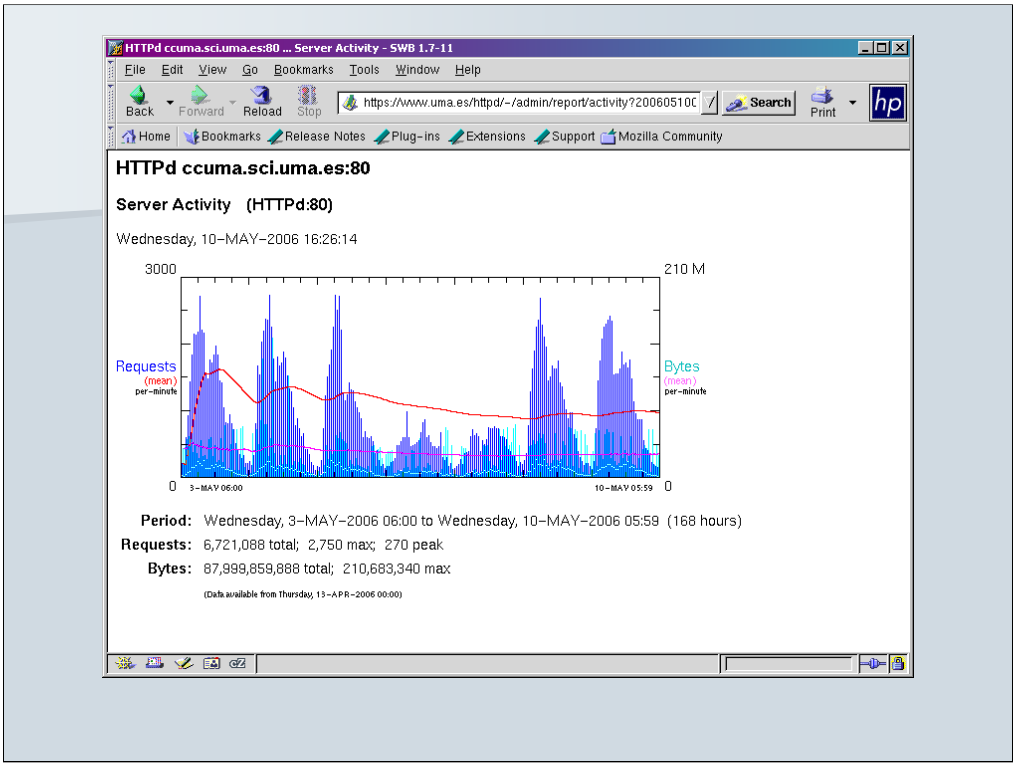
• Current network connections: 90 (these have largely persisted beyond the initial request, see Persistent/Current)

• Current requests in progress: 3 (actively being processed)

• HTTP/1.0: 20% (one fifth of all clients are still using this old protocol)

• Average duration: 349mS (on average a request is served in approximately one third of a second)

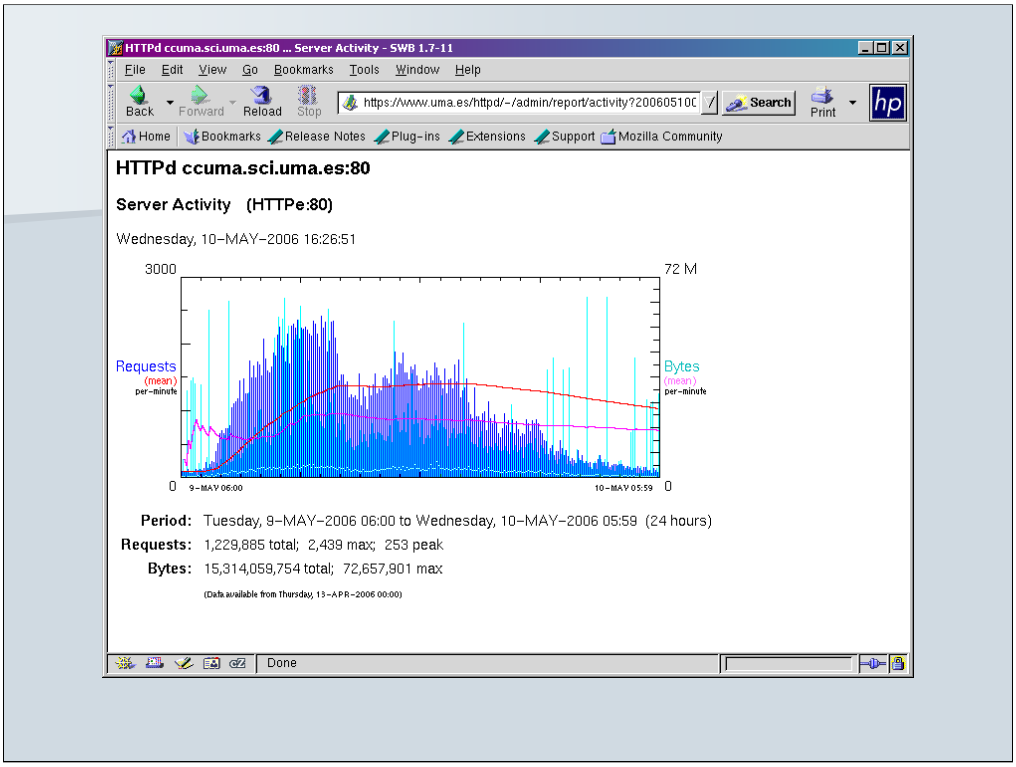
• WASD HTTP/1.1 ‘pipelines’ requests quite well when possible, with a maximum of 79 reported for any one request sequence.



This image shows the WASD *Server Activity* report and graph from the University of Malaga site described in the Case Study section. It is the busiest WASD site the author has access to.

Server activity statistics suitable for generating the graph are accumulated in a global section for a maximum of 28 days (this is a separate section to the server statistics of the previous slide). This graph shows a 7 day period. Weekdays and the weekend are quite obvious. Totals represent a per-minute accumulation.

During the week represented by the graph approximate 88 Gbytes have been transferred as a result of 6.7 million requests. The maximum transferred in any one minute was 211M bytes, or approximately 3.5M bytes per second (28Mbps). The maximum number of requests (this would be better labeled connections) handled in any one minute was 2750, or approximately 44 per second, with a peak of 270 connections concurrently processed.



Activity graph showing a single twenty-four hour weekday period.

I can imagine I see the southern-European traditional period of siesta in the mid-afternoon (and thoroughly civilized tradition I might add ☺).

Authentication

	Apache	WASD
Package*	Yes	Yes
SYSUAF	Module	Yes
PKI**	Module	Yes
Custom***	Yes	Yes

* package-specific username/password
** Public Key Infrastructure (X.509, etc.)
*** User-written authentication support

WASD natively supports credentials from a significant number of sources:

- the SYSUAF
- with or without VMS rights identifier possession
- the ACME services
- plain-text lists
- WASD-specific binary authentication databases
- X.509 client certificate
- RFC1413 (Identification Protocol)
- authorization agent (user-written CGIplus-based authenticator)

Scripting Support

	Apache	WASD
CGI	Just*	Yes
Perl	Module	RTE**
PHP	Module	RTE
Python	Module	RTE
Tomcat	Module	Reverse Proxy
Persistence	Yes	Yes

* Implied criticism of OpenVMS Apache performance
** RTE is a persistent Run-Time Environment

WASD supports all the major dynamic content environments available to OpenVMS Apache.

CGI performance under OpenVMS Apache is abysmal. There may be good reasons for this but a glance at the CGI performance data in later slides lends itself to 'abysmal' not being too wild a description.

Both support 'persistence'. This is a scripting characteristic rather than environment that lends itself to great efficiencies. See the following slides for more detail.

Persistent Scripting

So what is ‘persistence’ then?

The ability of the server to reuse resources (such as processes) over multiple requests

A scripting/interpretation engine retaining it’s initialized state over multiple requests

It is generally recognized that the maintenance of application multi-user concurrency through the use of discrete processes (sometimes referred to as heavyweight-threading) and the required O/S process context switching, is an expensive approach. Hence part of the impetus behind the more lightweight mechanisms such as POSIX Threads where the concurrency is maintained within a single process context.

Even in the Unix family of O/S, where the process is considered an expendable resource, it is recognized that the creation of processes to respond to single, particularly if short-lived, application demand, is a method that does not scale effectively in time or resource consumption.

With OpenVMS the process is considered much more of a permanent resource and is very much more expensive to create. Any ported application that relies on the cost and behaviour of Unix style processes will not scale appropriately under OpenVMS.

CGI scripting relies on creating a child process to service a single request. In keeping with the above observations it does not scale particularly well.

In addition, if the script relies on an interpreter or other engine that requires initialization, that will add further latency and CPU cycles.

Persistent Scripting

Why is ‘persistence’ so important?

Process activation expenses

- Latency
- CPU cycles

Scripting engine initialization

- Latency
- CPU cycles

Obviously if these activation expenses can be amortized over more then one usage then the cost per instance is reduced proportionally.

If the scripting resource can be made persistent and multiple successive usage managed the activation overhead may be absorbed almost completely.

Persistent Scripting

CGI paradigm is very expensive; solutions:

Apache
- child processes
- loadable modules

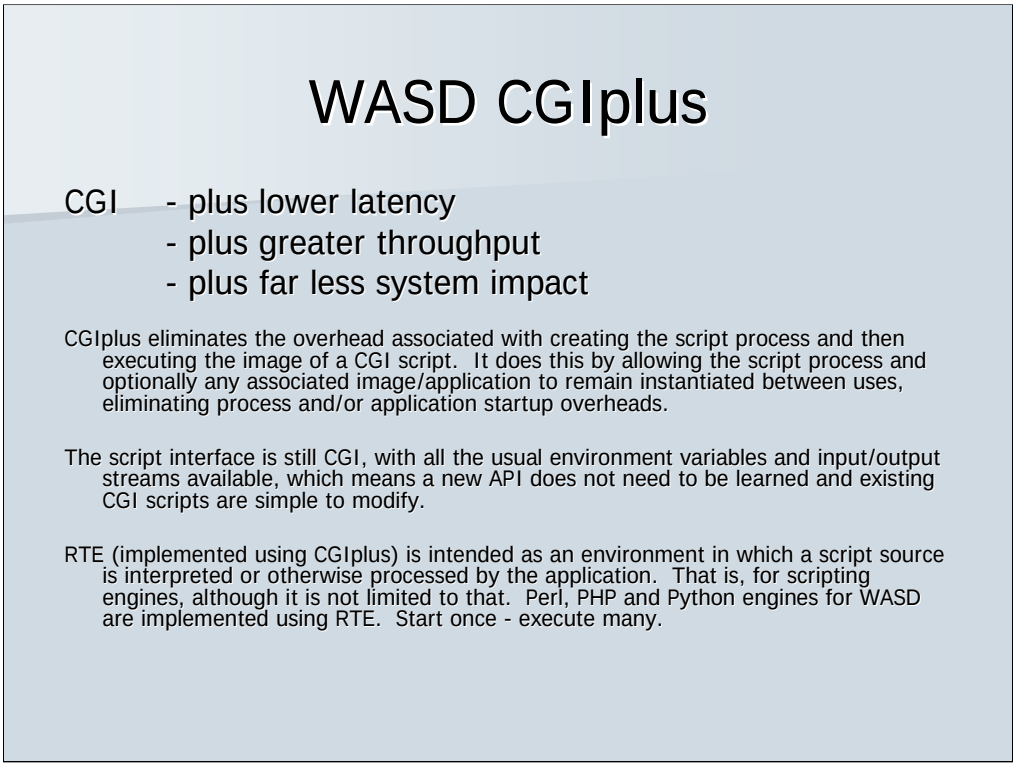
WASD
- reusable detached processes
- CGIplus
- Run Time Environment (RTE)

The original approach to providing dynamic content not part of the underlying Web server application was to run a script or executable in a child process and deliver the output from that back to the client. This was (somewhat) formalized in the CGI protocol and de facto standard.

To reduce the expense of this approach all server environments have needed to accommodate resource persistence for commonly used dynamic content environments – to make dynamic content using child processes the exception rather than the rule.

Apache manages request concurrency using multiple child processes. Into these it dynamically loads executable code ('dynamic shared objects', i.e. sharable images) for configured dynamic resource providers. Common examples are Perl, PHP and Python, but can be anything, including user-written modules. Each handles multiple, consecutive requests, activating the appropriate code as required. CGI is still a process-per-request paradigm.

WASD, faced with the same issues and a significantly more costly process creation, required similar resource persistence. OpenVMS processes lend themselves to reuse and so WASD goes one step further and generally maintains the process between CGI usage. It also introduced a small variant on CGI that allows scripts and/or scripting engines themselves to persist over multiple requests. These are known as CGIplus and Run-Time Environments (RTE).



CGIplus operates at two levels.

First, it manages reuse of processes.

> It creates detached processes on demand, distributes them to an appropriate request, activates the appropriate scripting application, transfers data to and from the script and the client, manages the process while idle and finally deletes a process when appropriate time periods expire. One process, many CGI activations. This is completely transparent to the CGI script itself.

Secondly, it allows a scripting application to remain resident.

> It creates a new or allocates an idle process and activates the scripting application in that. That application then remains resident and prepared to accept requests with very low latency. It loops from an idle state to active, accepting the CGI data from the server, performing required processing and output, indicating end-of-response, and returning to an idle state. It's still CGI though, with some minor accommodations for persistence. It's not uncommon for a CGIplus application to be ten times less latent than the same application executed as standard CGI.

An RTE is essentially a remain-resident CGIplus application described above that is an interpreter for other script sources supplied to it.

Performance

The test system was a lightly-loaded AlphaServer 4100 4/400 (4 x 400MHz CPUs), OpenVMS V7.3-2 and DEC TCP/IP 5.4. No keep-alive functionality was employed so each request required a complete TCP/IP connection and disposal. DNS (name resolution) and access logging were disabled. The server and test-bench utility (ApacheBench v1.3) were located on separate systems with 100 Mbps Fast-Ethernet interconnection.

On clustered, multi-user systems too many things vary slightly all the time. Hence the batching of accesses, interleaved between servers, attempting to provide a representative result.

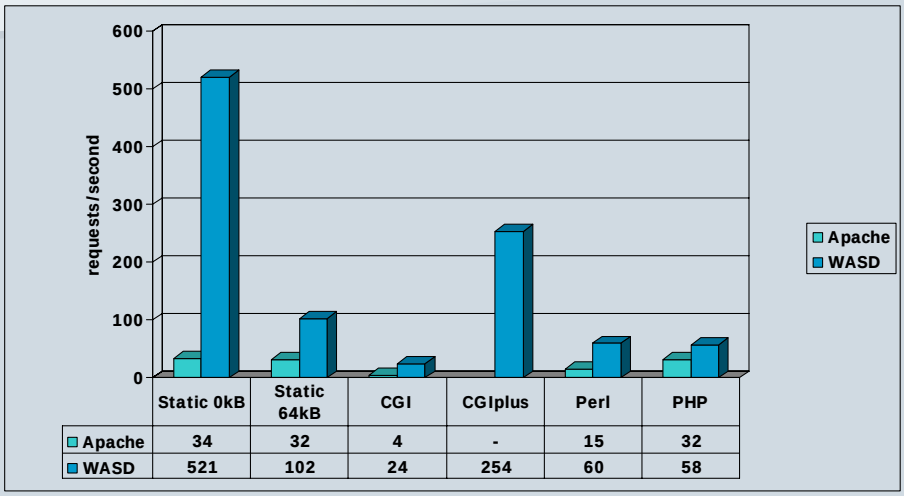
CSWS 1.3 (based on Apache 1.3.26)

WASD 9.0

Source: http://wasd.vsm.com.au/ht_root/doc/htd_2100.html

Indicative only - but instructive all the same.

Performance – 1 concurrent



	Static 0kB	Static 64kB	CGI	CGIplus	Perl	PHP
Apache	34	32	4	-	15	32
WASD	521	102	24	254	60	58

source: http://wasd.vsm.com.au/ht_root/doc/htd_2100.html

Static 0kB: File comprising zero bytes of content. Included to measure the throughput of the entire network connection establishment, request acceptance, resources access, response delivery, connection dissolution – without the overhead of actually delivering any resource content.

Static 64kB: File containing sixty-five kilobytes of content. Same measurement as above with the addition of multiple PDUs to the client.

CGI: An executable that generates a CGI response header and variable quantities of response content (according to activation parameters).

CGIplus: Same executable as CGI but persists between usages removing activation latencies. Included here for comparison with WASD CGI performance and shows approximately ten times greater throughput!

Perl: Persistent interpreter under both platforms running the following script

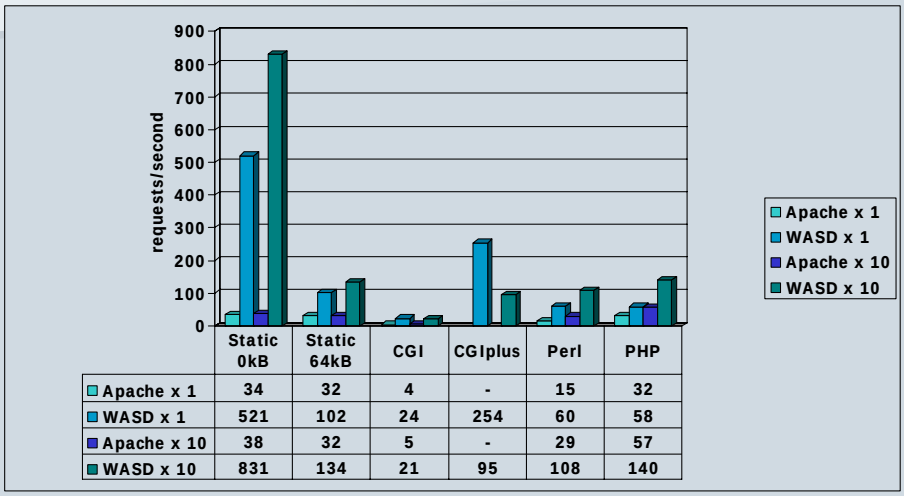
```
print "Content-Type: text/html\n\n
<B>Hello!</B>";
```

PHP: Persistent interpreter under both platforms running the following script

```
<?
php echo "<B>Hello!</B>“
?>
```

More information: http://wasd.vsm.com.au/ht_root/doc/htd/htd_2100.html

Performance – 10 concurrent



	Static 0kB	Static 64kB	CGI	CGIplus	Perl	PHP
Apache x 1	34	32	4	-	15	32
WASD x 1	521	102	24	254	60	58
Apache x 10	38	32	5	-	29	57
WASD x 10	831	134	21	95	108	140

source: http://wasd.vsm.com.au/ht_root/doc/htd_2100.html

The same performance results as described in the previous slide but this time with results from ten concurrent requests added for comparison.

Obviously what works well under light load might behave differently and a heavier one.

Case Studies

OpenVMS+WASD success stories

Case studies basically comprise three slides

1. Overview of site
2. Significant perceived advantages
3. Success stories always seem to have notable quotes ☺

Of course there may be much more to an individual site decision than presented here.

Case Study - Education

Universidad de Málaga - Spain

- 4 campuses; 19 faculties; 65 undergraduate courses; 3760 staff; 40,000 students
- A significant user of OpenVMS for email, Web, database and administration
- Planning migration from OSU to Apache in late 2002 evaluation revealed show-stopping issue with Apache

"A second threat for [SSL certificate] key disclosure exists during script execution because scripts run in the context of the server and have complete access to key files no matter where they exist (as long as they exist in a directory accessible to APACHE$WWW). Therefore, it is not advisable to allow the execution of arbitrary user scripts when using SSL." *OpenVMS Apache Release Notes*

University of Malaga, Spain (UMA).

As executing scripts under multiple accounts and having script authors not directly responsible to the web service administration made the chance of inadvertent or malicious SSL certificate exposure unacceptable. Subsequent to the deployment of WASD selected students also have been permitted to provide active content (mainly via PHP) further exacerbating this issue.

Case Study - Education

Universidad de Málaga - Spain

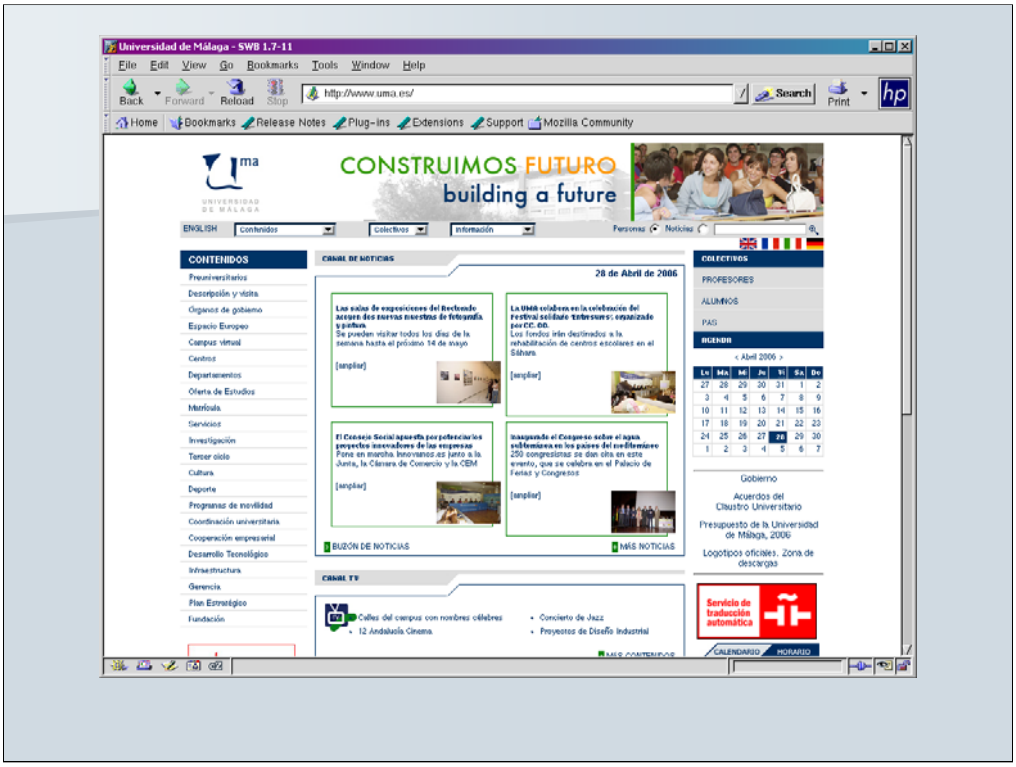
- Evaluated WASD in early 2003 and put it into production shortly after!
- 76 virtual servers
- >1M requests and >15GB per weekday
- >600 concurrent connections and >100 requests in-progress routinely supported (using 2-30 processes)
- X.509 based PKI authentication access control
- Extensive deployment using PHP, along with existing OSU scripts, and more recent CGI based applications

WASD supports a number of virtual servers limited only by practicality and available resources. The number of servers offered within UMA has grown significantly since moving to WASD.

It's not at all atypical for the site to process in excess of 6 million requests and 80 Gbytes of data over an academic week (see *Server Activity* graph taken from this site in a previous slide). These quantities can double during periods of student registration or result publication.

UMA is the most loaded site I have administrative access to and is included here to illustrate, amongst other things, that WASD can handle significant loading elegantly and efficiently.

WASD provides an OSU scripting emulation and allowed UMA to continue to use it's extensive suite of applications developed during the period of OSU deployment (I dislike the term 'legacy').



University of Malaga makes very extensive use of PHP to provide both it's content navigation system and the dynamic, interactive components of it's Web services. Even with much of it's content dynamically generated by this interpreted markup language average request response time still remains about one third of a second.

The WASD PHP engine uses the CPQ AXPVMS CSWS_PHP V1.2 product shareable image from a persistent RTE. This means the PHP engine only needs to be initialized the once before handling multiple script requests - sometimes thousands before becoming idle long enough for the server to consider removing it from the system.

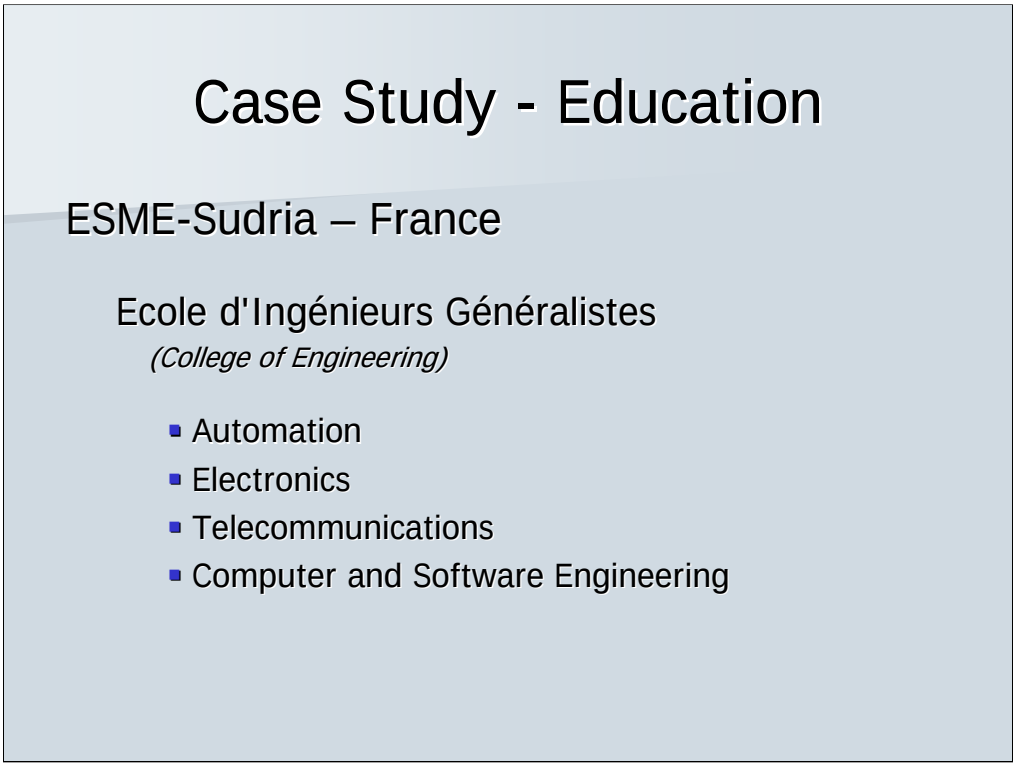
Checking the UMA system (it is the now weekend and late on a Saturday in Spain) I find eleven of the instantiated PHP engines on the system running under five different accounts. The number of scripts each one of these has been activated to interpret are 6894, 6731, 6953, 3697, 4831, 4254, 2440, 2277, 2006, 1349, 276. I have seen counts in excess of 25000. Obviously neither the OpenVMS Engineering PHP sharable image nor the WASD PHP interface leaks too many resources ☺

Case Study - Education

Universidad de Málaga - Spain

"WASD has allowed us to build a very robust, and above all, secure, web infrastructure, without having to give up twenty years of VMS knowledge. For us, the strongest points of WASD are excellent performance, excellent VMS security model integration and unbeatable support."

site: http://www.uma.es/



ESME-Sudria School of Engineering – Paris, France

This case study was a late addition to the session.

By kind invitation I visited the school in Paris on my way through to the 2006 OpenVMS Advanced Technical Bootcamp (yep, this one!) Although I have had a lot of interaction with the CIO at ESME-Sudria over the last five years (many WASD bugfixes and enhancements are due to the efforts of Jean-Pierre Petit) it took the visit and a formal presentation on the uses this engineering school has put WASD to for me to fully appreciate the clever and lateral thinking that has gone here to produce elegant, simple and powerful functionality.

After the visit and presentation I considered the site a must-add for the Bootcamp session. Certainly an OpenVMS+WASD success story!

Case Study - Education

ESME-Sudria – France

- Internet
- Intranet
- Standard proxy
- Reverse proxy
- Gatewaying

With thanks to Jean-Pierre Petit I include a number of slides lifted directly from his presentation and use them to illustrate just one aspect of their WASD usage – proxying in it's various forms. These slides are in the original French language (which I think just adds a certain exotique to we mainly English language audience ☺).

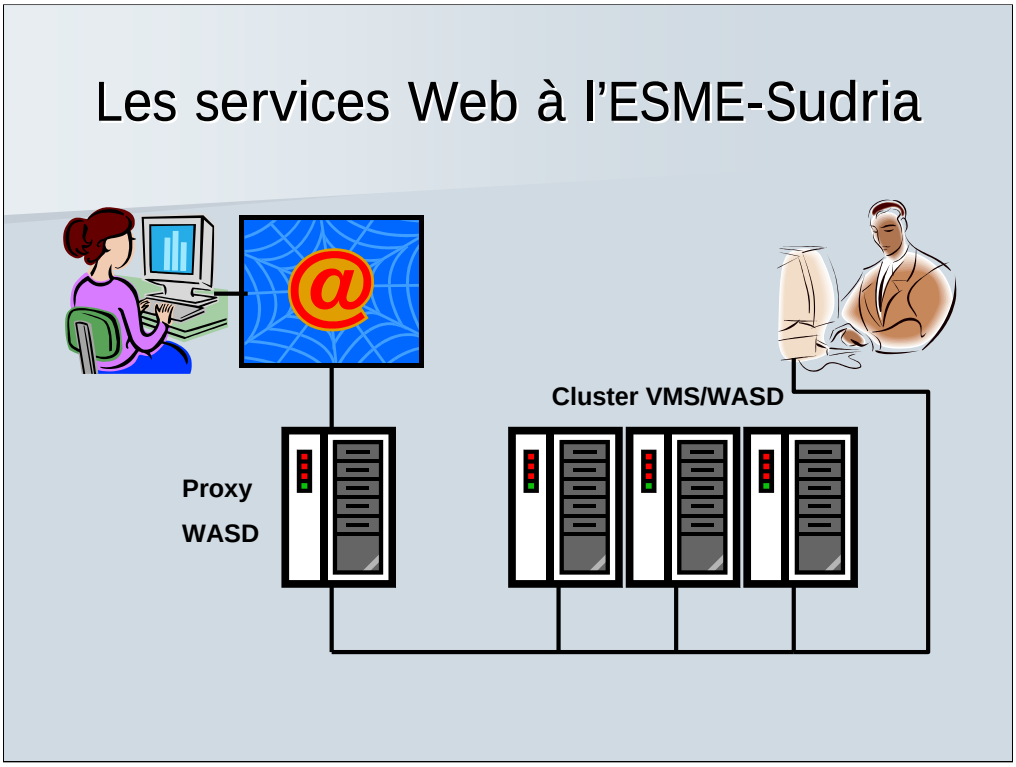
The full Powerpoint presentation in the original French as well as an English language version should be available (perhaps soon) from the WASD site:

http://wasd.vsm.com.au/other/

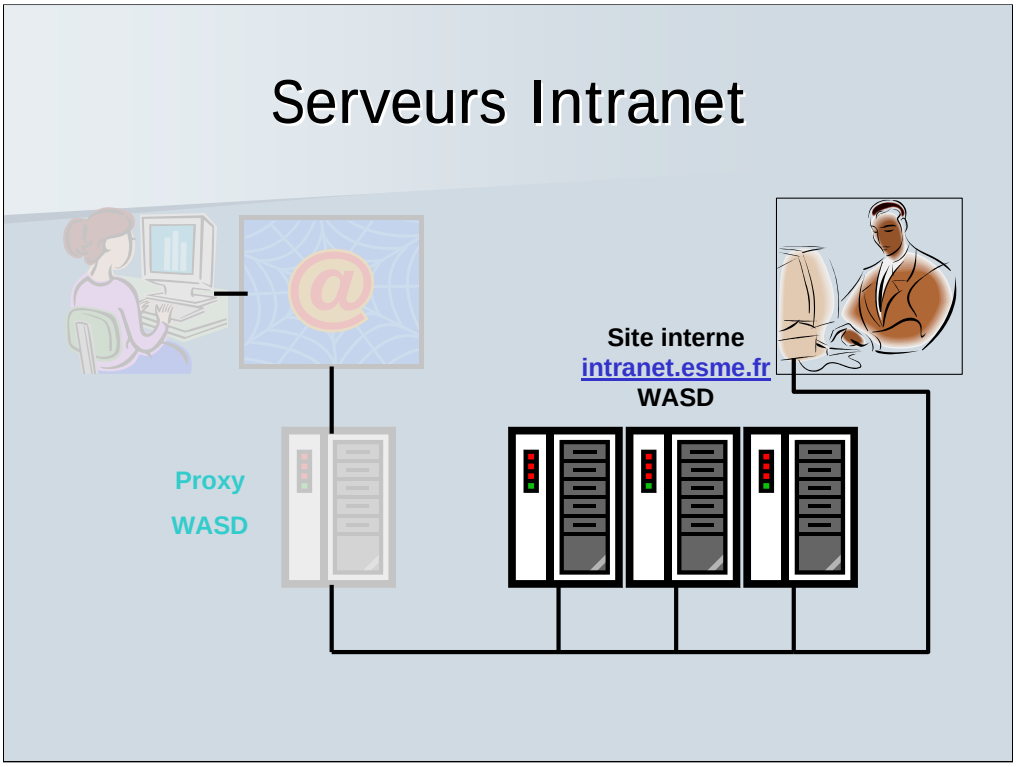
Stay tuned! I think I have convinced Jean-Pierre to find the time in an already crowded schedule to submit an OpenVMS Technical Journal article on the Web services provided at ESME-Sudria.

The following slides concentrate on the proxying aspects of WASD at ESME-Sudria in the context of other general and intranet services.

Any errors or omissions are entirely mine.



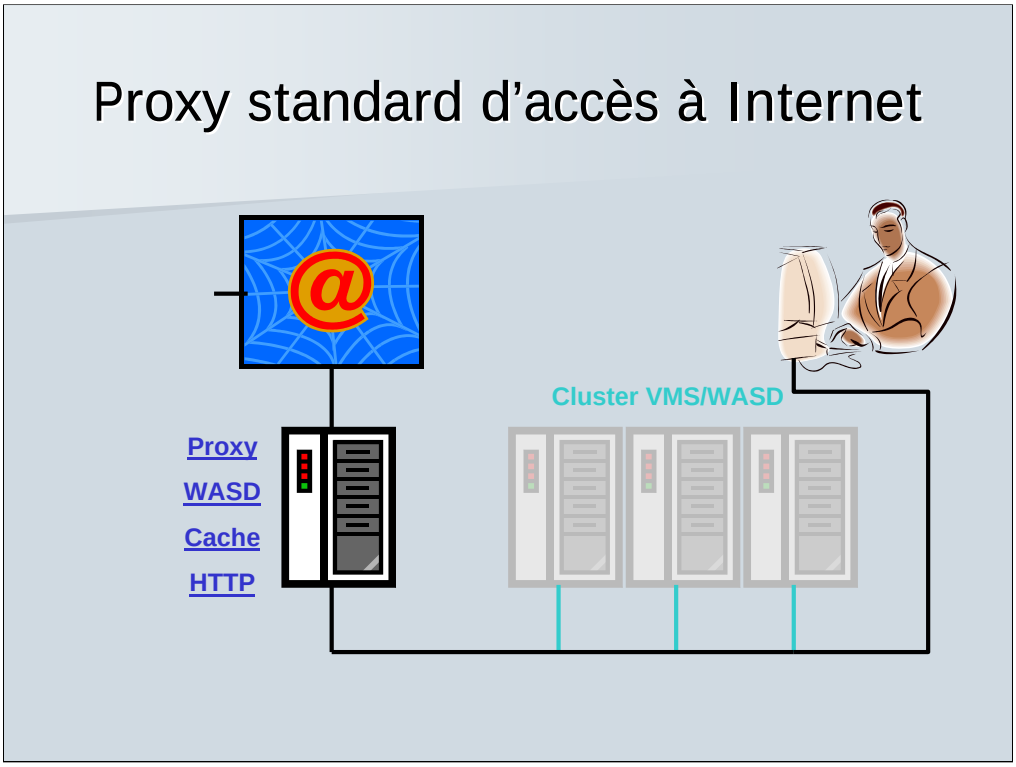
ESME-Sudria protects its' internal resources via firewall and WASD proxy.
Much of it's content and resources are hosted on an internal OpenVMS cluster.
External and internal access is appropriately provided to these resources.



Many internal services are provided to the student population not only from their desktop on campus but also at home (see later).

Also, and most interestingly, from a series of touch screens located around he campus, are available daily class timetables, room allocations, other timetabling information, student photographs, and other services.

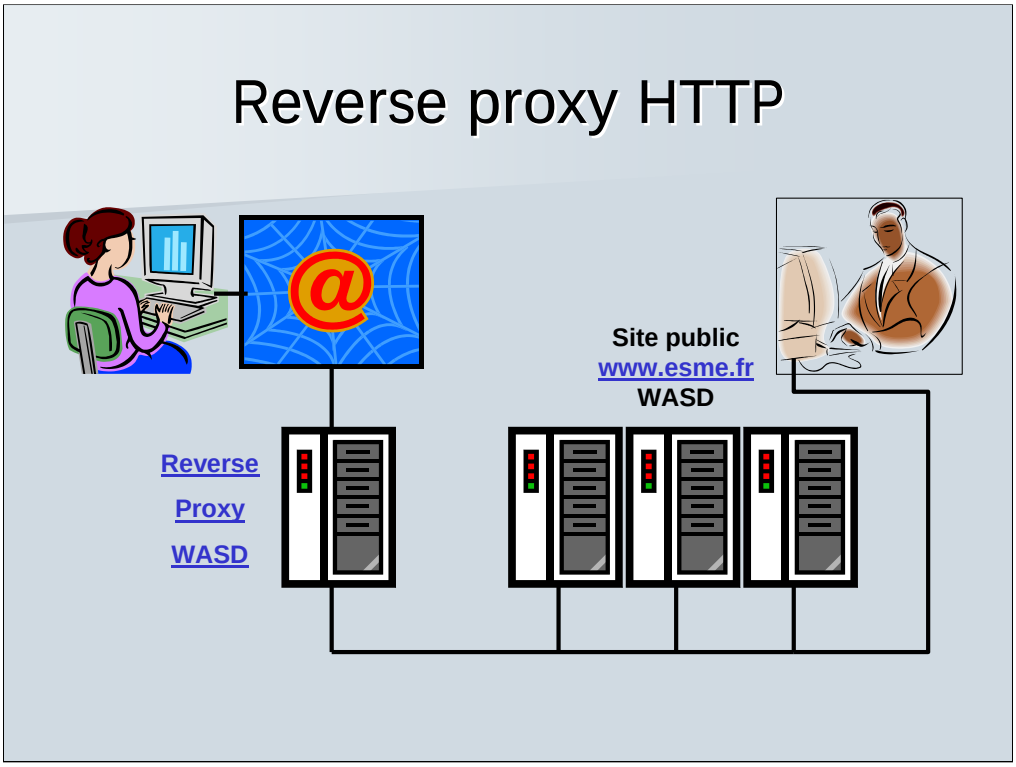
VMS and WASD feature as technical content in a number of courses, either as that or supporting application development using a number of environments including Python. WASD persona scripting allows such Web applications to be developed and run under a student's own account.



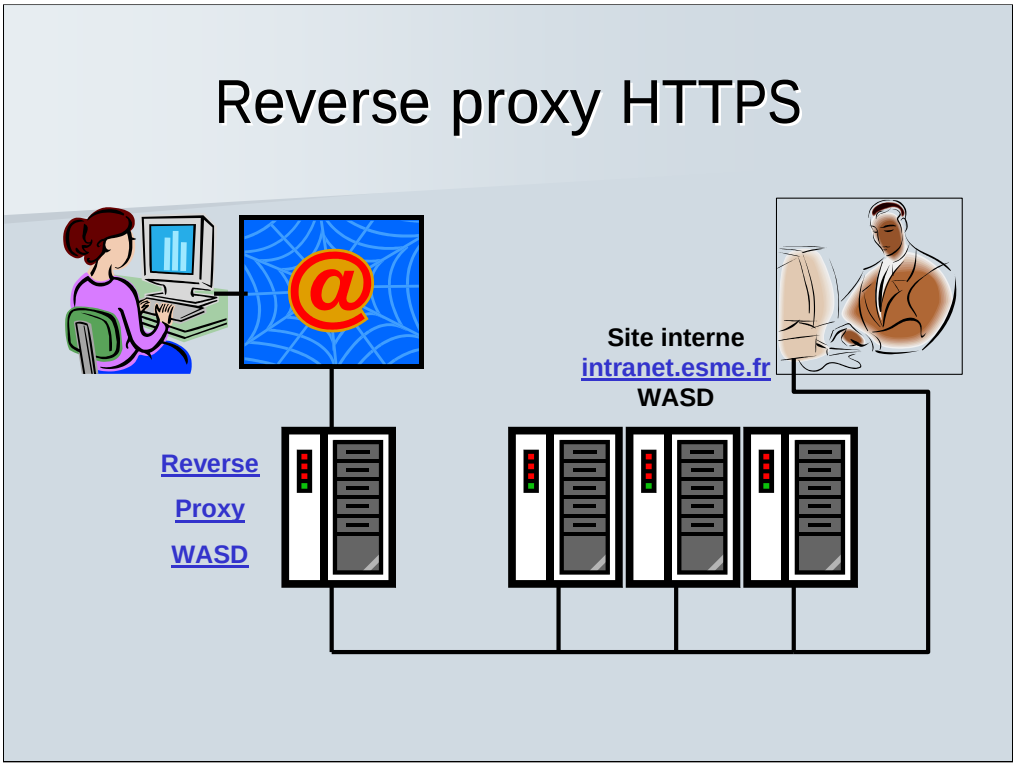
ESME-Sudria makes extensive use of WASD standard proxy and proxy disk-caching to improve responsiveness and reduce bandwidth consumption when accessing resources external to the school.

Proxy caching on disk is supported on a RAID 0 set to improve proxy disk I/O performance.

WASD request processing configuration (i.e. rewrite) is used to dynamically suppress advertisement and other unwanted material during external access.

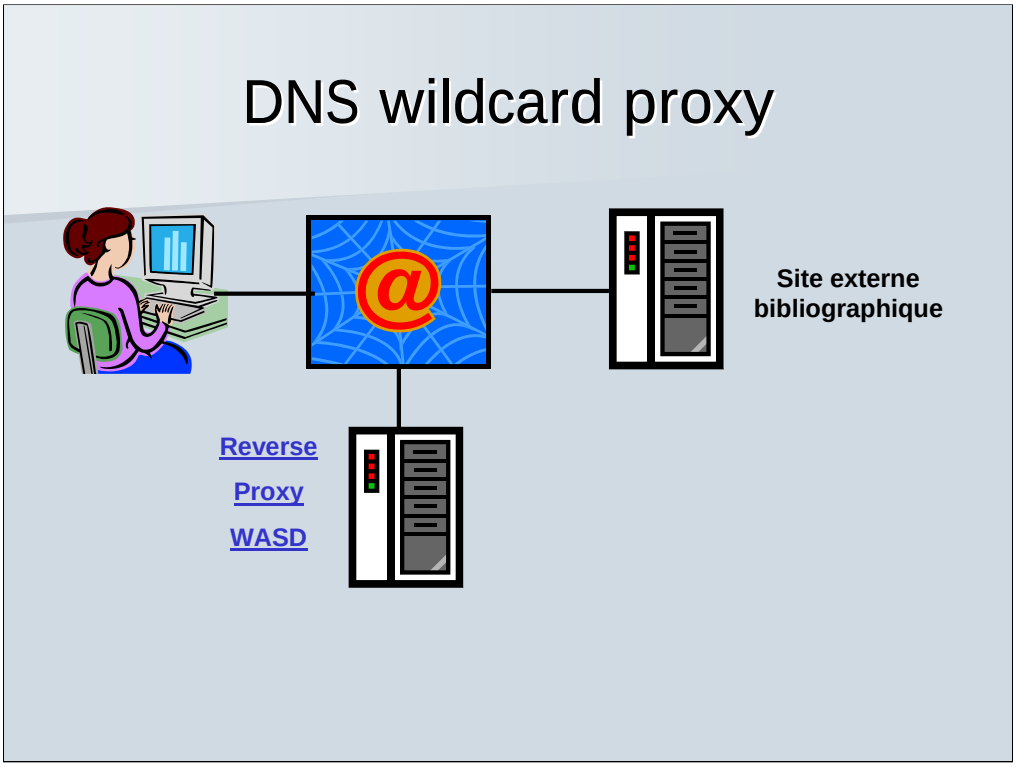


The resources of the public site are hosted on the internal cluster and made accessable via the proxy system using reverse-proxy (external to internal). This offers a measure of protection to the internal resources and allows failover and load sharing to be performed on the internal systems.



The ability of WASD proxy to gateway between Web protocols is used to provide secure access to some internal resources. The external client can connect to the WASD proxy using SSL which can then use standard HTTP or FTP to access the resource across the internal network. It can also proxy SSL to SSL (this is different to CONNECT proxy support).

Resources hosted on internal systems, including the primary VMS cluster, as well as other non-VMS systems (e.g. IIS) are accessed via WASD reverse proxy. This provides a measure of protection and even access control, with authentication and access control to these internal services being provided via WASD from SYSUAF credentials resident on internal systems.



An innovative application of WASD proxy allows students off-campus to continue to have access to external research materials (e.g. on-line library and other research facilities) that require access from the ESME-Sudria network (i.e. from an ESME-Sudria IP address).

A client external to the campus requests from the proxy server system (but not as a proxy request – which I'll grant you sounds a little confusing at first) a certain URI. This URI is recognised by WASD as a request for one of these external sites and it transparently accesses the resource from the remote site (using the ESME-Sudria host as the source) and returns it to the external client.

Case Study - Education

ESME-Sudria – France

"WASD has enabled us to webify more and more applications and develop brand new ones with excellent performances. CGI+ has provided us with applications that responds in a tenth of a second ... A lot of features are in use at ESME-Sudria: web servers, proxy, reverse proxy, DNS wildcards proxy... Even some IIS server are protected by authorization through a WASD reverse proxy, giving to VMS the ability to allow single sign-on to different platforms."

site: http://www.esme.fr/

Case Study - Finance

Coast Capital Savings - Canada

- Coast Capital Savings is a credit union servicing 300,000 customers in the Lower Mainland and southern Vancouver Island regions of British Columbia, Canada. Coast Capital Savings banking system runs on OpenVMS AlphaServers and is written in Greystone Technology M (M).
- WASD is principally used as an application server (middleware) for integrating traditional 'green-screen' financial database application with Windows-based (.NET) applications.
- The XML-SOAP-RPC mechanism implemented for this serves approximately 1500 interactive workstations, as well as a busy customer-facing IVR system.

For significant detail on this case study see the OpenVMS Technical Journal V7 article "WASD in SOAP/XML Transaction-Oriented Environments

http://h71000.www7.hp.com/openvms/journal/v7/wasd.html

Case Study - Finance

Coast Capital Savings - Canada

- Ease of integration
  - CGI or CGIplus programming in DCL, Python, MUMPS, C, etc.
- VMS security mechanisms
  - persona scripting and particular account contexts
- Performance
  - persistent CGIplus provides a low-latency (few milliseconds) high throughput transaction infrastructure
- Application management
  - load-balancing, throttling, script process rundown allowing 'gentle' application/server shutdown and/or system migration

Case Study - Finance

Coast Capital Savings - Canada

"WASD came bundled with a friendly gentleman in Australia who appears to be online 24x7 … also appears to read all the latest specs, to do tons of testing, and keep pushing WASD forward … makes WASD worth it's weight in platinum."

more information: http://h71000.www7.hp.com/openvms/journal/v7/wasd.html

Case Study - Telecommunications

EDS Telco Solutions Group - Australia

- Developed by EDS on behalf of an Australian telecommunications carrier providing landline, cell phone and Internet services. Due to Commercial-in-Confidence considerations, the customer cannot be identified.
- Service Profile data includes billing, product information, discounts, promotions, and mobile features information.
- To permit existing corporate systems and middleware to exchange Service Profile information, web services technologies based on XML, SOAP 1.1, and HTTP were employed. These enable the exchange of XML encapsulated information to and from retailers, OpenVMS applications and the GSM network hardware.

For significant detail on this case study see the OpenVMS Technical Journal V7 article “WASD in SOAP/XML Transaction-Oriented Environments”

http://h71000.www7.hp.com/openvms/journal/v7/wasd.html

Case Study - Telecommunications

EDS Telco Solutions Group - Australia

- Available for VAX platform
  - some remaining systems required consideration
- CGIplus persistent scripting
  - eliminate per-request process creation on busy systems
  - allow database context(s) to remain instantiated
- Script process termination
  - WASD issues $FORCEX before shutting-down idle scripts
  - allows exit handlers to elegantly release database context(s)
- Monitoring and troubleshooting
  - server statistics, WATCH facility, WOTSUP utility

Case Study - Telecommunications

EDS Telco Solutions Group - Australia

"Truth be known, I put my choice behind Apache initially due to the number of developers out there ... then I found out that WASD was developed specifically for VMS ... an OpenVMS solution. I'm glad my decision on choosing Apache was not adhered to because WASD has proved a very good choice indeed. WASD ... is cluster-aware ... synergic with the OpenVMS OS's philosophy and design. WASD developer(s) and community are helpful and very responsive.

Because it simply kicks-arse!"

more information: http://h71000.www7.hp.com/openvms/journal/v7/wasd.html

Translation into American English: "kicks-ass".

info-WASD Poll

These are lists distilled from respondent comments to the mailing list poll where some particular WASD attribute was of particular significance in the package preference.

Notes derived from the poll of the info-WASD mailing list on 25th Feb 2006:

"As you can see from the abstract the intent of the session is to highlight those characteristics of WASD that result in the choice of it over alternatives (though it is intended to be in comparison to Apache the same reasons might apply to other platforms). I am not wanting to script your responses but the reasons can be as simple and obvious as 'I need to run it on VAX'/'on VMS V7.1', or for complex technical reasons ('VMS Apache's security model is broken' springs to mind :-), stability, performance, scripting, 'philosophical', etc.

Over the years I have had enough contact with many sites to be able to reconstruct the session based on comments made in this forum and privately. Of course the environment for these sorts of Web services has changed enormously over the last decade and significantly in the last few years with the advent of Apache on VMS so I would like to present a fresh, accurate and up-to-date perspective. I would appreciate (all) your input and comments you might consider relevant to the topic, either via info-WASD or privately (just make sure you put 'bootcamp' in the subject line so I can find you easily in the SPAM quarantine).

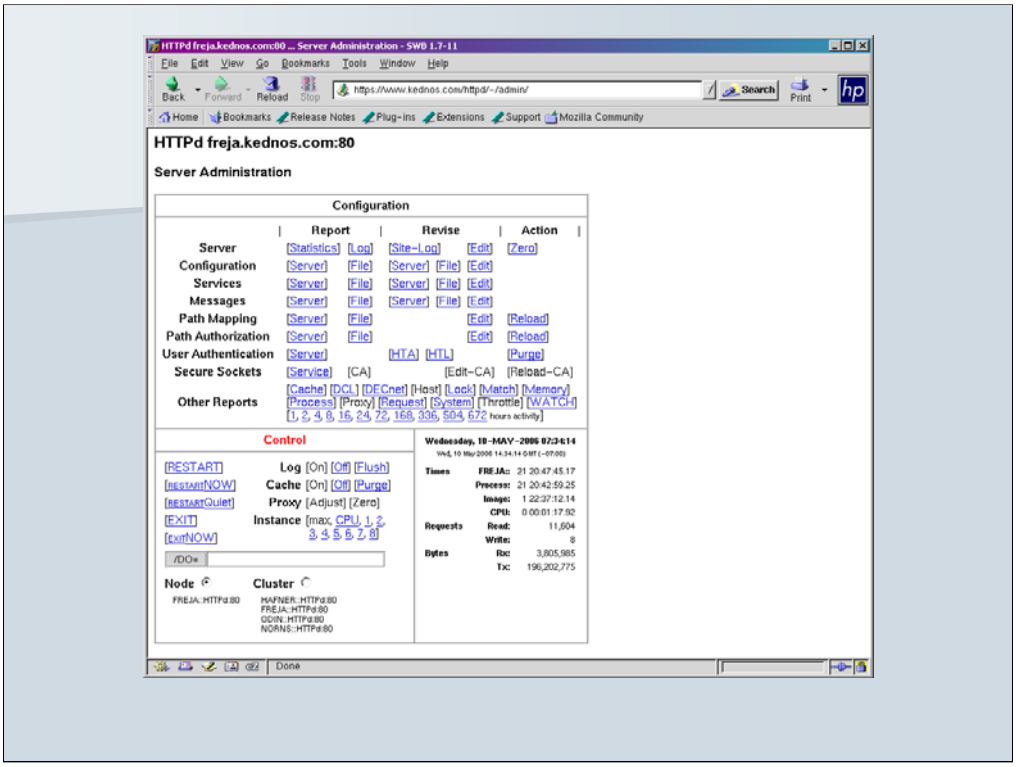
I will collate these and draw the session from them. Common themes will be discussed with explanations of any underlying technical detail (e.g. persistent scripting). Significant sites with specific requirements will be used as case-studies (with permission of course), either by name or anonymously (identified by market sector perhaps).

Here's your chance to have your say and be present (by proxy if not physically) at the premier OpenVMS event of the year!"

Poll - Differentiators

- VMS Integration
  - AST event driven model
  - OPCOM
  - SYSUAF
  - ACME
  - DLM
  - Mailboxes
  - Cluster ‘awareness’

These ‘differentiator’ slides try to list the attributes closely integrated with VMS or that are VMS-specific that poll respondents have expressed as elements that contribute to their choice of WASD in preference to Apache.



Note the ‘cluster-awareness’ demonstrated by this single node. The three other instances of WASD running on three other systems in the cluster are shown in the bottom right of the administration page. By selecting the ‘code’ or ‘cluster’ radio selector any of the server-control functions available from this page can be applied to this instance alone or to all instances in the cluster.

WASD is aware of, and interacts with, other WASD *instances* running on the same system and other systems in a cluster using the Distributed Lock Manager (DLM).

This slide also illustrates the primary *Server Administration* on-line, menu-driven server access and control interface.

Poll - Differentiators

- Performance
  - AST event driven
  - Single process model
  - Conservative resource consumption
  - Scripting

Poll - Differentiators

- Monitoring and Administration
  - WATCH
  - Server configuration (loaded)
  - WATCH
  - Server statistics
  - WATCH
  - $HTTPD/DO=<something>[/CLUSTER]
  - WATCH

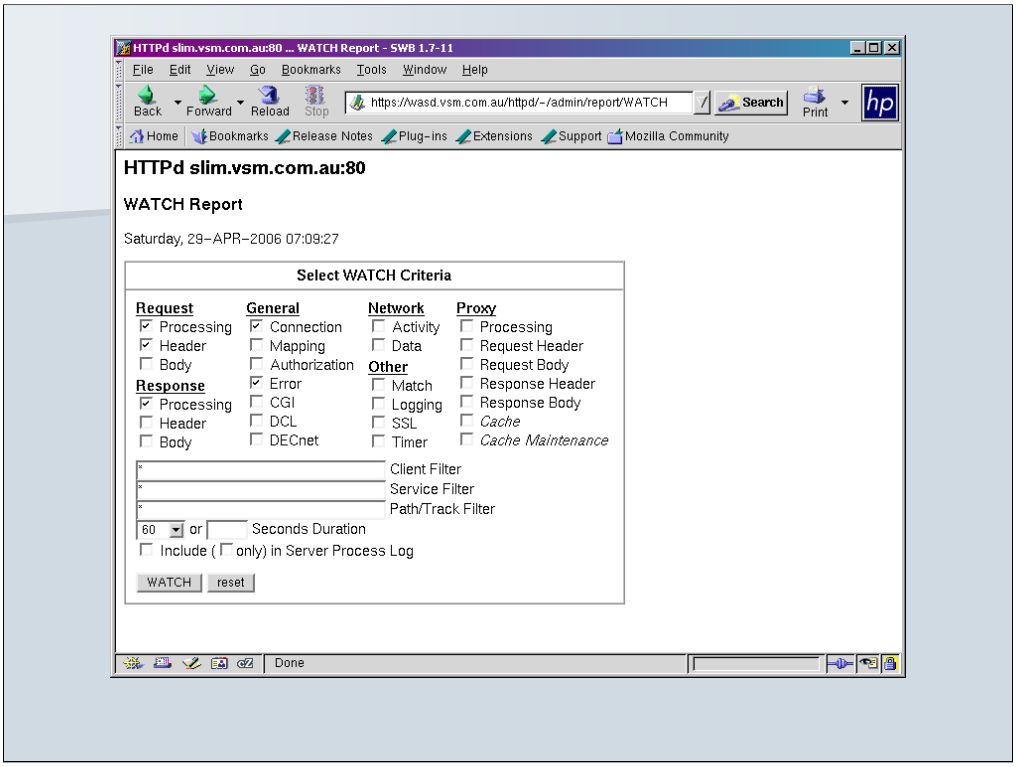
WATCH is mentioned as a strong differentiator time and time and time again. So what is it?

Poll - Differentiators

WATCH ...

provides an online, real-time, in-browser-window view of request processing in the running server. The ability to observe live request processing on an ad hoc basis, without changing server configuration or shutting-down/restarting the server process, makes this facility a great configuration, problem resolution and application development tool.

WATCH executive summary.



WATCH is arguably the single most useful WASD tool available to site administrators and web developers alike and was reported a valuable adjunct during development of this EDS solution.

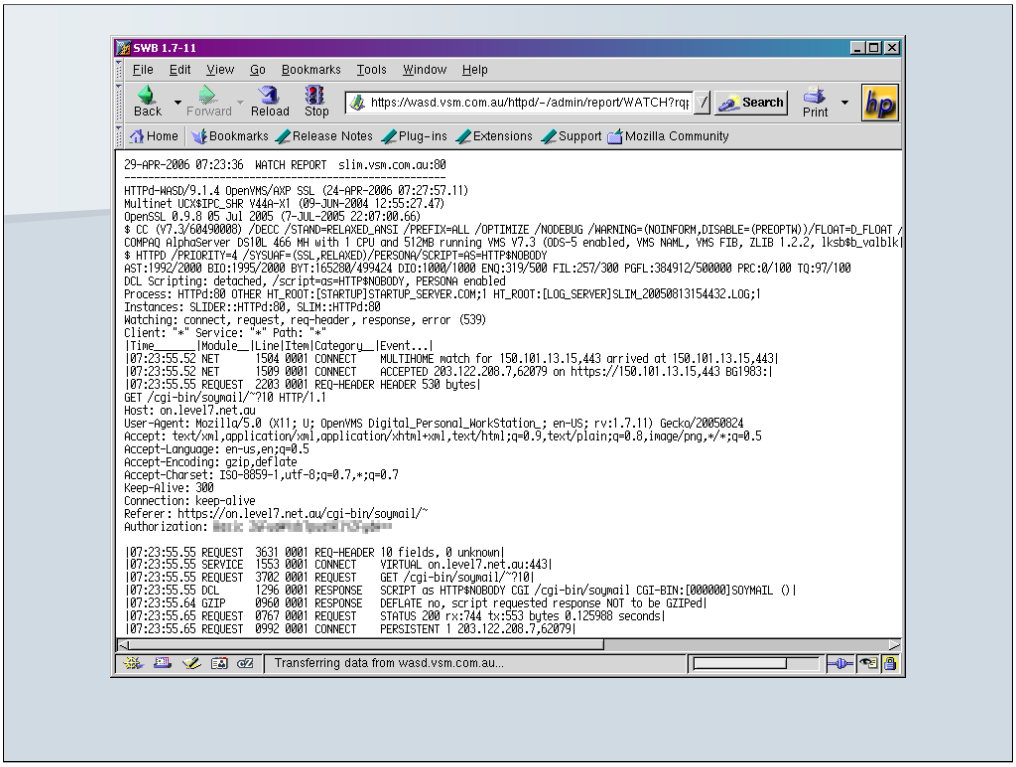
It provides an online, real-time, in-browser-window view of request processing in the running server. The ability to observe live request processing on an ad hoc basis, without changing server configuration or shutting-down/restarting the server process, makes this facility a great configuration and problem resolution tool. It allows (amongst other uses)

- assessment of mapping rules
- assessment of authorization rules
- investigation of request processing problems
- observation of script interaction
- general observation of server behaviour

This slide shows the item selection menu.

Various elements and levels of detail in request processing can be selected according to requirement. Filters allow selection of specific requests against characteristics (for live servers) reducing the quantity of data provided.

Hitting the [WATCH] button produces the report illustrated on the next slide.



This slide displays an example WATCH report.

WATCH reports are a valuable adjunct to configuration and server troubleshooting and are often provided by users of WASD during problem analysis and resolution.

Each report begins with a section describing the server software, it's hardware and software environment, it's startup parameters, and other information relevant to troubleshooting.

After that is a series of event point reports. These are time-stamped, have the source module name and line number, a unique WATCH number so that events can be related to specific requests in progress, and then some event information. This is intended to provide an indication of the status of that item at that stage in the processing. The event point may provide further data in free-form related to the processing.

In the example above the prologue section can be seen terminated by the request event header

```
|Time_____|Module__|Line|Item|Category__|Event...
```

after which a network connection can be seen being accepted, the HTTP request header seen to be displayed, and then a script being activated as one of a number of major points in responding to the request.

Poll - Differentiators

- Scripting
  - Performance
  - IPC based on mailboxes
  - CLI activation and DCL symbols
  - CGIplus / RTE
  - Persona
  - CGI response header directives
  - OSU emulation

Poll - Differentiators

- Proxy
  - Standard proxy
  - Reverse proxy
  - Tunneling
  - Disk cache model and implementation

Poll - Testimonials

- "I have stayed with WASD because I like the product … Apache would have to be very much better than WASD … and I don't see that happening, ever."
- "WASD just ran from the FREEWARE CD copy … Besides it was also a fast server which hardly needs significant attention. It is well designed (IMHO) and sports a huge number of interfaces …"
- "Back in the beginning of the century [☺] we were faced to the necessity of making a lot applications available through the web. Most were based on the VMS security model. The ability to to run scripts under the user's persona …"
- "WASD has been performing very well in our demanding conditions. It is even resilient enough to keep serving pages even when there was a hideous bug [☹] that killed some server processes, thus keeping us up. We couldn't be happier with the software and with the excellent mood of its author."

These 'testimonials' have been culled from responses to my poll on the info-WASD mailing list requesting input for this session (see extract on earlier slide notes). Some responses were long (much, much too long for these slides) while others were short and sweet. I have attempted to put a representative smattering of comments from the contributors. Those posted publicly can be found in the info-WASD archives

http://wasd.vsm.com.au/ht_root/other/info-WASD.html

Some were supplied privately.

They are, necessarily, personal opinions (but no less valuable for being that).

Some are from those where English is not their first language. I have not presumed to 'correct' these for grammar or syntax. The sentiments need no correction.

Thanks to all those who contributed and may recognize their words immortalized herein!

Poll - Testimonials

- "Although I have not used HP support for Apache, I have found that HP support for other HP-ported products (Kerberos, SSH, COM, etc.) to be a little difficult to obtain ('Uh, do we support that product?') … rather than 'Here is how you do it, sorry my docs weren't clear' or 'I will build that into the next release'…"
- "I found the experience to be easy, and the support (documentation and mailing list) to be far superior to any other of the webservers."
- "WASD comes into the scene. It had the needed reverse proxy feature that made possible a connection to a Tomcat server (or to any other protected server) and had an excellent privilege separation model. We ported our configuration and, in record time, we had a test system running, that let us move quickly into production."

Poll - Testimonials

- "[VAX 6000 running VMS 6.0] … we have been the 'low-end hardware/software' beta test site … not that you would notice. WASD's betas are arguably more stable than other people's production releases."
- "… there is the best in class support that comes from the southern hemisphere. The kind of support you dream of … you find a problem, send a mail, go to bed and, when you wake up in the morning, there is a cute [☺] answer!"
- "Can't improve on all the responses, but in terse terms … VMS integration, security, performance, extensibility, reliability, support. Use Apache? … Haven't used it under VMS, but have under Tru64 and it is cumbersome by comparison."

Rumination

WASD: why is it chosen when there's OpenVMS Apache?

Let's chew-over what has been presented in this session.

Rumination

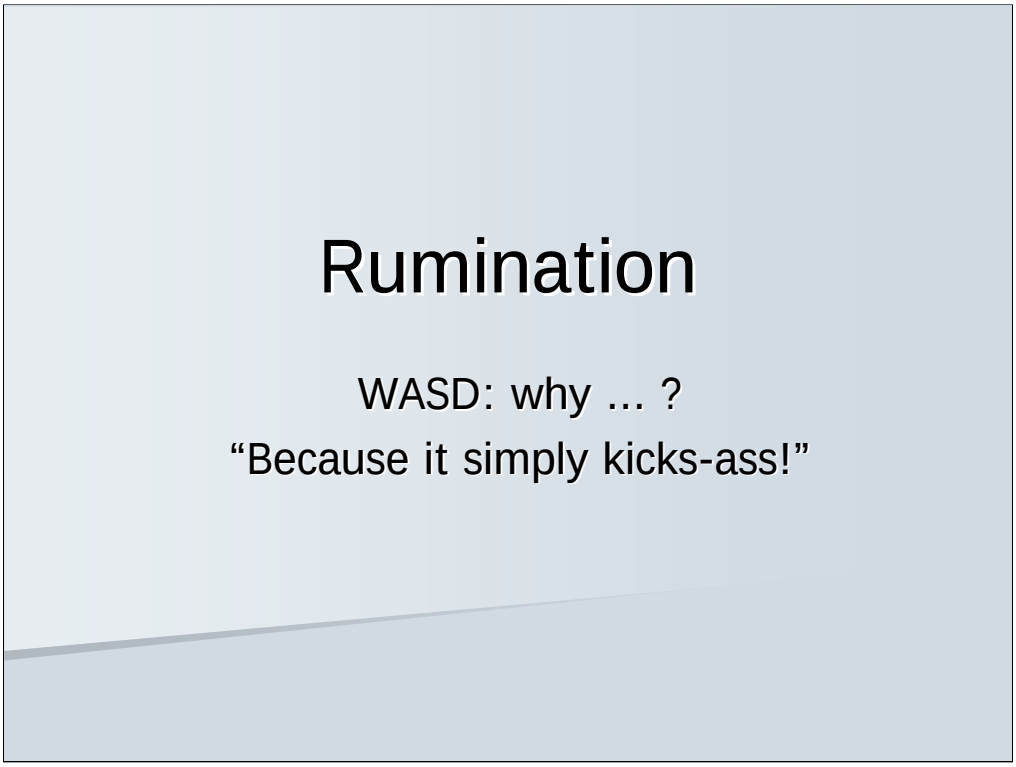
- WASD does everything Apache does
- WASD CGI + CGIplus/RTE persistence
- WASD performance
- WASD conservative resource usage
- WASD/OpenVMS integration
- WASD tools – e.g. WATCH
- WASD reliability
- WASD support

And … no, the repeated presence of the string “WASD” on this slide is not an effort at subliminal persuasion through logo repetition ☺

Rumination

WASD: why … ?

“Because it simply kicks-arse!”



And a translation for the USA portion of the audience.

Demonstration?

- Find me during the Bootcamp.
  - +61 407 883422
- We'll sit down at an Internet kiosk and spend some time at WASD sites large and small.
- Want to know more about WATCH? Ditto!
- Where to get it?
  - http://wasd.vsm.com.au/wasd/

Questions?