EHP System Architecture

Introduction

The web server physical architecture significantly impacts the overall performance of the Earthquake Hazards Program website (http://earthquake.usgs.gov/) in terms of both reliability and maintainability. This document will present a plan for re-architecting the website and the systems that obtain and post content.

Motivation

The current master/slave web architecture is proving inadequate for current needs, much less new requirements already being incorporated into the system. Even with the addition of a backup master server it is not sufficiently fault tolerant to meet National Earthquake Hazards Reduction Program (NEHRP) requirements. In addition, it suffers from performance bottlenecks both in processing real time products as they arrive and in synchronizing information to the slave web servers.

The following is the result of many months of discussion about basic requirements (robustness, consistency, fail over, extensibility, scalability, etc.) and architectural elements that might be used to meet these requirements. Although the proposed solution is complex, it seems to be the simplest architecture that meets all the requirements.

Overview

There are at least two geographically separated nodes in the architecture. Any one node is fully capable of handling all website requests. Each node operates independently of other nodes, and no fail-over is required. In addition, each node has internal redundancy to improve the fault tolerance of the node itself.

Within each node there is a three tier architecture. The first tier is a cache tier. The second tier is an application tier. The third tier is a master tier. Machines within a given tier all communicate using distinct replication protocols (see each tier's detailed section below for more information on this). Communication between tiers works over network protocols and uses a load-balancer to ensure successful messaging.

Cache Tier

The cache tier accepts connections from the public through a Content Delivery Network (CDN). When possible, this tier serves cached responses to any request. If a cached response is not found in this tier, the request is sent to the application tier through a load-balancer. The application tier processes the request and generates a response. The response is then cached, honoring HTTP expires headers, in the cache tier such that subsequent requests for the same content are available in the cache.

Template files, like the banner image, change infrequently. The caching layer caches and handles requests for this infrequently changed, but frequently requested, content. This frees the application layer to serve dynamic requests that change frequently, such as real time earthquake pages.

The cache tier requires at least two physical servers for redundancy purposes. Within this tier servers are set up as "siblings" within their caching architecture. With this set up the siblings can "cache-fill" their cache from any sibling cache in the architecture. In order to make the real time content more timely, real time product updates need to invalidate the cache for their products.

Example 1

> Assume CacheA and CacheB are set up as siblings in a caching architecture. Each server's cache is currently empty. Next assume CacheA receives a request for `/some/page.php`. CacheA recognizes its cache is empty and proxies the request to the application tier. The response from the application tier is cached in CacheA's cache for subsequent requests. Next assume CacheB receives a request for `/some/page.php`. CacheB recognizes its cache is still empty, but also recognizes CacheA (a sibling) still has a valid cached response. Rather than proxying the request back to the application tier, CacheB now "cache-fills" from CacheA and serves the now-cached response for `/some/page.php`. Subsequent requests to either CacheA or CacheB for `/some/page.php` will now be served from their respective cache until the content expires or is invalidated.

Application Tier

The application tier is accessed via requests from the cache tier. These requests may be for static or dynamic content. Static content is any content that *will not* vary by request. This includes HTML, CSS, Images, JS, XML, or similar types of files. Dynamic content is any content that *may* vary by request. This includes PHP, PERL, Python, Coldfusion, JSP, or similar types of files. Note that a PHP *file* that serves HTML *content* is considered dynamic content. Any request that accesses the database (read or write) is a request for dynamic content.

If the incoming request to this tier must read from the database it connects to a local, read-only replicated copy of the database. If the incoming request to this tier must write to the database it connects to the remote, master tier database. These writes immediately replicate back down to the local, read-only replicated copy of the database.

Servers within the application tier are configured in a clustered environment. All application servers access a shared file system. The shared file system is divided into three sections; real time, static, and user files. Real time files are generated in response to an event based on information sent to the master tier servers. These generated files are then replicated to the application tier for public consumption. Static files are developed on the development server and sent to the master tier servers as part of the daily build of the website. These developed

files are then replicated to the application tier for public consumption. User files are generated in response to a user request and are created on the application tier itself. These files may be custom images, data, or similar types of files.

Example 2

A user submits a request from any of our many web applications. This process is intensive, and several files are generated in response to this request. Each of these files is written to the shared User Files section of the cluster file system, and links to these files are sent to the user. When a user requests one of these files, the request might be processed by a different application server which also has access to the shared User Files section of the cluster file system to serve the content.

Master Tier

The master tier is isolated from public requests. It is responsible for real time processing, and file system and database replication. Real time processes run independently on each master server and replicate real time file system content to the real time section of the application tier cluster file system.

The master copy of all database content resides on the master servers. Each master server is configured to replicate user database content from each other in a multi master configuration. Real time and static database content are not replicated between masters. Real time content is updated independently on each master server, and static content is updated during the daily build.

All application servers use one of the master servers as a replication master. A process on each application server monitors this replication to make sure it is still active, and automatically fails over if needed.

Similarly, the master copy of real time and static file system content resides on the master servers. Each master server replicates real-time and static file system content it receives to the application tier cluster file system. User file content is never seen on the master tier servers.

Example 3

A new version of “Did You Feel It?” is delivered by the Product Distribution Layer (PDL). Both master servers run the generalized indexing process, update the “realtime” database, and replicate product files to the application tier’s shared file system.

Example 4

A request for the current “Did You Feel It?” page is received by Cache Server1, and has not previously been cached. The load-balancer sends it to Application Server2. Application Server2 satisfies the request by fetching information from the “realtime” schema and files, allowing it to render the requested html, which is passed back to Cache Server1. Note that the application server does not do any real time processing of any “Did You Feel It?” files, it only serves

requests for this data. Indexing and replication are handled by the master servers.

Network Considerations

The caching tier receives requests from the public, and fulfils those requests either from data in its cache or by making HTTP requests to the application tier. The application tier receives HTTP requests only from the caching tier, and database and file system replication from the master tier. The master tier receives data from internal USGS systems.

The architecture relies on systems in each node being in close network proximity to each other, ideally on the same switch. All communication between tiers in a single node could occur in a private network inside a NAT. Master tier servers would have one network interface accessible from internal USGS space to receive content, and a second network interface connected to the private network. Application servers would only connect to the private network. Cache tier servers would have one network interface accessible from the internet, and a second network interface connected to the private network.

USGS network security requirements may necessitate all systems in a production node being located in the DMZ. This is not a problem, although all systems in a single node should be connected to the same switch.

Implementation Plan

Each node in this architecture requires seven (7) servers plus one (1) switch. The general plan for migrating to this new architecture is to create one node entirely from new hardware. Once this node is in place we can use it to actively serve the public website while we use existing machines to create the second, geographically-separate node.

For the first node we currently have two (2) new machines, one (1) formatted new machine, and two (2) new machines to be purchased by the IT department. This leaves us in need of two (2) new servers and one (1) new switch for this first node. Initial estimated cost for the hardware to be purchased comes to ~$5,000 per server and ~$1,500 per switch. Accounting for incidentals etc... we estimate an upfront hardware cost of no more than ~$15,000 for the first node.

Once the first node is functional, it is fully capable of handling all traffic for the website. At this point we can start re-purposing existing machines to create a second node. This will require shipping some machines to a new geographic location. We plan to re-use six (6) existing machines for this second node. This leaves us in need of one (1) new server and one (1) new switch for the second node. We estimate an upfront hardware cost of no more than ~$10,000 for the second node.

Once all servers are purchased and available we estimate roughly one development iteration (3 weeks) is required to implement each node. A development iteration requires all involved persons be available and able to work on the project

Relationship to NEHRP Public Architecture

The web architecture described above addresses the tightly integrated system required to meet current and projected needs for NEHRP product distribution via the web. Although there is no software or communications requirement for the various architectural elements to be physically close together, co-locating elements will make it easier and more cost effective to install, operate, and maintain the underlying equipment. Other elements include:

1. Web architecture extensions:
    a. SSL servers (supporting secure user input for DYFI?, ENS, IRServer, etc.)
    b. A storage server (for large volume research products accessible through the web servers)
2. NEHRP architectural elements:
    a. ENS (Earthquake (email) Notification Service)
    b. CISN Display servers (notification distribution to CISN Display clients)
    c. Earthquake Catalog servers (providing searchable access to historical and current earthquake products)
    d. EIDS public hubs (acquisition and delivery of primary earthquake messages)
    e. PDL public hubs (acquisition and delivery of large and/or binary earthquake products)

EHP Web Architecture

October 14, 2010