EHP System Architecture

Table of Contents

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-designing the website architecture 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 topics were all considered, and used as a general road map during the creation of this proposal.

- **Archives**
  - Large sets of historic data make rsyncs slow.
  - Accessed infrequently but are important to scientists and engineers.

- **Interactive applications**
  - Online web apps that require interaction with users are unable to be cached.
  - Use too much CPU when web traffic is high.

- **Databases**
  - Accessed by website for dynamic pages.
  - Hard to synchronize on multiple servers.
  - Should be separate from web server.

- **Real-time web pages**
  - Most important product.
  - High web traffic after a significant earthquake.

- **Dynamic webpages**
  - Access database (interactively) to create webpage.

- **Internal webpages**
  - Not for public consumption.

- **Static webpages**
  - Don't change very often.
  - Never high traffic load.
-
- **SSL (secured) webpages**
  - on separate virtual (and/or physical) server?

- **SVN**
  - Version control system.

- **PDL/Indexer - product distribution layer**
  - Product indexing and distribution, QDM replacement.

The following is the result of many months of discussion about basic requirements and architectural elements that might be used to meet these requirements.

Overview

In the proposed architecture 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 (multiple machines) to improve the fault tolerance of the node itself.

Within each node there is a three tier architecture:

1. Cache tier
2. Application tier
3. Master tier

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.

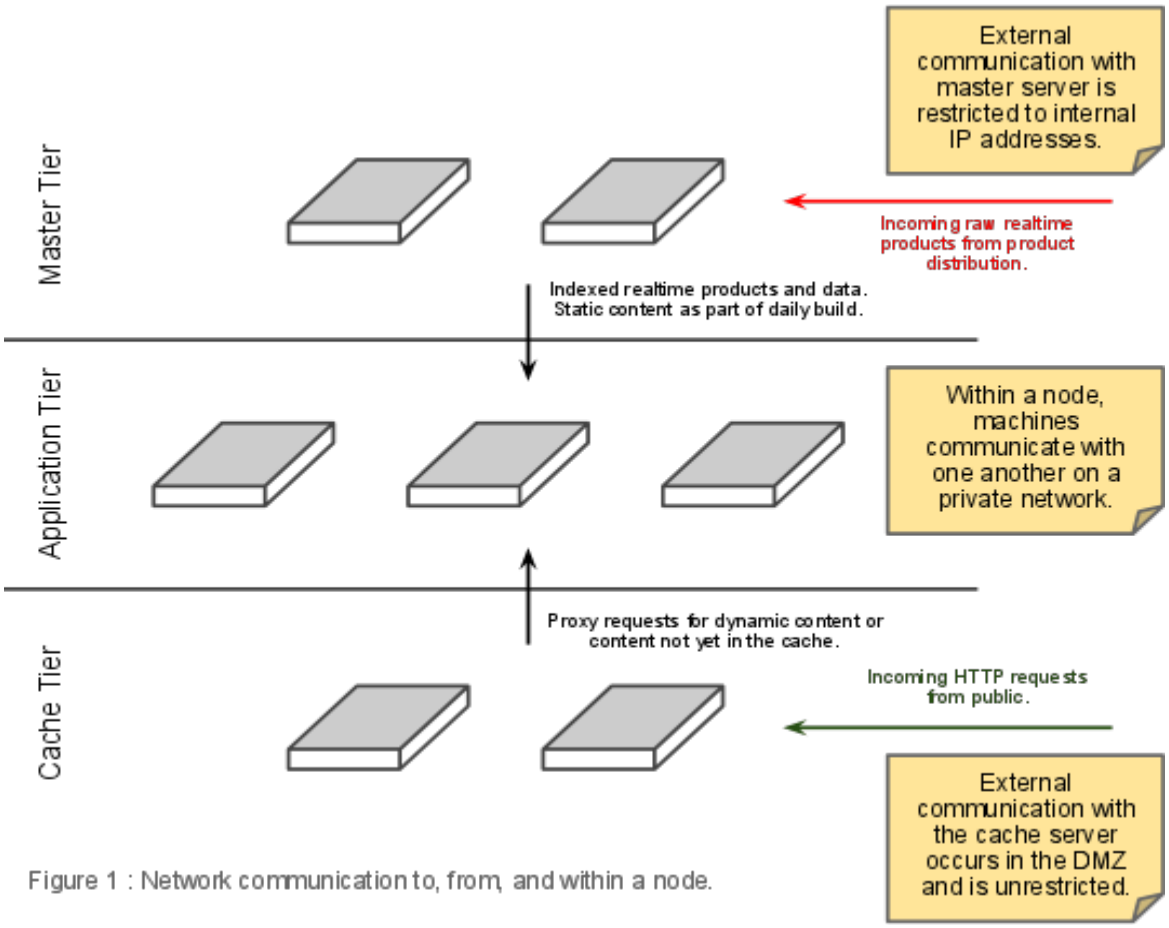


Figure 1 : Network communication to, from, and within a node.

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 occurs on a private network. 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.

1. Cache Tier

The cache tier accepts connections from the public through a Content Delivery Network (CDN), currently implemented by Level3. 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 tier 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”. Siblings can “cache-fill” from any other 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

A. 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/file.php`. CacheA recognizes its cache is empty as checks its sibling’s cache. Since CacheB is empty as well, CacheA then proxies the request to the application tier.

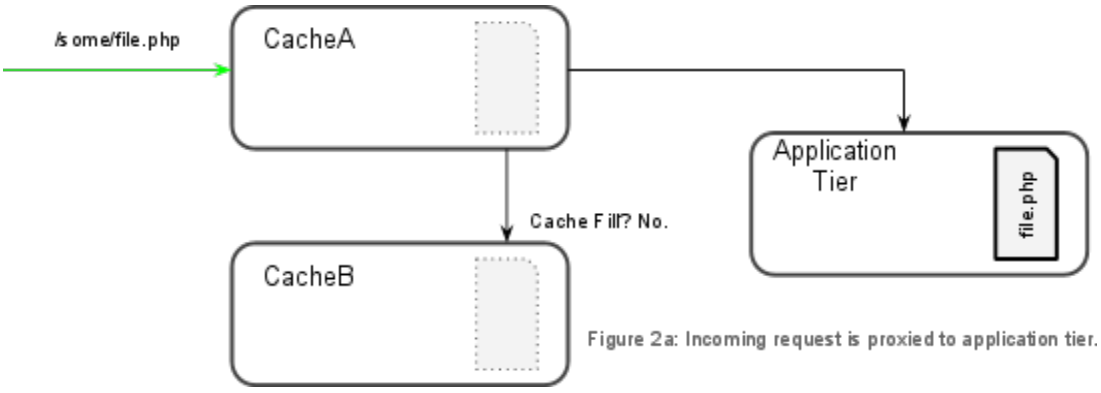


Figure 2a: Incoming request is proxied to application tier.

B. The response from the application tier is cached in CacheA's cache for subsequent requests. CacheA then forwards the response back to the user.

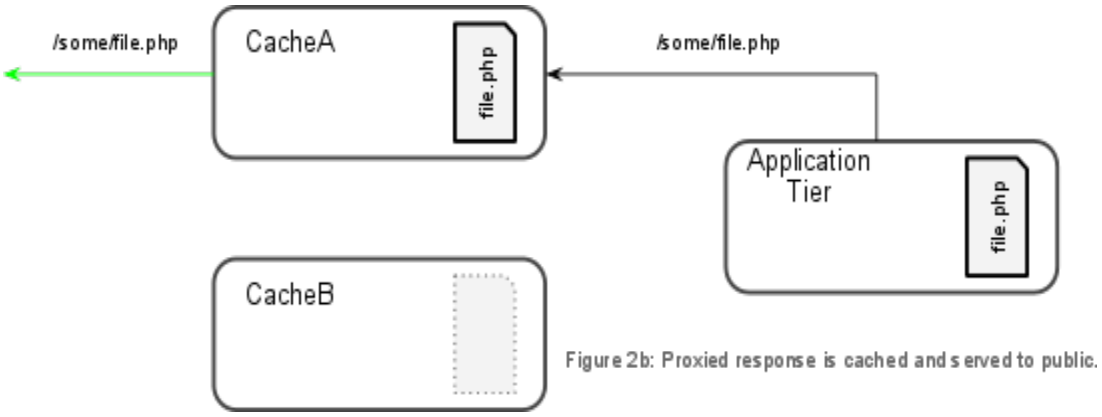
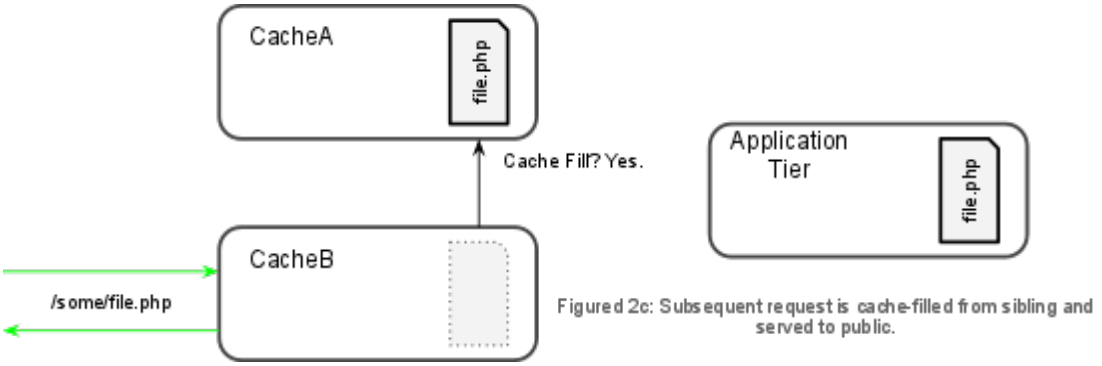


Figure 2b: Proxied response is cached and served to public.

C. Next assume CacheB receives a request for `/some/file.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/file.php`.



Figured 2c: Subsequent request is cache-filled from sibling and served to public.

D. Subsequent requests to either CacheA or CacheB for `/some/file.php` will now be served from their respective cache until the content expires or is invalidated.

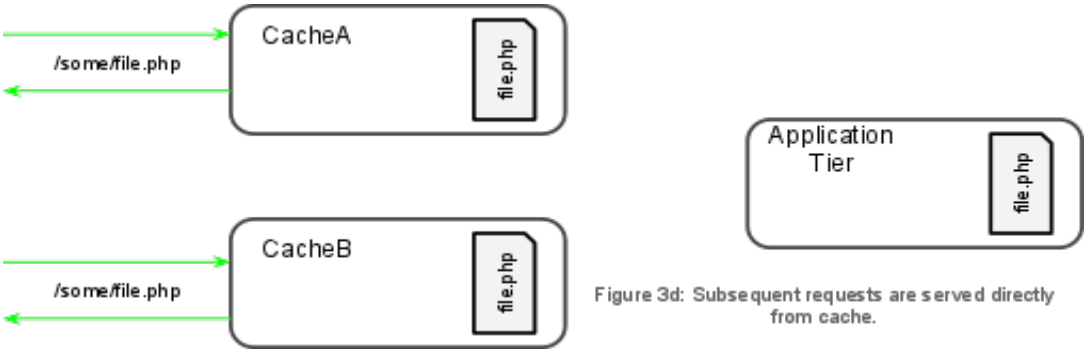


Figure 3d: Subsequent requests are served directly from cache.

2. 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.

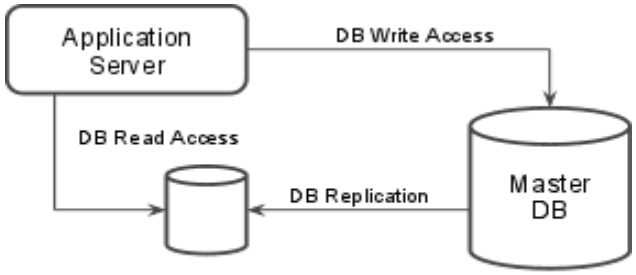


Figure 3: Read/write access from the application tier.

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, tested on the staging server, and sent to the master tier servers as part of the daily build of the website. These static 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.

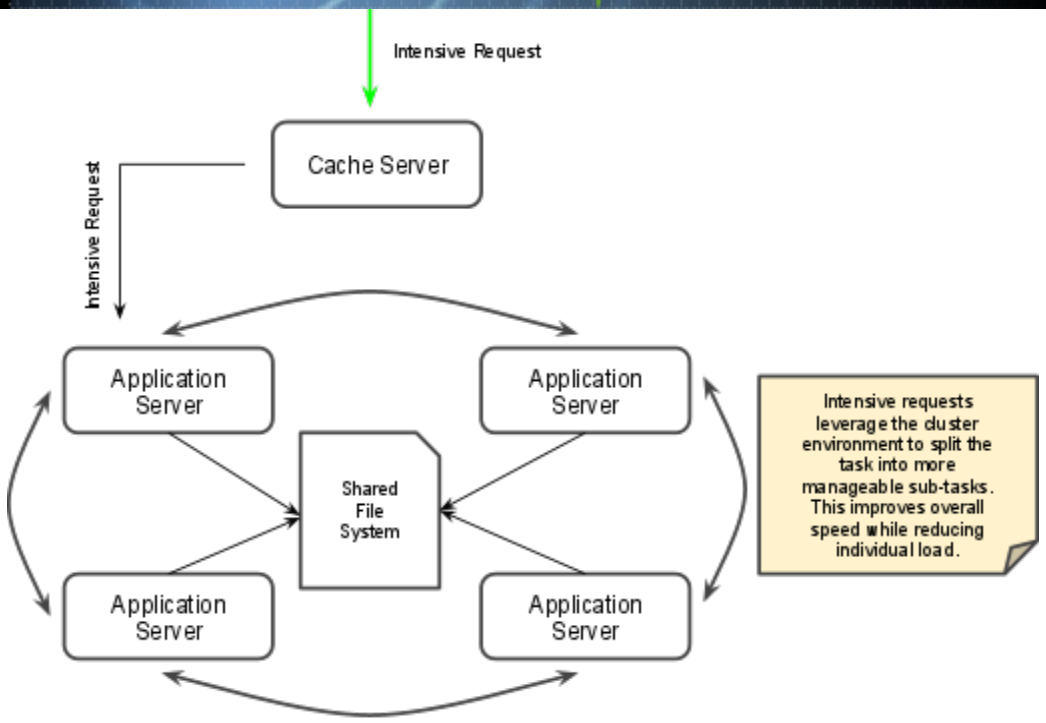


Figure 4: Dynamic requests for process-intensive content.

3. 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.

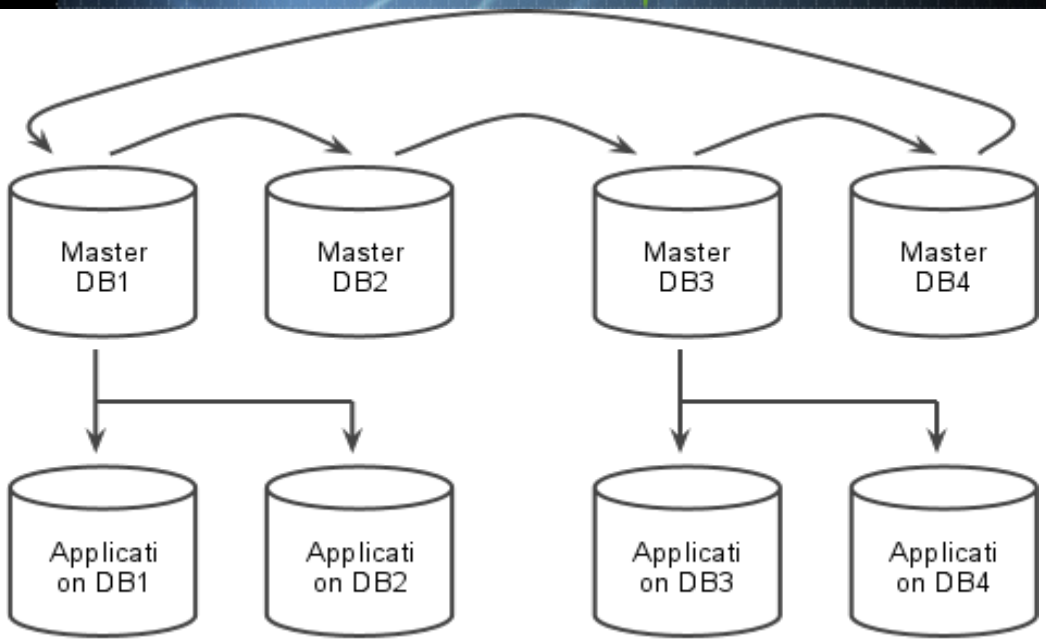


Figure 5: Multi-Master, Ring Replication

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.

USGS network security requirements necessitate that all systems in a production node must be located in the DMZ. This is not a problem since border firewalls can be configured to allow access to the master servers only from selected USGS addresses and public access to cache servers.

The Big Picture

Figure 6 (below) brings together all aspects of what has thus far been proposed. It attempts to display as much information as possible without becoming overly complex. The following information can be used as a reference when viewing Figure 6:

- Blue box: Geographic boundary.
- White box: Individual physical server.
- Grey cylinder: Database.
- Small grey rectangular polyhedron: Logical software.
- Arrows: Indicate direction of data and/or request flow.
    - Straight lines: File system data/requests.
    - Curved lines: Database data/requests.
    - Red lines: I/O between internal servers and the node.
    - Black lines: I/O within a single node.
    - Green lines: I/O with the general public or DMZ.
    - Solid lines: Both static and real-time content.
    - Dashed lines: Only static content.
    - Dotted lines: User-generated content.
- Numbered Labels: Reference numbered bullet points below the figure. They do not indicate a chronological (or any other) ordering of events.

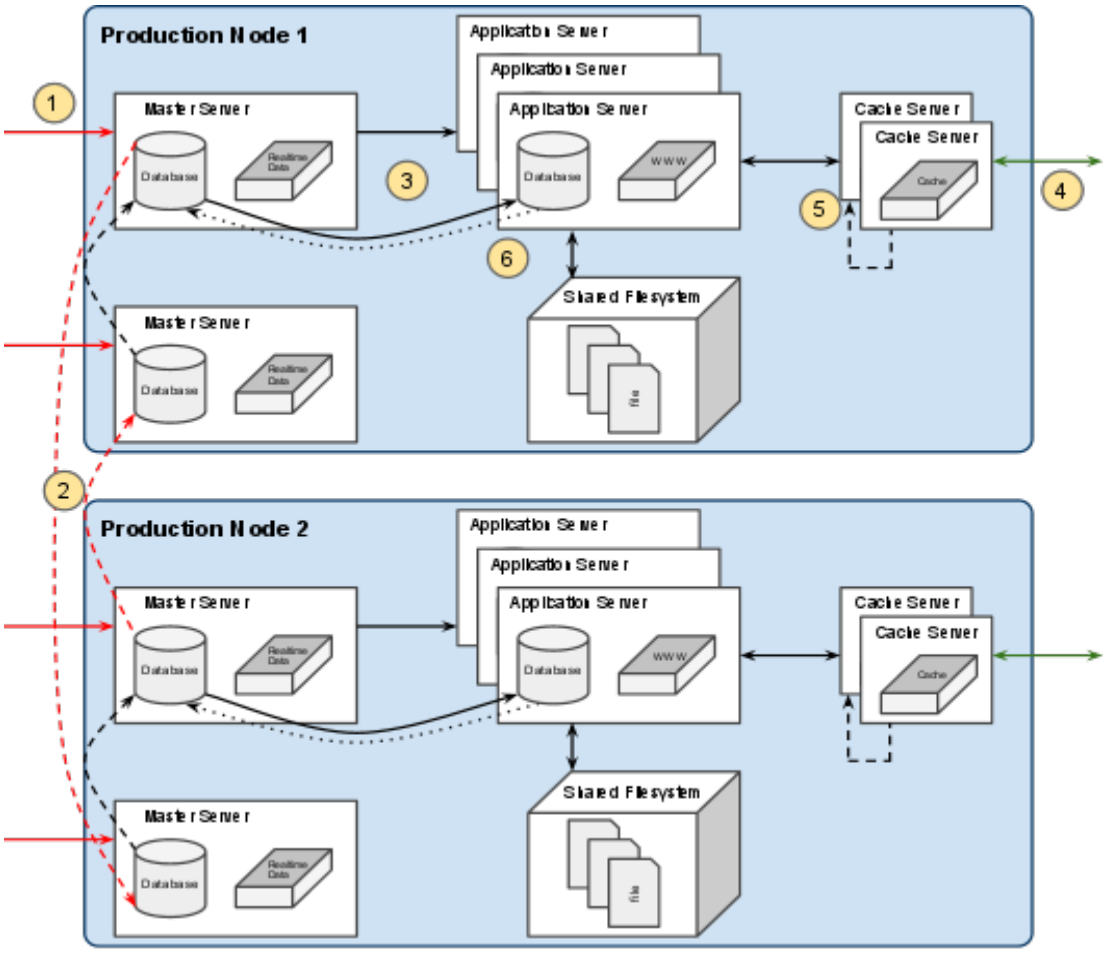


Figure 6: The big picture.

1. Incoming data is received on the master servers. Real time data come from product distribution as it is available while static data comes from the staging web server as part of the daily build. Database content is provided through both mechanisms as well.
2. Static database content is replicated between each of the master server databases to ensure the database stays in sync.
3. All database and generated file system content are replicated from the master tier to the application tier servers.
4. Website visitors make a request for one of the web pages. After going through Level 3, the request hits a server in the cache tier (if not served by Level 3).
5. Requests for content are first checked against the server cache, and then against any sibling server cache. If the content is not found in any cache, the request is proxied back to the application tier.
6. Application tier services requests for content. It generates files as needed, reads from and writes to the database and reads/writes files from/to the shared file system to generate the requested response.

Implementation Plan

We have two general implementation options for migrating to the proposed architecture. The first option utilizes all new hardware to make the transition. This approach is more costly but protects the website from any potential downtime. The second option re-uses as much existing hardware as possible. The second approach saves money but leaves us temporarily vulnerable if catastrophe should strike during the transition.

Each node in the proposed architecture requires seven servers plus one switch. The generalized plan for migrating to this new architecture is to create one node entirely from new hardware. We currently have three available machines to use for this node so we would need to purchase four new machines and one new switch. **Each server costs roughly $5,000 and each switch costs roughly $1,500.** Accounting for incidentals, this brings the estimated cost for the first node to roughly $25,000.

Implementation of the second node can proceed in one of two ways. The first approach is to re-use existing hardware to build the second node. The second approach involves purchasing all new hardware. The general trade off of these two approaches is the former is more cost-effective and the latter is more robust during transition.

Option 1: $25K (Node 1) + $10K (Node 2) = $35K

Since a single node is fully capable of handling all traffic for the earthquake website we can re-purpose existing hardware to build the second node. We have six machines in the current architecture (EHPMaster, EHPBackup, EHP1, EHP2, EHP3, EHP4) that can be re-purposed which leaves us in need of purchasing one new server and one new switch. Accounting for incidentals, this brings the estimated cost for the second node to roughly $10,000.

Each node has internal redundancies that almost eliminate the chance of a single-point of failure. However, since the proposed architecture calls for all machines in a given node to be co-located at a single geographic site, this does leave the node vulnerable to a single-point of failure with regard to a geographic outage. Geographic outages could occur due to external network failures, power failures, or catastrophic events.

Since we will have to ship several machines between states, we will operate under this exposed state for no less than six weeks. **A rough outline of effort is as follows:**

Week 1: Machines are shipped to single geographic location.
Week 2: Local SA wipes each machine and re-installs base operating system.
Week 3: Local SA installs each machine into network, configures and tests firewalls.
Week 4: Web Administrator installs and configures web software. (Apache, PHP, MySQL etc...)
Week 5: Web team creates copy of website on new node.
Week 6: Web team tests, configures, tweaks new node to prepare for moving into production.

Option 2: $25K (Node 1) + $40K (Node 2) = $65K

To avoid leaving the website vulnerable to a geographic single-point of failure we must purchase entirely new hardware for the second node. **The process of implementing the second node is relatively the same as under option 1, however during this time we will have full geographic redundancy.** Accounting for incidentals, the estimated cost to build the second node entirely from scratch is roughly $40,000.

Option 3: Additional $100K

There are some other topics to consider when setting up the new architecture as well. The proposal calls for the servers in the application tier to be configured in a cluster environment with a shared file system. **An improvement on this system is to use an external SAN for this file system.** This is a very viable option but requires an additional (roughly) $50,000 per node or $100,000 overall.

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 other NEHRP 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. ENS (Earthquake (email) Notification Service)
2. CISN Display servers (notification distribution to CISN Display clients)
3. Earthquake Catalog servers (providing searchable access to historical and current earthquake products)

4. EIDS public hubs (acquisition and delivery of primary earthquake messages)
5. PDL public hubs (acquisition and delivery of large and/or binary earthquake products)