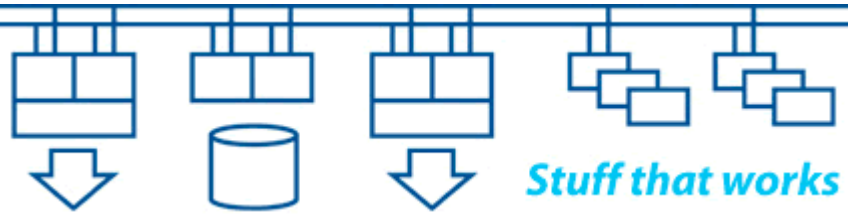
HP Sweden OpenVMS Update

Performance - getting the best from your new platform

Colin Butcher

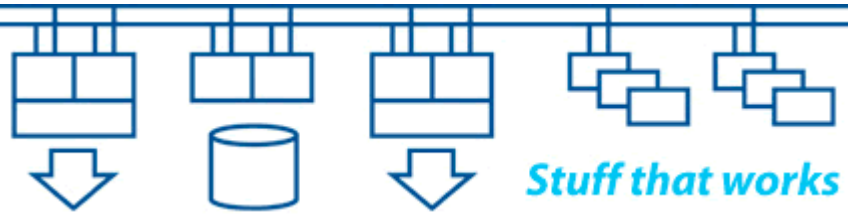


Overview

- Integrity is different from Alpha and VAX, so we should make appropriate changes for performance
- Setting the new systems up for good performance is one of the hidden aspects of porting applications
- We need systems to easily handle the normal workload
- We need systems to be capable of absorbing unexpected spikes in workload without problems
- We don't want to spend our time managing performance and doing tuning exercises

Topics

- Principles of performance
- Designing and implementing scalable software
- Making use of multiple CPUs (cores and hyperthreads)
- Making use of memory
- Making use of the storage IO subsystem
- Making use of the network IO subsystem
- Hardware platform – power-up / restart / dump
- Q & A

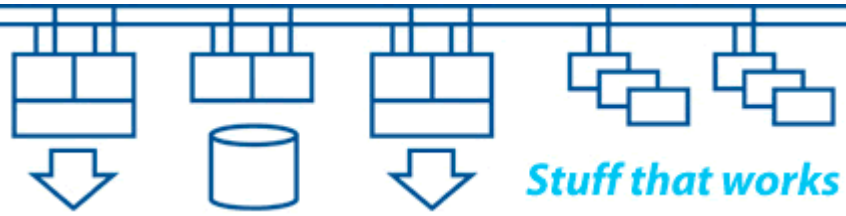


How can we characterise performance?

- What is "fast" or "slow"?
- What are the performance requirements of the system we're designing or investigating?
- What can we measure?
- What comparisons can we make?
- What "footprint" can we look for?

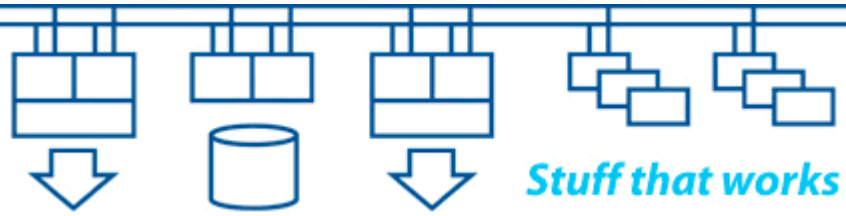
Bandwidth – determines throughput

- Worst case is the maximum capacity of the smallest path in the system (the "bottleneck")
- It's not just "speed", it's throughput in terms of "units of stuff per second"



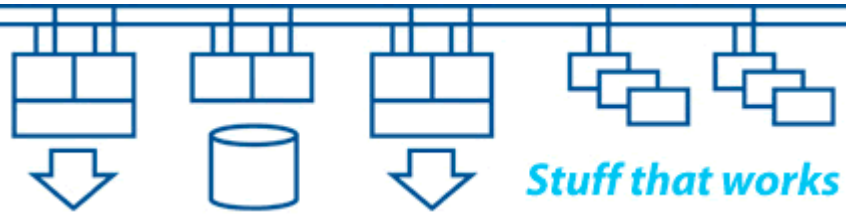
Latency – determines response time

- Worst case is the sum of all the delays in the system
- Latency determines how much "stuff" is in transit through the system at any given instant
- "Stuff in transit" is the data at risk if there is a failure

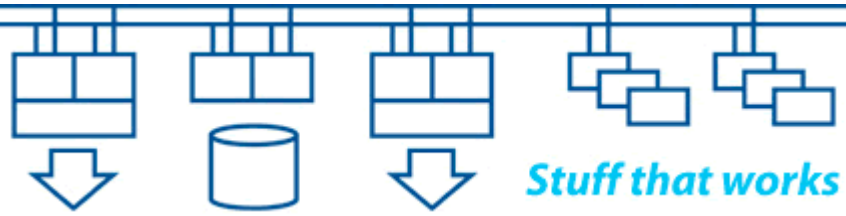


Jitter ("div latency" or variation of latency with time)

- Determines predictability of response, ie: timeouts
- Ideally zero
- Wildly fluctuating latency is a "bad thing"
- Latency fluctuations can cause system failures under peak load

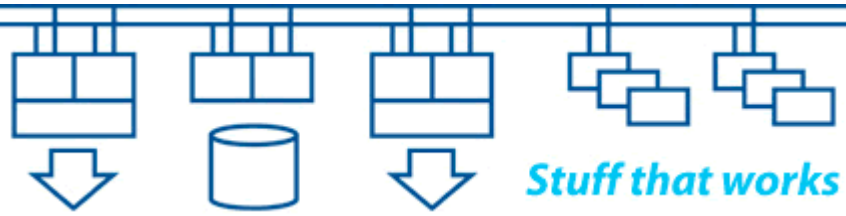


- Size systems to cope with peaks in workload
- Minimise "wait states" (caches, parallelism)
- Minimise contention for resources and data structures
- Understand the need for synchronisation and serialisation of access to data structures
- Maximise "User Mode", minimise the other modes:
  - The fastest IO is the IO you don't do
  - The fastest code is the code you don't execute



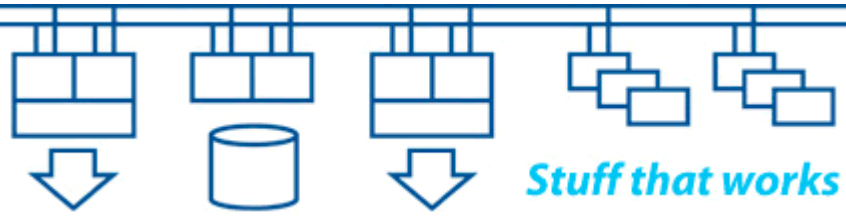
- Understand scalability – do as much as possible once only, do little as possible every time

- Understand how the applications could break down into parallel streams of execution:
  - Some will be capable of being split into many small elements with little interaction between the parallel streams of execution
  - Some will require very high interconnectivity between the parallel streams of execution
  - Some will require high-throughput single-stream processing



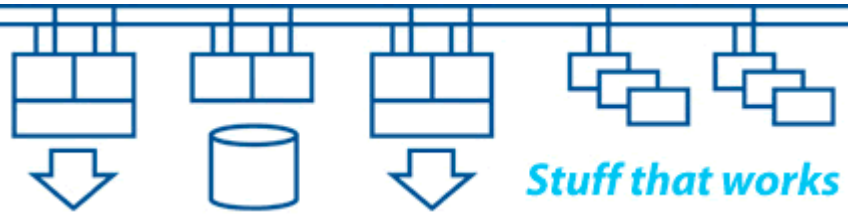
- Check for code making assumptions that the system is a uniprocessor machine:
  - Flags controlling access to an entire global section
  - Loops polling for flag status changes (spinlocks)
  - Data structures not protected from operations that may happen in parallel instead of sequentially
- Use the lock manager to serialise and synchronise access to data structures
- Minimise wait states by having appropriate granularity of access to data structures
- Take null locks out, then simply convert them as needed

- Generate the Instruction and Data streams for processing by the system
- Different types of instructions and data are split out into separate sections (shared data, read-only data, local read-write data etc.) for use by the linker
- Generate code for a specific machine architecture
- Optimisation re-orders the code to take advantage of hardware parallelism and processing efficiencies
- Generate debug information
- Linker lays out the image address space and provides hooks for the image activator



Limits to software performance

- Ability to make use of hardware parallelism
- Granularity of data structures
- Synchronisation techniques
- Serialisation techniques
- Scalability techniques
- Compilers
- Application design
- Designing and writing very good code requires very good programmers



- Compilers are the key to performance
- Use the documented mechanisms provided by the operating system – read the release notes and new features, then use the current mechanisms
- Exception handling (LIB$SIGNAL etc.)
- Data alignment (unexpected alignment faults)
- Floating Point format (IEEE by default)
- Implicit assumptions
- Debugging, ELF & DWARF formats
- Integrity calling standard and register usage

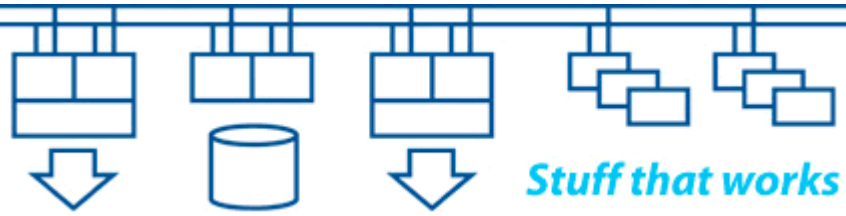
Java Virtual Machines

- Java is not a compiled language running native instructions – it uses an intermediate form known as "byte codes" which are processed by the Java run-time environment (Java Virtual Machine)
- The Java run-time environment uses significant amounts of memory and performs "garbage collection" intermittently to remove objects no longer in use
- Tuning a system to run Java well can be "interesting"
- Hotspot VM and JRAT on Integrity only
- Netbeans profiler

Making use of CPU

- Hyperthreading – affects entire box, very workload dependent
- Fastpath IO devices and distributed interrupt handling
- Introduce parallelism into your code and batch jobs where possible
- Dedicated CPU for lock manager (local locking)
- Compression and encryption, eg: SCS compression, BACKUP compression
- QUANTUM, workload dependent – many SYSGEN parameter defaults changed in V8.2
- Power management

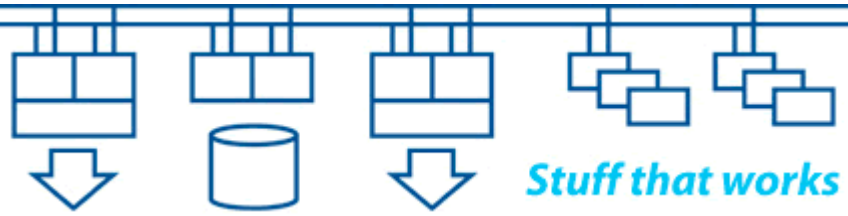
- Use memory (XFC, resident images, DECram etc.)
- Revisit working set sizes - WSMAX and process quotas (WSDEF / WSQUO / WSEXT / PGFLQUO)
- Use RMS global buffers
- Revisit RMS system defaults
- XFC - beware "double caching" and introducing performance penalties
- GH regions – map lots of memory with a small number of page table entries
- INSTALL /RESIDENT and GH region size
- 64bit P2 space and memory reservations
- DECram and HBVS to disc



- FC bandwidth is important – what else are you sharing your storage bandwidth with? Why?
- Rotational latency no longer matters, nor does balancing the IO load to the spindles
- SAN zoning, preferred paths and inter-site links
- EVA cache size and volume characteristics (cluster factor, extend quantity, volume expansion etc.)
- HBVS – only shadow what you really need to
- HBVS – many shadow sets let you control how rapidly shadow copying proceeds during recovery
- HBVS mini-copy and mini-merge policies
- HBVS block count to match EVA cache size

Making use of network IO

- Split protocols across adapters
- Jumbo frames (LAN_FLAGS and LANCP)
- SCS window size (SCACP CALCULATE)
- TCPIP "failsafe IP" and IP addressing scheme design
- 802.1Q VLAN tagging
- DECnet-Plus single rail with LAN failover v multiple rails with load balancing
- Understand latency effects of long-distance networks - check the real distance!
- Fast LAN transmit timeout (LAN_FLAGS bit 12)
- TCPIP - keepalive, delack, MTU discovery
- RMS block sizes for network IO



- Power-up / boot / dump / restart time
- Integrity Server memory tests on power-up
- Where to put crash dumps? Local or FC?
- Compressed selective dumps
- EVA configuration – disc groups, VRAID 0+1, levelling behaviour
- Virtual tape libraries
- EVA mirrorclones, snapshots etc.
- EVA CA – FC "fast write" behaviour

Thank you for your participation

Q & A