;login:

THE MAGAZINE OF USENIX & SAGE

June 2001 • Volume 26 • Number 3

inside:

SYSADMIN

Designing an External Infrastructure
by John Sellens

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

designing an external infrastructure

by John Sellens

John Sellens is the general manager for Certainty Solutions (formerly GNAC) in Canada, based in Toronto, and is proud to be husband to one and father to two.

<jsellens@certaintysolutions.com>

This article discusses some of the things that you should consider when you are designing the systems infrastructure for an "external site."

What do I mean by an "external site"? An external site is a computing installation that is intended primarily to provide services for people outside your organization, or which is located at a physically remote location. The most common example of an external site is, of course, a Web server (or servers) located at a co-location facility, but there are other kinds of sites that are "external," such as the point of sale and store-management systems used in a retail store chain, the systems used to transfer purchase orders to suppliers, or an incoming fax server. Most of this discussion is going to be phrased in the context of a Web-serving infrastructure, but much of the discussion will be applicable to other kinds of external sites.

Introduction

Before I start talking about specifics, let's spend some time on a slightly more abstract discussion.

Characterization

External sites usually have some or all of the following attributes:

- Remote location – typically located in a co-location facility of some form
- External users – usually primarily used by customers (or potential customers) or business partners
- 7x24 operation – expected to be "always" available
- Flexibility – needs to be able to cope with changing demands or traffic levels

Internal sites, by contrast, often have fewer demands placed on them. The service loads tend to be more predictable, off-hours downtime is easier to schedule, users are often easier to notify, and the equipment is more likely to be just down the hall, rather than across town or across the country.

Basic Concepts

In this discussion, I'll make use of the following concepts:

Redundancy: The use of multiple instances of particular components to reduce the impact of hardware failure. The most common example is using a pair of disks in a mirrored configuration to guard against disk failures.

Replication: Similar to redundancy, but refers more to the duplication of services than to the server hardware components themselves. The use of multiple SMTP or DNS servers is a common example of replication.

Separation of Functions: The use of separate servers for different services. This is also the "don't put all your eggs in one basket" concept.

Reliability: The ability of a system to cope effectively with failures or unusual conditions.

Accessibility: The ability to gain the appropriate access to your systems, even in times of service, system, or component failure. For example, a modem and a telephone line can be very useful when your regular network connection is dead.

Recovery: The process of restoring an impaired or failed service or system to its normal state.

Security: The appropriate access and other controls that protect your systems and services from attacks (intentional or unintentional). Security in this context is more like a topic than a concept.

I will cover how these concepts can be applied to design an appropriate external infrastructure in order to reduce the risk of site failure, and to shorten the expected time to repair a failure should one occur.

It's worthwhile to note that all of these concepts (and more) can also be applied in the design of internal sites and will result in a better infrastructure. But the nature of external sites, most notably their external visibility and often remote location, makes these concepts especially relevant for external sites.

Adding reliability to a site usually adds additional complexity and cost.

Definitions

I've already defined what I mean by an "external site." Here are a few more definitions:

Service: The actual facility or process provided by your site. We most often think of services like Web content, mail, or FTP, but other common services include such things as calendaring, product catalogs, file storage, and so on.

Server: Most often a computer, used to provide a service, or a part of a service.

Component: A device (typically) that provides some function or ability. This includes such things as disks, network equipment, power bars, and yes, servers.

System: The collection of components that work in concert to provide a service.

Some of the definitions attempted here make some subtle or slight (or even somewhat obtuse and obscure) distinctions, but I thought it would be useful to attempt to distinguish between the four different terms.

Balance

Effective system and infrastructure design often involves a number of tradeoffs – the time-honored "cheap, fast, good – choose two" has a certain ring of truth. In external infrastructure design, we're often faced with conflicts between the three following goals:

Simplicity – Cost effectiveness – Reliability

In other words, adding reliability to a site usually adds additional complexity and cost.

In most cases, we design site infrastructures to provide an appropriate level of reliability, while keeping an appropriate balance between cost and the risks and consequences of site failure. What is "appropriate" will depend on your specific situation – your budget, peace of mind, and level of aversion to public relations problems all enter into the appropriateness equation.

Infrastructure Elements

Most external sites are designed for a particular purpose, making use of particular components, software applications, and size and design criteria. But no matter what the final overall design, a site usually contains some (or all) of the following types of components:

Computing Systems: They come in different sizes and different complexities, and they run different operating systems. But they're all intended to run some form of software that performs a function or provides a service.

Storage Disks: Disks, tape, optical, etc.

Redundancy is the primary tool for guarding against server hardware failure.

Networking: Usually some form of hub or switch, host network interfaces, and an uplink to the Internet or a wide area network.

Firewall or Filtering Gateway Router: A barrier of some kind to limit what forms of access to the site are allowed.

Load Balancing: Larger Web sites almost always utilize some form of load balancing to share the service load across multiple servers. Load balancing can be provided through software, DNS-based mechanisms, or by load-balancing hardware.

Support Elements: The power bars, uninterruptible power supplies, console servers, modems, monitoring and environmental devices, etc.

Applying the Concepts

Redundancy

Redundancy is the primary tool for guarding against server hardware failure. The idea is simple – if you've got two (or more) parts performing a single function, things will (probably) keep working if one of the parts breaks.

It wasn't very long ago when redundant parts were relatively expensive, and redundancy was used only in high-end sites. Nowadays, with commodity components, redundant components should be included in just about every server.

The most common application of redundancy is with disk drives, where two or more drives are configured for mirroring (RAID 1) or parity striping (RAID 5). RAID configurations allow systems to keep functioning with no data loss even if a disk fails. With disks as cheap as they are these days, and with RAID software included with just about every operating system, you should almost always configure your servers to use some form of RAID configuration.

Other components that are often configured for redundancy are: power supplies, CPUs – you didn't think dual processor systems were just for added speed, did you? – memory boards, and network interfaces. Most servers these days can be easily configured with redundant components for most of the critical parts.

Redundancy is a great technique and can be cost-effectively applied in just about every situation, regardless of your application or environment.

Replication

Replication is used for two purposes: resilience of the service/system in the face of server failure and scalability of the service. In external sites, the most common example of replication is in the use of multiple "identical" Web servers, with the load shared across an entire "server farm."

If your application or service can be built (or configured) to work across multiple services, you'll end up with a much more reliable and scalable system; a server farm of 10 (or 100) Web servers can cope much more effectively with a server failure (due to OS bugs, catastrophic hardware failures, etc.) than a single monolithic server, and provides an obvious method for upgrading overall system capacity.

However, replication can't be achieved simply by plugging in another server (obviously); your application and system need to have been designed and/or configured to be able to make use of replicated servers. For example, multiple Web servers won't do much good if your Web site's address is assigned to just one of the boxes.

Server replication can usually only be useful in concert with some external help. Replication is sometimes available in services through the application or its interfaces. For example, some "middleware" software programs provide Web server plug-ins or other interfaces that select one server in a replicated pool from a list of currently available servers (e.g., Apple's WebObjects software).

Don't make your Web server, application server, and database server all the same box.

In most cases, however, some form of external load balancing is used. The most common examples of external load balancing are "round-robin" DNS configurations or load-balancing hardware devices.

Round robin DNS is the simplest (and cheapest) method of load balancing, but it has a number of drawbacks. It is implemented by defining a service name with a number of IP addresses, one for each server in your server farm. DNS servers will provide the list of IP addresses in response to lookups, but will rotate through the list of addresses, putting each different address at the head of the list in turn. (At the time of writing, `nslookup www.microsoft.com` provides an example of the use of round-robin DNS.) The primary drawback of round-robin DNS is in times of server failure, when some percentage of users will be given the IP address of the failed server as the first one to try, and will either fail to connect or get a delayed response while the initial connection to the failed server times out. It also performs poorly when the server maintains some form of "session state" between connections; if you connect to a different server next time, it may not have the correct context to continue your service.

These problems can be addressed by using intelligent load-balancing appliances, such as those made by Alteon, F5 Labs, and others. These are devices that (typically) look like an Ethernet switch, but which "re-write" IP addresses in packets passing through them to load-balance traffic across a group of servers. They do this by performing periodic "health checks" on the servers in their pool for availability, response time, etc., and by maintaining some form of state table that matches the two ends of a load-balanced connection. A real discussion of load balancing is beyond the scope of this article, but suffice it to say that these devices can be quite sophisticated, powerful, and very effective in keeping services running and available.

One final aspect of replication that needs to be considered is that of data or content distribution, i.e., keeping everything consistent across an entire server farm. This can be a complicated problem, with lots of revision control and timing issues, but can usually be handled effectively in software for services that use relatively static data, using tools such as `rdist`, `rsync`, `CVS`, or various commercial software packages. Replicating database servers is usually a much more complicated exercise, especially when database updates are driven by external sources (such as Web site customers submitting orders). Up to a certain size, database server replication can be handled by clustering or database replication by the database software itself, both of which can be quite intricate. But beyond that, things can get very complex, with large amounts of custom software keeping things consistent.

The underlying and as yet unspoken message is this: plan ahead for replicability and understand how the components of your application will interact when there's more than one of each.

Separation of Functions

As mentioned above, this is the "don't put all your eggs in one basket" concept. By this, I mean use separate servers for each service or function – don't make your Web server, application server, and database server all the same box. There are several reasons why

A reliable system is one that is designed to be effectively maintainable, resilient in the face of change or failure, and quickly recoverable should something go wrong.

this is a good idea. The most obvious reason is that when a server fails, you only have one service impacted, rather than two (or more!). It also makes troubleshooting much easier if you don't have to worry about the possible interactions of two competing applications on the same server. Finally, separating functions makes capacity planning and upgrades much simpler (in most cases). Given the wide range of server capacities (and prices) these days, it's almost always possible to cost-effectively separate your services onto separate servers.

Reliability

Reliability in this context is a grab bag of things to consider in the context of the overall system design, implementation, and ongoing operation.

A reliable system is one that is designed to be effectively maintainable, resilient in the face of change or failure, and quickly recoverable should something go wrong. Reliability encompasses a wide range of "best practices": proper planning, considered design, effective documentation, and consistent execution.

I won't say more than that about reliability here; I will instead refer you to my (upcoming) SAGE booklet, *System and Network Administration for Higher Reliability*.

Accessibility

For external sites, often located where you aren't, accessibility is a much more important issue than it is for a server that's just down the hall (or under your desk).

By "accessibility" I mean the ability to get appropriate access to the components of your site both during normal operation and during times of failure. The most obvious example of "access" is some form of network login (e.g., telnet, SSH), but access also includes "out of band" access for when your firewall or router is confused, or when your network connectivity is absent; console access when a system is at the "boot" prompt or has crashed; access to the "big red" (power) switch when necessary; and access to the hardware when it's necessary to replace a failed component. Let's look at each in turn.

Network Login: This is most commonly "just software." Most UNIX folks are familiar with SSH these days (and if you aren't, you absolutely should be), which provides secure remote logins, file copies, and command execution (among other interesting things). Other approaches to network login include plain old telnet, and "remote control" applications such as VNC or PCAnywhere. These applications allow access for normal everyday maintenance activities, as well as emergency repair in situations where most components are still functioning.

Out of Band: When your network access is broken, having an alternate path into the internal network of your external site can be a blessing. Anyone who has ever changed the configuration of a remote router or firewall has likely (or perhaps, hopefully) understood the possible complications of a "slip of the finger."

Out-of-band access is most commonly implemented with some sort of dialup connection, usually a modem on a serial port of a computer or router, but it can also be implemented in other, more complicated ways. Make sure that when you're planning your site, you consider what will happen when your primary connectivity fails.

Console Access Network: login works just fine as long as a system is running normally. When it crashes and is sitting in single-user mode, or it's at the boot prompt, or the BIOS is waiting with the message "keyboard missing, press F1 to continue," you'll need some form of console access. Some servers and devices have serial console ports, which can be connected to a modem, terminal server, or the serial port of another device. Other servers require a keyboard and display in order to deal with some problems. The latter typically require a remotely accessible KVM (keyboard, video, mouse) switch, or someone onsite to deal with the problem.

Power Control: A number of companies make remotely controllable power bars that you can connect to via serial connection, telnet, SNMP, or Web browser, and that allow you to turn devices on and off remotely when they get completely wedged. Some of these devices even have environmental monitoring built in, so you'll be able to tell when your air-conditioning has failed (or your server is on fire). Great tools, very useful, and usually well worth the investment.

Remote Hands: Depending on the problem, and where your system is located, having the (pre-arranged) ability to call someone at the remote site and tell them which button to push or cable to swap can save a lot of aggravation. Many co-location facilities offer this service, as do a number of third-party service companies. The key here is "pre-arranged" — you've got to make sure that everything is in place before you need it. And that's not just having someone's name in your phone list; it's having an agreement, effective contact information, and proper documentation of both the procedures and the configuration of your site.

When things are broken, it's no time to start wondering how your site was built and configured, and what options were chosen when.

Recovery
When things are broken, it's no time to start wondering how your site was built and configured, and what options were chosen when.

There are two primary considerations to recovery: having a well-defined (and well-rehearsed) process for restoring the system to a fully operational state and the ability to do just that on a timely basis. Consider the following: using standard mechanisms for configuring servers, such as Sun's "jumpstart" mechanism, to ensure a fast and consistent recovery process; having effective vendor support contracts in place, so that repairs can be completed within an appropriate time frame; keeping your documentation complete and up-to-date, with rack elevations, network diagrams, copies of configuration files, disk partition information, equipment model and serial numbers, and anything else you can think of; and having an effective backup (and restore) process.

Security
Security is a topic that is far too involved to be covered effectively here, so I will say only this: make sure that you have both an effective security policy and an effective security implementation that together provide the appropriate balance between the risk of a security exposure and the overall effectiveness of your system.

Closing
Designing an effective external site infrastructure can be a very complex process of trying to balance conflicting needs and priorities while delivering a working and cost-effective system. The discussion here has not covered every possible alternative or every possible problem area, but I hope that it has given you some things to think about and will help you implement systems that don't keep you up at night.