Beyond 24x7

High Availability in Higher Education 2010-07-14

S2 – Monday Morning

A rather potted version of a story we're all familiar with in some guise. Point is that the customer knows that there is something important to discuss, but might not have the training or experience to identify the real underlying issues or to express their requirements very well.

S3 – Availability: Plenty of Ideas

S4 – Availability Defined

Note that availability != uptime, since the client might not be able to use it even though the system is up – perhaps due to network errors, or an application fault.

S5 – Scope of Availability Concerns

Point of this slide is to show that availability touches on pretty much every aspect of IT service management.

Some of the more important relationships are:

Capacity: insufficient capacity will lead to availability problems;
Service Level: need to agree and communicate the availability targets, even if these are aspirational rather than contractual;

S6 – The Expanded Incident Lifecycle

A service failure is called an Incident, and we can see that there is a clearly defined lifecycle that is followed each time a fault occurs:

- Occurrence
- Detection
- Diagnosis
- Repair
- Recovery
- Restoration
- Closure

The average interval between detection of successive incidents is the MTBSI.
The average period between detection and normal operation is the MTRS.
The average period during which the service is available for use is the MTBF.

S7 – Quantifying Availability

Availability is often expressed as a percentage of the Agreed Service Time that the service was actually ready to respond to requests.

It's too late in the afternoon for a maths lesson, so if you want to check the working on these ways of calculating availability.

The "obvious" calculation at the top can be expressed using the very specific and measurable intervals that we saw in the expanded incident lifecycle.

In turn, this enables us to see clearly that improving availability is a matter of either decreasing MTRS – faster recovery, or increasing MTBF – fewer incidents during a period of AST.

S8 - The Cost of Downtime

HE isn't like a normal business:

- Hard to define the monetary value of our services
  - Core business activities - teaching, learning, and research - often happen in a fairly flexible manner
  - Hard to identify “mission critical” or even “business critical” applications
- Our customers pay in full, up-front

But still costs us:

- Helpdesk time and contingency arrangements
- Lost hours
- Damage to reputation both of the University (externally), and the IT services team (internally)

S9 – Common Causes

Two main types of cause – planned outages where we can attempt to fit them into agreed maintenance hours in order to avoid impact on availability, and unplanned outages which tend to be harder to mitigate.

S10 - Analysing the Causes

Quite a bit of variation as to what the underlying root causes are, but anecdotal evidence and consultant reports do loosely agree. Here's one from Gartner:

- Other (including system): 20% (hardware, OS, environment, natural disaster);
- Application: 40% (bugs, performance, impact of change);

Operator error: 40% (not performing or performing incorrectly);

The key thing to note is that having the downtime caused by system faults will only improve availability by 10%. Also, there is often additional technical complexity associated with this approach, and that can make things worse on the operator error side.

[* Scott, TG-07-4033 (Gartner), picked up in RFC3439 ("Some Internet Architectural Guidelines and Philosophy", Dec 2002), referenced widely including Sun, IBM]

S11 - What Would Improve Availability?

Will depend on your specific service, organisation, and infrastructure. I found an interesting paper where a group of IT experts were given a list of areas and asked whether best practice in each area would improve availability of IT services. Sufficient people were asked to carry out a statistical analysis and rank the areas in order of confidence that they would improve availability.

The top areas were

- **Change Control -** reduces frequency of incidents, higher MTBF
- **Monitoring -** enables faster response to incident, so lower MTRS
- **Requirements** – understanding which services need to be available and when
- **Operations** - reduces frequency and enables quicker response, so higher MTBF and lower MTRS

Resilience of client/server was down in lowest quartile.

[Proceedings of the 4th International Workshop on Software Quality and Maintainability (2010).]

S12 – Three Steps to Availability Heaven

We can focus on three key areas when we are designing a service with availability in mind.

S13 – Step 1: Understand and Negotiate Requirements

S14 - Negotiating Requirements

The aim is to find a good compromise between the competing desires of the organisation to maximise availability but minimise cost.
We need to remember that we are the technical experts and they are the business experts, so we might need to help them ask the right questions to determine the best solution. For example, if they keep mentioning “24x7” or “critical business function” then it might be that this is all they have ever heard about in this context.

A good approach might be to talk through some typical usage scenarios – a typical day or an annual event that places heavy demands on the people running it.

S15 – Types of Business Application

During your negotiation you will doubtless come across the question of how important the service is to the business. There is a danger of this being driven by the personal priorities of individuals rather than the business needs, so the following classification might help to depersonalise the discussion.

Mission Critical

Customers and vital business functions affected in a way that massively impacts revenue and profitability

- Immediate reduction in revenue
- Damaging for the company's commercial reputation and credibility
- Long-term outage threatens bankruptcy

An example would be the web site of an online shop.

Business Critical

Often not so directly customer-facing but likely to be supporting vital processes

- Indirectly affects revenue generation and may prevent collection of revenue
- Supports customer facing activities
- Inability to collect revenue efficiently
- Long-term outage can significantly reduce company cash flow

An example of a business critical service is the customer billing application.

Business Operational

Out of direct line of service to customer, impact typically seen as reduced productivity

- Internal users only
- Reduced efficiency and increased cost of operations

An example of a business operational service is enterprise messaging.

Administrative Services / Task Critical

Typically the domain of a small user group, with minimal cost implications of outage.

- Internal users only
- Reduced individual performance and productivity

Examples of administrative services applications are desktop applications.

S16 - Availability Offerings

Going back to the planned / unplanned division of downtime causes, we can think about addressing either or both of these types of cause in order to offer a level of availability to match the business requirements.

The starting point with high availability is to take steps to minimise or mask the effects of component failure on users, without necessarily eliminating outages.

If we add in fault tolerance / resilience, to enable the service to operate correctly despite component failure, then we can reduce the occurrence of unplanned outages.

Alternatively, we can arrange for our planned maintenance activities to not disrupt service, eg. by preparing upgraded servers offline and bringing them into service instantaneously rather than upgrading servers in-situ. This is called continuous operation.

If we tackle both the planned and unplanned outages then we're heading for continuous availability.

S17 – Step 2: Select a Technical Solution

S18 - Fault Tolerance

- Redundancy in the box increase MTBSI by reducing likelihood of component failure impacting on service
- Multiple paths to SANs and network

Usually when we talk about resilience we are referring to components within a single server, but you could treat whole servers as components and build in resilience at this slightly higher level. This is called clustering.

S19 – Simple / Client-Based

A very rudimentary type of cluster is a farm comprised of several servers doing the same thing. This relies entirely on the client being able to make intelligent decisions about which member of the farm to contact. This has the attraction of being very lightweight, but the downside is that you're heavily dependent on the user having a well designed client.

One of the typical problems with this is that once a client has started to communicate with a particular member, there may be a delay (timeout) after failure of the member before the client switches to another node. In addition, the way that many clients use round-robin DNS records means that in many cases they pick one of the returned IP addresses and never try the others.

The upside is that you can eliminate planned work causing an outage by removing each node from the farm in turn, carrying out the upgrade or maintenance, and then adding it back to the farm.

S20 - Active-Passive with Heartbeat

Single network interface, configured on both hosts but only up on one at a time.

Heartbeat enables hosts to check on status - if service on one host goes down then the other should STOBITH and bring its interface up., with an appropriate arp announcement to get MAC address tables updated if required. In a deployment where the two nodes are at different sites or on different networks, a third server called a quorum server is deployed to avoid the situation where loss of the heartbeat causes both servers to bring their interfaces up.

Interesting technologies: CARP (cluster shares a virtual network interface - a dynamically assigned master handles external connections); Linux-HA which is a heartbeat implementation for your favourite OS.

S21 - Active-Active with Load Balancing

Multiple nodes - in this case on a private network behind a load balancer. Well, in fact a pair of load balancers to make sure we haven't introduced a new SPOF. Complications in that the load balancers may need to share state information - a classic use case is sticky sessions where once a client has started to talk to one of the cluster nodes then any further connection or communication will be directed to the same cluster node.

Having LBs at this level can offer several fringe benefits, including the ability to off-load any SSL work here and free up the cluster nodes, dynamic addition / removal of nodes during maintenance and upgrade, or routing of particular clients to particular servers - eg. if you offer a differentiated service level or are staging a modified service for a group of beta testers.

I've shown what is called “inline load balancing” where the servers sit on a private network behind the LBs, but a common alternative is the “end node” configuration where the servers and load balancers sit alongside each other on the public network, with clients still connecting to the LBs, but here the servers can also participate in direct connection if necessary.

Taking resilience even further up the chain though, we might want to look at a replica data center - clustering across sites and across networks

S22 - Multiple-site Resilience

Routers can use 802.1q VLANs to effectively present the two sites as a single physical network - so you can share an IP subnet across the two sites.

In a basic configuration however, one of the routers is appointed as a "home" for the VLAN, so if this router fails then you're still down. There are ways to circumvent this, one interesting one is VRRP (Virtual Router Redundancy Protocol) which enables the VLAN master to be established out of any remaining operational routers.

At this stage you have probably have eliminated everything back to the client's own network connection, so it's out of your hands.

S23 - Virtualisation

You can't do a presentation with talking about the "V"-word.

- Virtualisation is not a solution in its own right, but it can provide tools and platform to help realise many of the approaches described previously;
- Often able to dynamically assign resources, so can make active-passive designs rather more efficient;
- Tools perform common tasks eg. deploy new machine from installation template: links into CMS and SOP in management section.

S24 – Step 3: Design a Management Solution

S25 - Configuration Management

CM = system state:
  - configuration files,
  - file modes,
  - package installation state,
  - users,
  - firewall rules
and includes higher level architecture - how separate CIs link together to deliver service

Rather than simply creating a static central store of configuration files and directives, it is common to build these from a number of inputs, or sources. These may comprise of file templates, parameters, and declarations that can be compiled to generate the desired configuration.

The critical part of this diagram, and one which can easily be forgotten, is the periodic generation of a compliance report. This provides notification of any unauthorised changes in the system configuration, or could trigger an automatic deployment to bring the system back into an approved state. As a result, changes to system configuration must be made via the CMS, and this encourages consistent application of standard operating procedures.

Reduce likelihood of operator error (most common?) => increase MTBSI

Simpler (automated?) deployment of replacement systems => decrease MTRS

S26 - Standard Operating Procedures

Standard operating procedures ensure that operational activities are carried out in a repeatable and reproducible way.

It also consolidates experience and draws on lessons learned during development or operation.

A simple example might be the creation of a subversion repository in our RCS. We need to create the filesystem components, set file permissions to ensure that the web server has access, and configure subversion ACLs to permit appropriate user access. Then there might be post-commit hooks, entries in an index of repositories and so on. Enough stuff that without it all being written down there's a high chance that it'll be done differently each time or by each person, and that leads to inconsistency and ultimately either service failure or extended MTRS.

S27 - Monitoring

What do you monitor

- Network interface availability (PING)?
- Web server availability (port 80)?
- Web content delivery (HTTP)?

Need to think about trying to measure what that user is seeing.

When to monitor it - hourly, every minute, 5 minutes?

- Depends on the service itself
  - SSL certificate validity: daily
  - HTTP service up: minute
  - Latest data available from upstream: hourly

Why do you want it

- To detect and respond to incidents
- To record trends and make forecasts

How do you know when something has gone down?

- Automatic monitoring
  - Nagios (and derivatives) , Cheops{,-ng} , Zabbix , OpenNMS , ZenOSS , NetDisco
  - WhatsUp , FogLight
- Notification, eg. email, SMS

S28 - Change Control and Other Processes

- Purpose of Change Management* is to ensure that:
  - Standardised methods and procedures are used
  - All changes [...] are recorded in the Configuration Management System
- Incident Management
  - The primary goal [...] is to restore normal service operation as quickly as possible
- Problem Management
  - The primary objectives [...] are to prevent problems and resulting incidents from happening, to eliminate recurring incidents and to minimize the impact of incidents that cannot be prevented.