UKCMG Annual Conference 2007

Principles of Performance

Colin Butcher

Overview

Overall system performance depends on matching the system configuration to the requirements of the application. Performance issues are often the cause of system failures and disruption, thus affecting availability.

System designers need to understand both the application and the behaviour of their target systems in order to write code that is reliable, which has minimal performance impact and which scales well in a production environment.

The performance envelope

- **What does the business require the systems to do?**
- **What happens to the business if the systems fail?**
- **What happens if you push beyond the limits?**
- **How far from the edge are you?**
- **How do you know?**

How can we characterise it?

- **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 does “normal” look like?**

Basic principles

- **Bandwidth – determines throughput**
- **Latency – determines response time**
- **Jitter (“div latency” or variation of latency with time) – determines predictability of response**

Basic principles - analogy

Think about a motorway (freeway):

- **What is throughput?**
- **What is latency?**
- **What is jitter?**

Bandwidth

- **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

- **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

- **Jitter ("div latency" or variation of latency with time)**
  - **Determines predictability of response**
  - **Ideally zero**
  - **Wildly fluctuating latency is a "bad thing"**
  - **Poor jitter can cause system failures under peak load**
  - **Small changes can trigger big variations in response**

What can we do?

- **Introduce parallelism and reduce “wait states”**
- **Use caches of different kinds at various levels**
- **Minimise contention for resources and data structures**
- **Understand the need for synchronisation and serialisation of access to data structures**

Parallelism and scalability

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

- **Understand how the applications could break down into parallel threads of execution:**
  - **Some will be capable of being split into many small elements**
  - **Some will require high-throughput single-thread processing**
  - **Some will require very high interconnectivity between the parallel threads of execution**

How do we go looking?

- **Time series performance data is just data – it does not tell us what to do**
- **The sample rate must let us see the transient peaks and spikes in workload**
- **You must understand your workload and know what it looks like over time**
- **There is no substitute for experience to interpret and analyse the data**

Time Synchronisation

- **Need to be able to order and correlate performance data and events across the entire environment**
- **UTC Timestamp format**
- **External reference clocks**
- **Performance data without correlation to external events may be of little use, or worse, can be misleading**

Limits to software performance

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

Software considerations

- Operating system design and “matching” to the hardware
- Operating system configuration
- Performance tuning
- Application design (scalability, data integrity, availability)
- Compilers generate code better than you can
- Use the operating system features rather than “roll your own” mechanisms
- Testing for performance and scalability, not just functionality

Storage infrastructure

- **Size storage subsystem based on minimum components and maximum estimated throughput**
- **Need adequate backup capacity and throughput in order to meet permissible backup windows**
- **Understand application behaviour and storage performance requirements**
- **SANs – beware of contention for access to arrays**

Network infrastructure

- **Scale network so that overall performance is based on minimum essential number of paths and maximum estimated traffic**
- **Understand protocol behaviours in multi-path infrastructures**
- **May wish to take advantage of installed bandwidth capacity to provide additional functionality when everything is working**

Problem solving concepts

	Before	Now	After
Environment			
System			
Component			

Performance investigation

- **What “footprint” can we look for within the system?**
- **What effects does a performance problem cause?**
- **What has changed, not necessarily within the system?**
- **Most performance problems are transient**
- **Is there a problem at all?**

Thank you for your participation

Q & A