Systems performance engineering

Connect Germany May 2018 (Leipzig)

Understanding availability and performance

Colin Butcher CEng FBCS CITP
Technical director, XDelta Limited
www.xdelta.co.uk

XDelta – what we do

- Lead mission-critical systems projects:
  - Strategic planning
  - Technical leadership
  - Project direction
- Minimise risk of disruption to business:
  - Design for change while in continuous operation
  - Prepare in advance for ease of transition
- Ensure long term success through skills transfer
  - Mentoring and teaching

Agenda

- Introduction
- Performance concepts and principles
- System platform and infrastructure
- Software and coding
- Design and testing
- Trouble-shooting

Introduction

- What do we mean by “system” ?
- Performance is only part of the whole problem

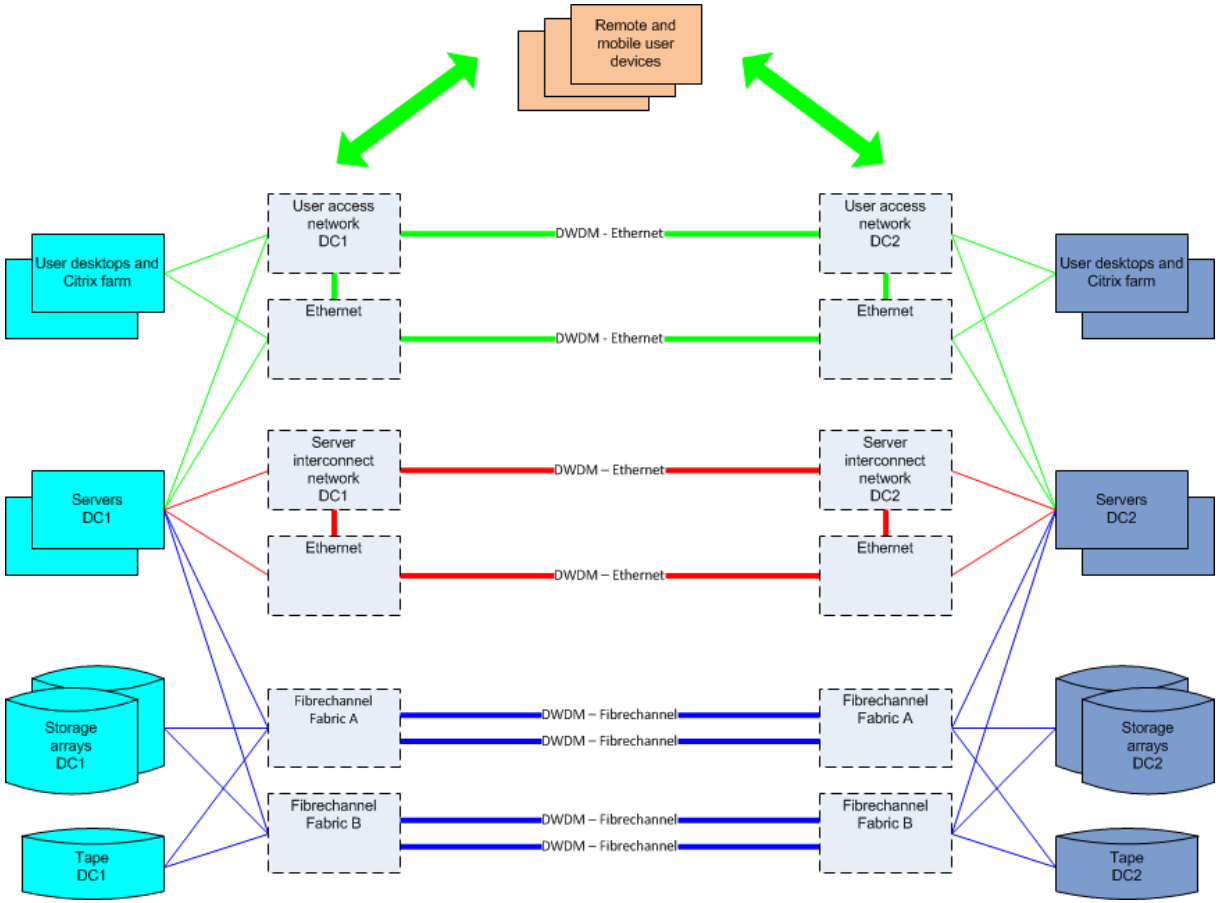
Systems engineering

… is a multi-disciplinary and holistic approach to creating something to meet a specific purpose.

From the NASA Systems Engineering Handbook, June 1995:

“[Systems engineering] is a field that draws from many engineering disciplines and other intellectual domains. The boundaries are not always clear, and there are many interesting intellectual offshoots.”

What constitutes a system ?



xdelta

Performance, security and availability

A system that does not meet its performance requirements or that is vulnerable to attack is a system that is at risk of being unavailable.

Security features such as encryption, monitoring and alerting can have significant performance overheads.

Performance related failures are often transient and exceedingly difficult to fully understand and resolve.

Systems need to have sufficient capacity and inherent performance to deal with the workload within an acceptable period of time under normal, failure and recovery conditions.

What does success look like ?

- What is the purpose of the system ?
- What are the consequences of failing to perform ?
- What are our performance criteria ?
- How can we demonstrate that we've met them ?

Principles of performance

- Performance characteristics
- Parallelism
- Scalability
- Abstraction layers

Performance characteristics

- Bandwidth – determines throughput
  - It's not just "speed", it's "units of stuff per second"
- Latency – determines response time
  - Determines how much data is in transit
  - Data in transit is at risk if there is a failure
- "Diff latency" (variation of latency with respect to time) or "jitter" - determines predictability of response
  - Important for establishing timeout values
  - Latency fluctuations can cause system failures under peak load

Current technology trends - parallelism

- Move from low core count, high clock rate processors to high core count, low clock rate processors
- Multiple I/O paths for storage subsystems
- Multiple I/O paths for network interfaces
- Implicit assumption is that parallelism can be achieved

Parallelism – how can you make use of it ?

Understand how your workloads could break down into parallel streams of execution:

- Some will be capable of being split into many small elements with little interaction
- Some will require very high levels of interaction and connectivity
- Some will require high throughput single-stream processing

Exploiting parallelism

- Serialisation of access to data structures
- Synchronisation of access to data structures
- Communication between parallel streams of execution
- Don’t leave it all to the compilers
- Algorithm design is the key to better systems

Capacity and scalability

- Contention and saturation – running out of capacity
  - What else are we sharing our capacity with ?
  - Queuing theory

- Increasing the capacity of the overall system:
  - “Scale up” or “vertical scaling” – adding resources to a machine or buying a bigger machine (CPU count, memory, I/O adapters, etc.)
  - “Scale out” or “horizontal scaling” - adding more machines

Abstraction layers

"All problems in computing can be solved by introducing another layer of abstraction."

"Most problems in computing are caused by too many layers of complexity."

We need to strike a balance that is appropriate for the kinds of systems we're building.

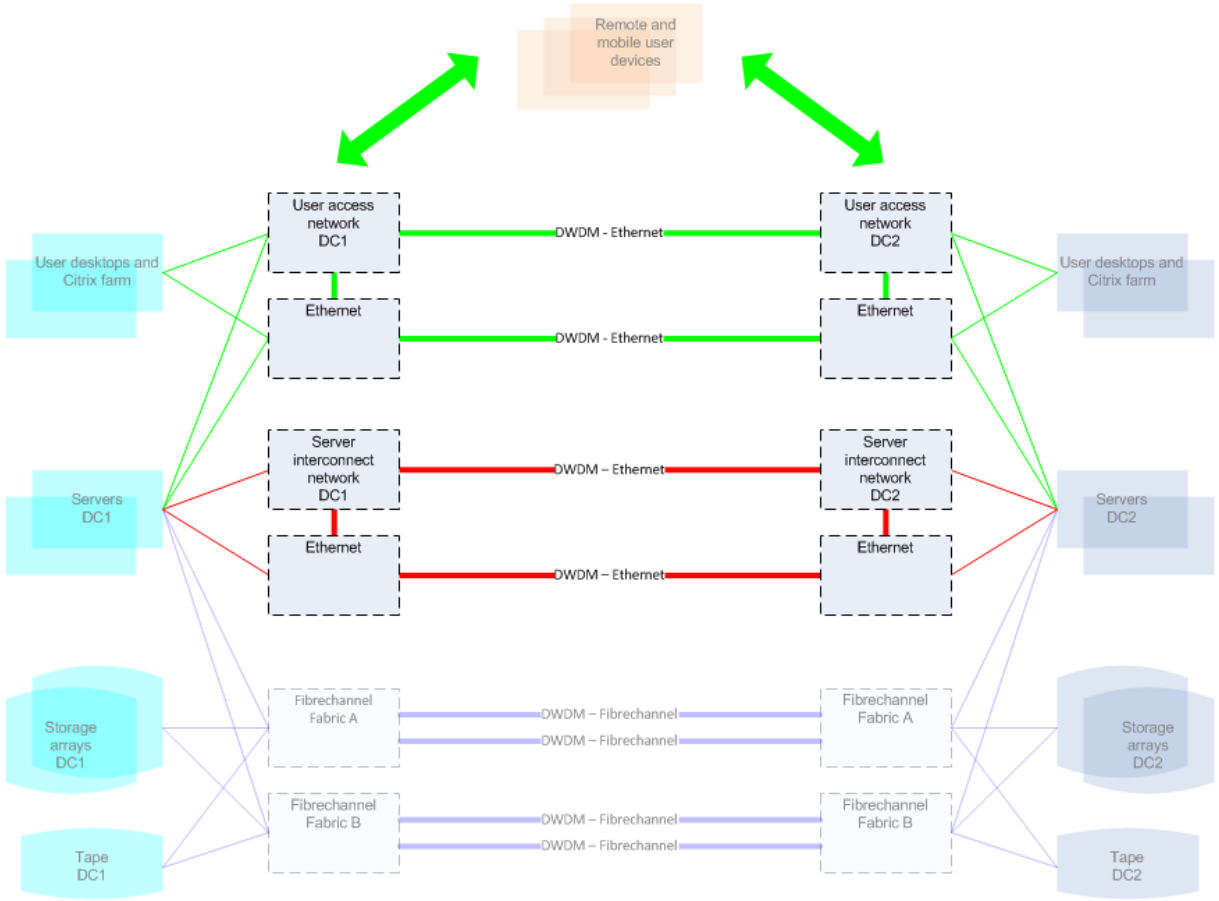
Using abstraction layers

- What looks like your dedicated resource is just a slice of a much bigger thing over which you may have little control:
  - What looks like a network isn't the whole network
  - What looks like a disc isn't a disc
  - What looks like memory isn't all of memory
  - What looks like CPUs aren't all the CPUs
- The operating system allocates and manages machine resources
- It's even more complicated in a virtualised environment

The “Hall of mirrors”

- You can’t see everything that’s going on
- The view is often distorted
- Hiding things makes it easier to deal with the bits you’re interested in
- Hiding things makes it much harder to understand what’s happening, especially with performance related problems

Data network infrastructure



Data networks

- Availability – avoid single points of failure
- Traffic segmentation - VLANs
- Traffic management – QoS
- Load balancers
- Firewalls
- Traffic optimisation

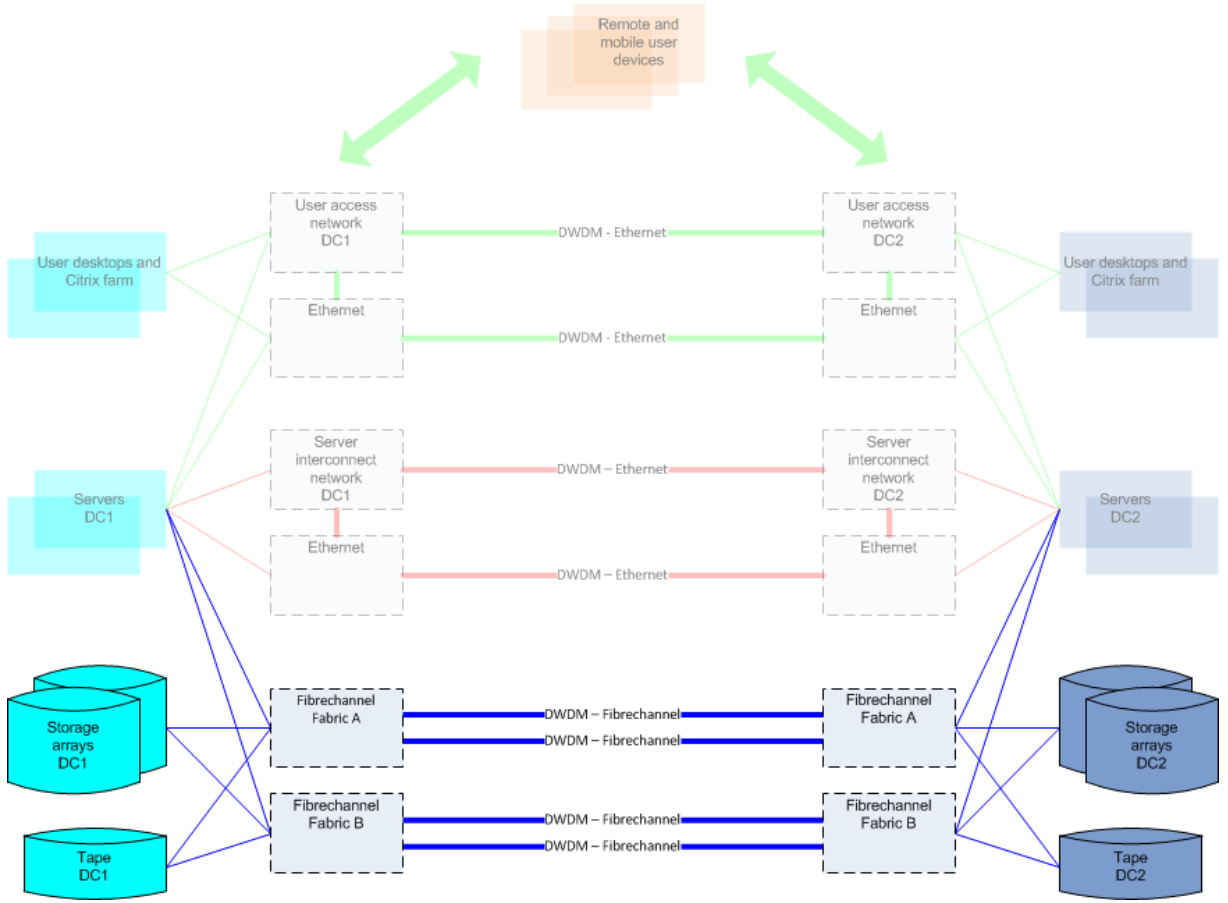
Data networks - segmentation

- VLANs are used to segment a data network:
  - Implemented by using 802.1Q tagging of packets
  - Systems can behave as if they are switches and send tagged packets for multiple VLANs over the same NIC
  - Switch configurations generally map Layer 3 IP V4 subnets to Layer 2 VLANs and enforce IP routing between VLANs
  - Extended VLANs can span multiple sites
- Firewalls and Load balancers
- WAN traffic flows

Data networks – performance issues

- What happens when network paths fail / recover ?
- What happens when network paths become saturated ?
- Can we use all of the available bandwidth ?
- QoS (Quality of Service)
- End to end encryption brings new challenges for firewalls and traffic optimisation

Storage infrastructure



xdelta

Storage networks

- Connectivity between servers and storage devices
- iSCSI – uses data network infrastructure
- Fibrechannel – infrastructure designed for storage data
- SAN fabrics (Storage Area Networks)
- Multi-path devices

Storage arrays

- Array controllers - large protected and mirrored caches
- Array controller “hides” the behaviour of the physical devices and distributes the I/O load
- Array based operations – snaps, clones
- Performance issues:
  - bandwidth to and from the array controller pair
  - contention by systems for access to the storage array
  - controller processing overheads (eg: RAID 0+1 v RAID 6)

Multi-site data replication

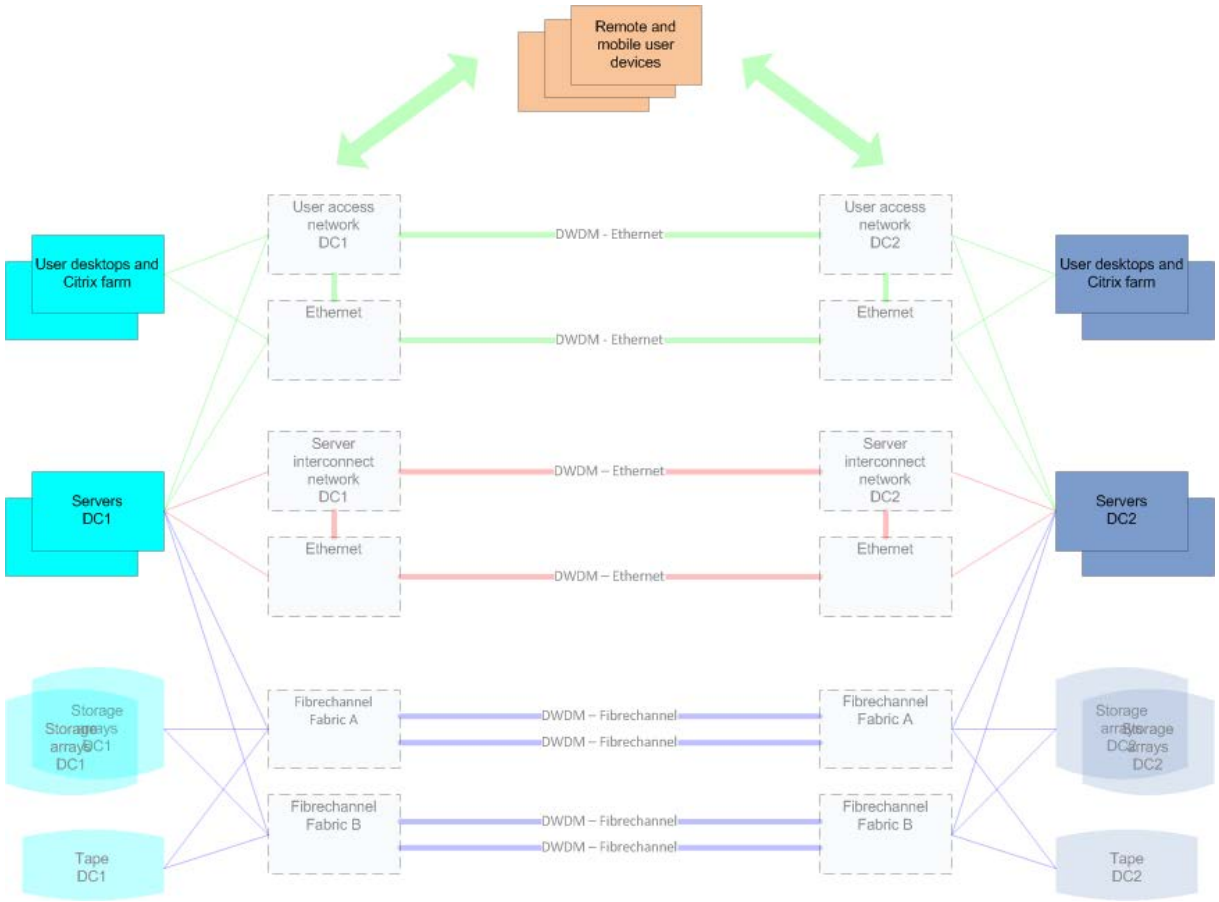
- High availability, Disaster tolerance, Disaster recovery

- Synchronous or Asynchronous replication ?

- Maximum I/O write rate limited by distance latency and number of information exchanges needed to write data
- Data integrity and performance overhead of checking
- Recovery from failure requires bulk copying to get back to fully replicated data

- I/O reads occur from local storage

System platform – hardware and OS



Hardware platforms

- Big high-end multiprocessor systems
- Large “server farms”
- Stand-alone systems for highly secure or mission-critical / safety-critical environments
- Current trend is high core count / large memory machines
- Virtualisation runs a hypervisor on the physical hardware platform to host many virtual machines

Hardware platforms - blades

Blade technology brings virtualisation of the system infrastructure (chassis components):

- Virtual connections from processing components over backplane channels
- Modular systems provide great flexibility of configuration and interchangeability of components

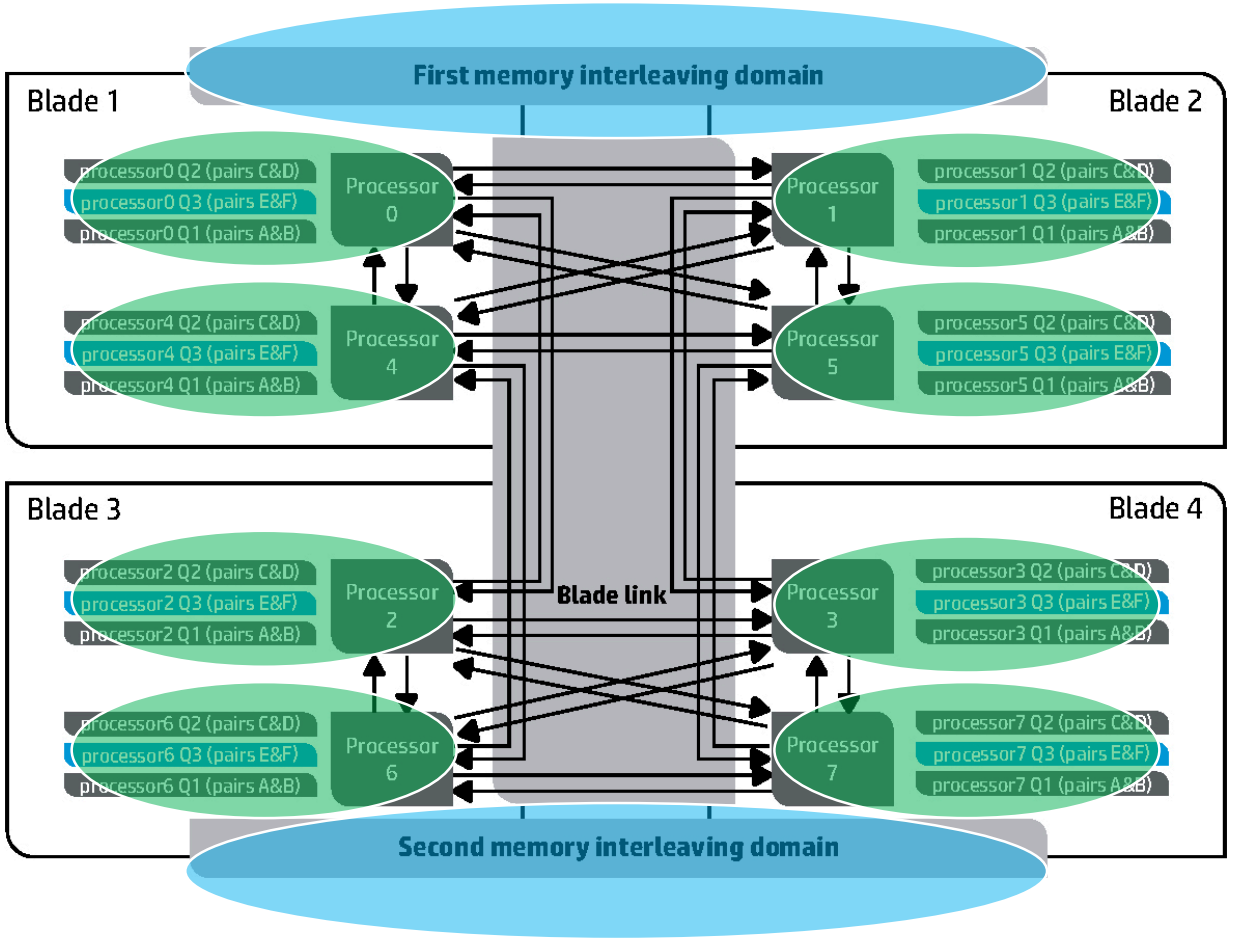
Hyperthreading

- Implements parallelism within a single CPU core
- Two co-threads allow one thread to proceed while the other thread waits for data or instructions - overlap of memory fetch operations
- “CPU” count will appear to double when enabled
- Does this fit your workload ?
- Beware the “idle loop” !

NUMA (non-uniform memory access)

- Locality of memory and I/O devices becomes important
- Set memory interleave behaviour at machine firmware level
- How to lay out data structures for equitable access ?
- Understand how the OS uses memory for caches
- How to exploit system for best performance ?

NUMA architecture – HPE bl890c-i4 blade



Operating systems

- Share out and arbitrate access to machine resources
- Protect users from each other
- Give the illusion that each user has the machine to themselves
- With virtual machines, the hypervisor is a “master” OS with the “guests” nested inside
- Java comes with its own run-time environment

OS performance tuning

- Usually has a limited effect, unless systems are badly set up in the first place
- Need to set systems up to suit intended workload (system parameters, process quotas, network buffering and packet sizes, cache sizes, I/O paths to storage, etc.)
- Most improvements come from working on the code
- Must have metrics to understand behaviour

Virtualisation and Cloud Services

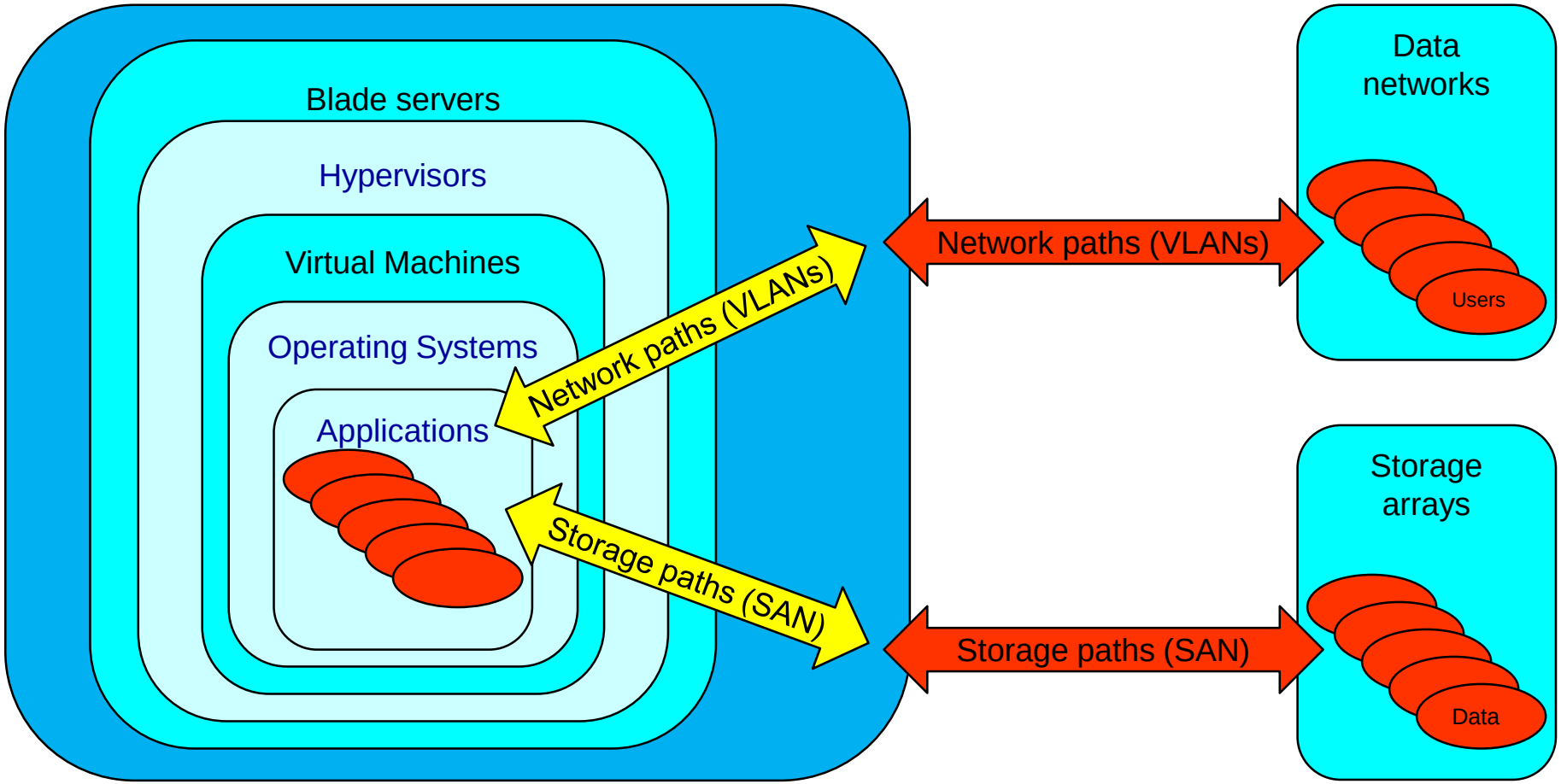
"When I use a word, it means just what I choose it to mean - neither more nor less."

Humpty Dumpty, Through the Looking Glass,
by Lewis Carroll.

Virtual = .NOT. Physical

xdelta

Abstraction layers in a virtual world



xdelta

Workloads in a virtual world

- To the “hypervisor”, each and every virtual machine is a workload needing physical hardware resources
- Within a virtual machine, each application (and the operating system overhead) is a workload
- What level of interaction is there between the virtual machines that run your applications ?
- What happens when your host hardware runs out of resources or when virtual machines move to another hardware host ?

Accessing cloud services

- Your data are not in the same location are you are
- You are entirely dependent on your network connection
- Bandwidth and latency govern the behaviour you get
- Does it matter where your “stuff” actually runs ?
- Unsuitable for high I/O write rate systems
- Unsuitable for low / consistent latency systems

Design and testing

- Design for performance
- Load testing
- Understand the whole system

Designing for performance

- Size systems to cope with peaks in workload

- Eliminate “wait states” as best you can

- A faster machine just waits more quickly!
- Don’t make it go faster, stop it going slower
- The fastest I/O is the I/O you don’t do
- The fastest code is the code you don’t execute
- The idle loop is anything but idle

Code paths and data flows

- How does your code scale up ?
- Separate out static data from dynamic data
- Minimise frequently executed code paths
- How to reduce impact on system and surrounding network and storage infrastructure ?
- Think parallel, not sequential

Writing scalable code

- A uniprocessor machine can only execute one code stream at once – there is no inherent parallelism
- Check for code making assumptions that the system is a uniprocessor machine:
  - Flag bits controlling access to an entire memory region
  - Loops polling for flag bit status changes
  - Data structures not protected from operations that may happen in parallel instead of sequentially
- Use the OS mechanisms (locks) to serialize and synchronize access to data structures
- Minimize wait states by having appropriate granularity of access to data structures
- Take null locks out, then simply convert them as needed

Compilers

- 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 target 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

I/O performance v CPU performance

- Physical I/O operations typically take a few milliseconds to complete

- We can execute a lot of CPU instruction cycles in a few milliseconds (1 GHz = 1 nanosecond, thus 1 million instruction cycles per millisecond)!

Analysis tools and instrumentation

- Code analysis tools can help with finding interactions and heavily executed code paths
- Build instrumentation into your software, then you don't change its behaviour by adding temporary code
- Pay attention to state transitions and event timing
- Wait states and cache synchronisation are expensive

Testing

- How can we simulate realistic scenarios ?
- Test for scale, not just functionality
- Test to find out what really happens under load and under failure conditions
- Performance failures are usually transient, so how will you capture fine-grained enough data to capture a problem ?

Trouble-shooting

- Understand the whole system
- Performance data and trend analysis
- “It’s slow” – what next ?

Performance data and trend analysis

- Without data for historical comparison, how do we know what's reasonable ?
- Without data, we're guessing
- Data needs to be synchronised in time across everything
- Don't jump to conclusions – correlation does not imply causation
- Most problems are combinations of several things

“It’s slow” !

- What do they mean:
  - Is it responding poorly ?
  - Is the responsiveness varying too much ?
  - Are batch jobs running slowly ?
  - How long are key business processes taking ?

- Is the expectation unreasonable ?
- Has it always been like that, but something else has changed recently ?
- Is there anything wrong at all ?

Interactions and unexpected effects

- How can we segment a system to minimise performance impact and failure propagation ?
- How can we protect against data loss and corruption ?
- Recovery from failure usually has most impact: eg: data replication back to a known good state
- How quickly do we need to recover ?
- How much data are we prepared to lose ?

Typical problems

- Too many implementation decisions are based on cost without fully understanding the trade-offs

- Most systems are I/O write rate limited:
  - Inappropriate use of virtualisation
  - Inadequate bandwidth between data centres
  - Too great a distance between data centres

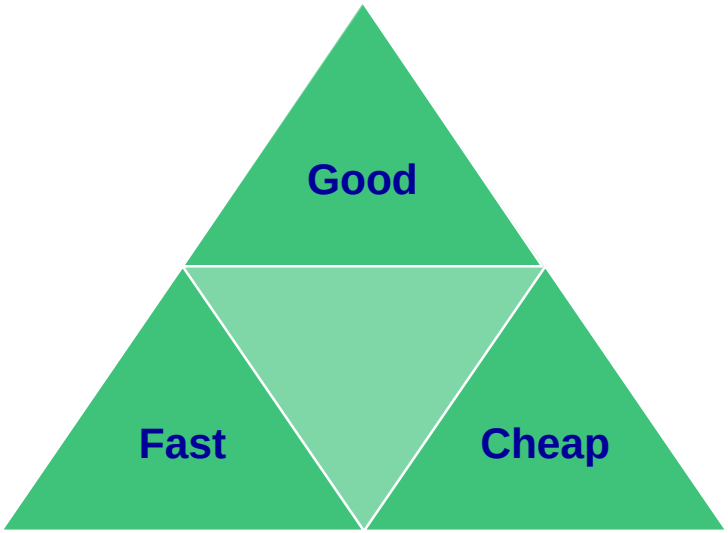
- Systems don’t scale well:
  - Inappropriate use of non-compiled languages (eg: Java)
  - Code not suited to parallelism with many instances
  - Poor database schema design – insufficient granularity

Summary

- Performance cannot be considered in isolation
- Think parallel, not sequential !
- Performance can’t be easily added later
- Testing must include realistic loads and scenarios
- Trouble-shooting and resolution requires good information and a thorough understanding of the whole system

Conclusion

- You usually get what you pay for !



Systems performance engineering

Connect Germany May 2018 (Leipzig)

Thank you for your participation

Colin Butcher CEng FBCS CITP
Technical director, XDelta Limited
www.xdelta.co.uk