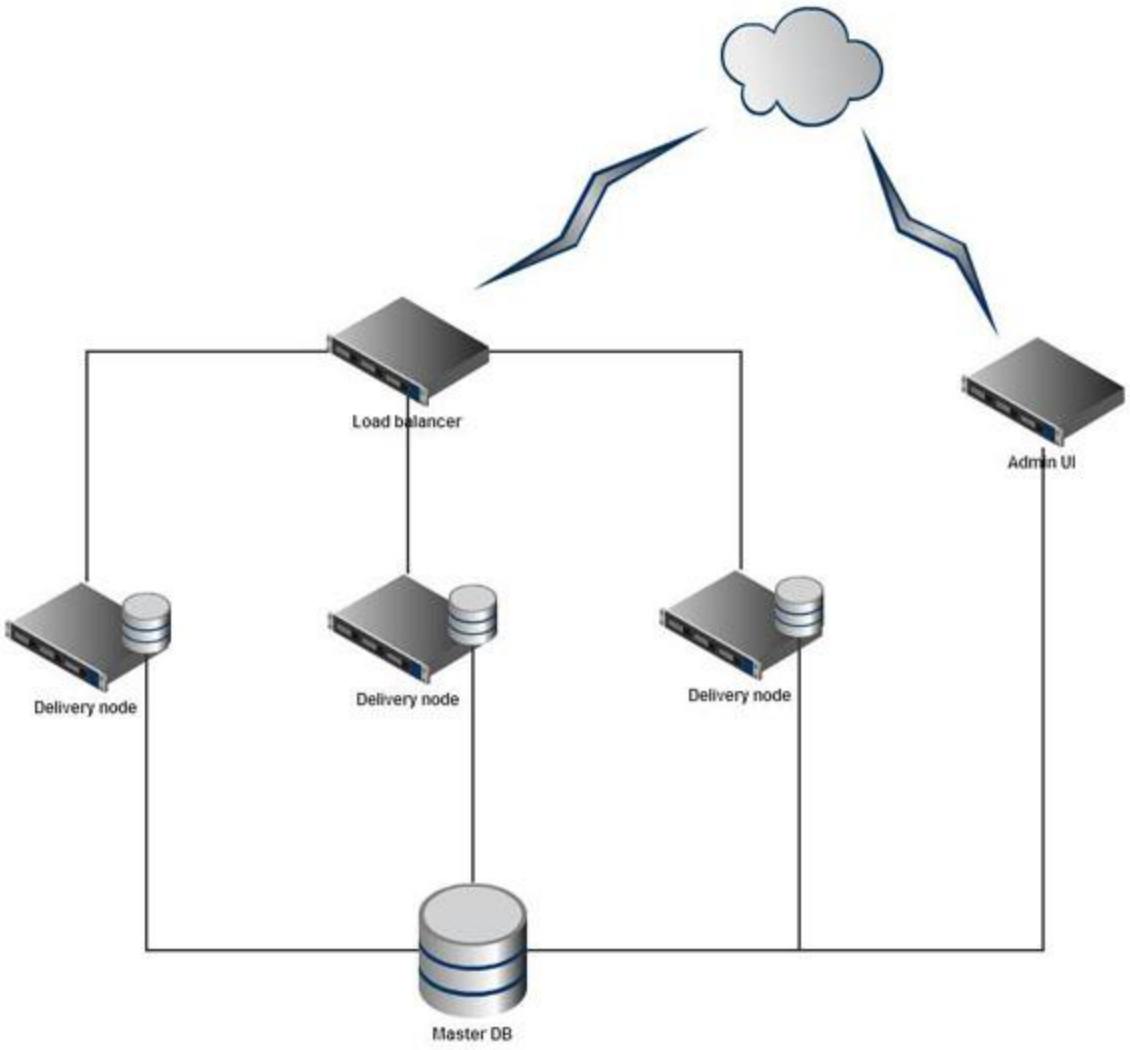
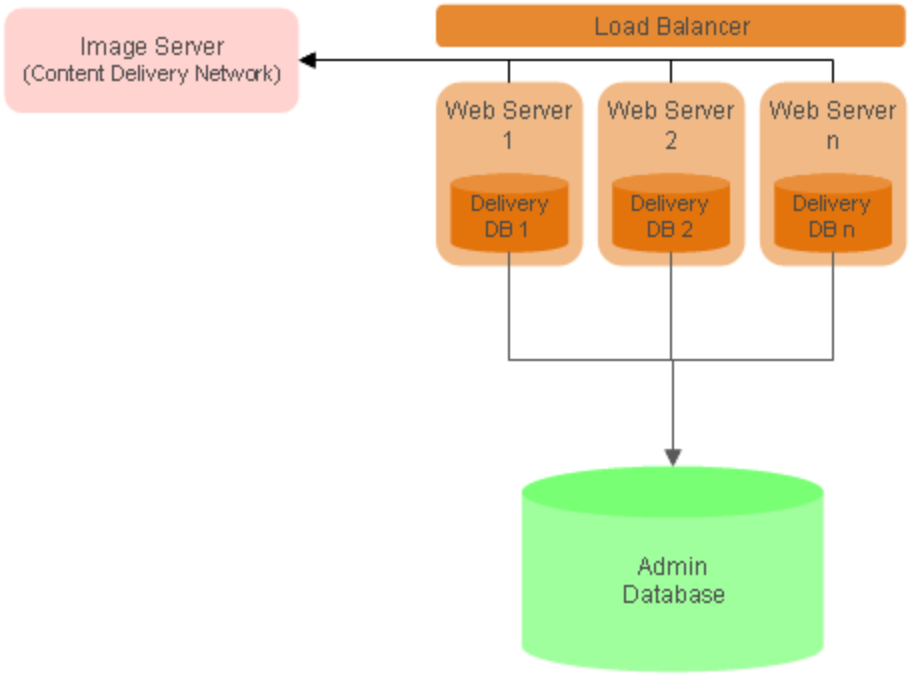
openx links

http://www.openx.com/docs/2.8/adminguide/Distributed%20statistics
http://www.openx.com/docs/whitepapers/distributed-statistics
http://www.openxtips.com/2009/09/tip-28-simple-openx-scaling/
http://www.sherin.co.in/openxcluster121/
http://www.sherin.co.in/openxhandbook/
http://blog.openx.org/12/faster-page-load-times-happier-users/
http://www.openxtips.com/2009/07/tip-20-protect-your-site-from-openx-hangs/
http://www.openxtips.com/2009/08/tip-26-run-the-maintenance-script-manually/
http://www.openx.com/docs/2.8/adminguide/Running-maintenance
http://www.openx.com/docs/2.8/adminguide/Managing%20performance
https://developer.openx.org/wiki/pages/listpages-dirview.action?key=COMM
http://blog.openx.org/10/serving-billions-of-ads-using-openx/
http://www.mikeonads.com/2010/04/04/the-challenge-of-scaling-an-adserver/

Moreover, it can easily be divided into three main functional parts: Web-based advertising and statistics management, ad engine (deciding to deliver ads and logging the data), and the engine to process statistics and prioritize campaigns.


Load balancer
Admin UI
Delivery node
Delivery node
Delivery node
Master DB

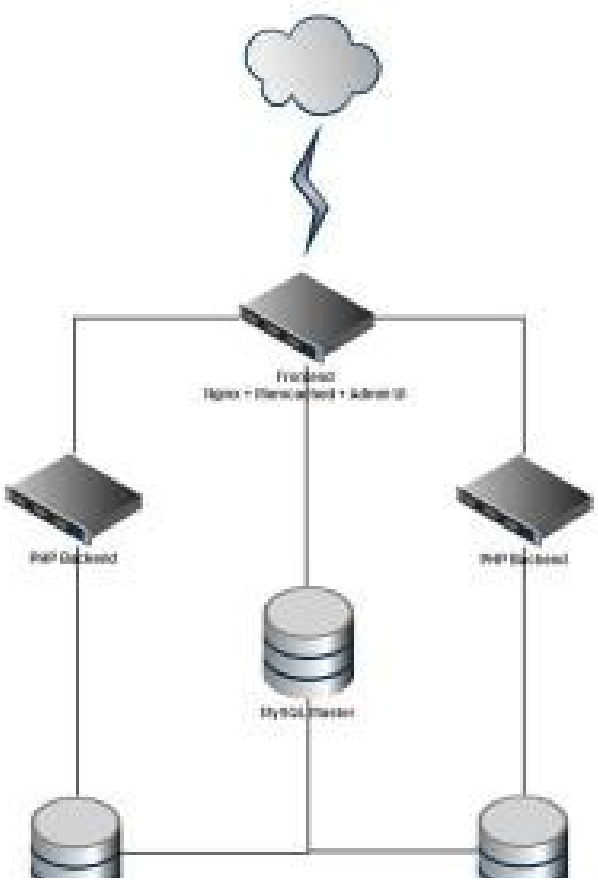


Here, the interface functions of campaign management and processing of raw statistics are implemented on a separate server,while the delivery features are delegated to a set of servers, taking on the main load in the system.

Each delivery node has its own MySQL server configured to replicate data from the master database.

This script runs on the master server and saves statistics to the main database server.
This way you can reduce the load on the main database server and distribute requests across the delivery scripts run on any number of servers.

We slightly modified this scheme, taking into account our current hardware and load tests results. All incoming requests are handled by the frontend, which distributes delivery requests between the PHP-backends and processes administration interface queries.



The engine of statistics and prioritization and the Memcached server used to cache the database information, also run on this server. All these tasks require relatively low computing capacity.

The main load is shared by the PHP backend server.

The database is replicated from the master server to two slave servers, to optimize performance and ensure data integrity.

So, we have improved the previous scheme, taking non-specific functions of the database server from the PHP backend.

Now we can scale the various aspects of the system (database servers, PHP backend servers) independently of each other.

The frontend runs a high-performance Nginx HTTP server which outputs advertising graphics and other static data and proxies requests to execute PHP scripts.

The frontend runs a high-performance Nginx HTTP server which outputs advertising graphics and other static data and proxies requests to execute PHP scripts.

On the backend side, spawn-fcgi + PHP 5.1 are installed. PHP 5.3 has several improvements in terms of performance, in particular, integration of the php-fpm module, but OpenX has not yet finalized its support.

For OpenX installations based on a dedicated high performance database server, InnoDB is advisable.

The following settings were made for Nginx:
see page...

To get an insight on further optimization, consider the main factors on system performance. To do this, you should understand the basic system patterns. When a visitor enters a page with a banner, the following queries are made:

1. Request of promotional materials for each zone or for all zones.
2. Query of advertising graphics (static image). We are not factoring it in, as it makes a minor load on the servers and is easily processed by Nginx.
3. Impression logging.
4. Click logging.

At stage 1, a read request is sent to the database to determine the set of banners available and select the banner to be shown, based on priorities and campaign settings.

At stage 1, a read request is sent to the database to determine the set of banners available and select the banner to be shown, based on priorities and campaign settings. You may cache it, but the cache should be updated periodically to keep track of priorities. At stages 3 and 4, impression counter is incremented in the raw statistics table with the INSERT ON DUPLICATE UPDATE query. This cannot be cached, of course, but still certain techniques may apply.
Here are the limiting factors:

- Need to read from the database
- Need to write to the database
- PHP script performance.

To optimize database reads, provide caching of all information requested from the database.

Therefore, we recommend to immediately enable caching, having not forgotten to increase the maximum number of connections to the Memcached server.

If everything possible is fully cached, requests to the cache may take a substantial share of query processing (about 20%). In view of this, it is recommended to set the following

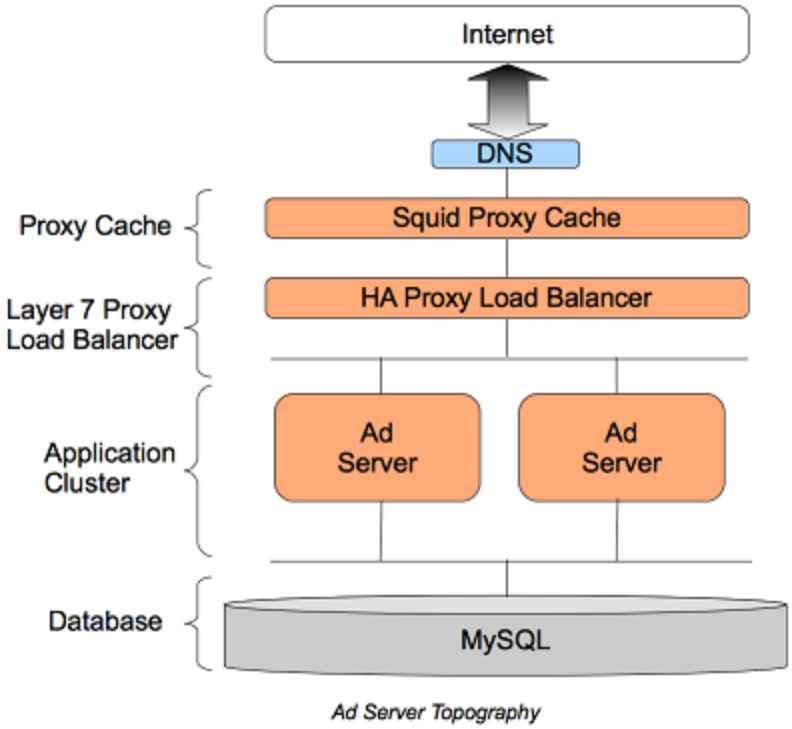
Memcached server parameters: use a binary protocol, use UDP.

The earlier discussed distributed statistics architecture can partially overcome the slowdown due to writing delivery logs.

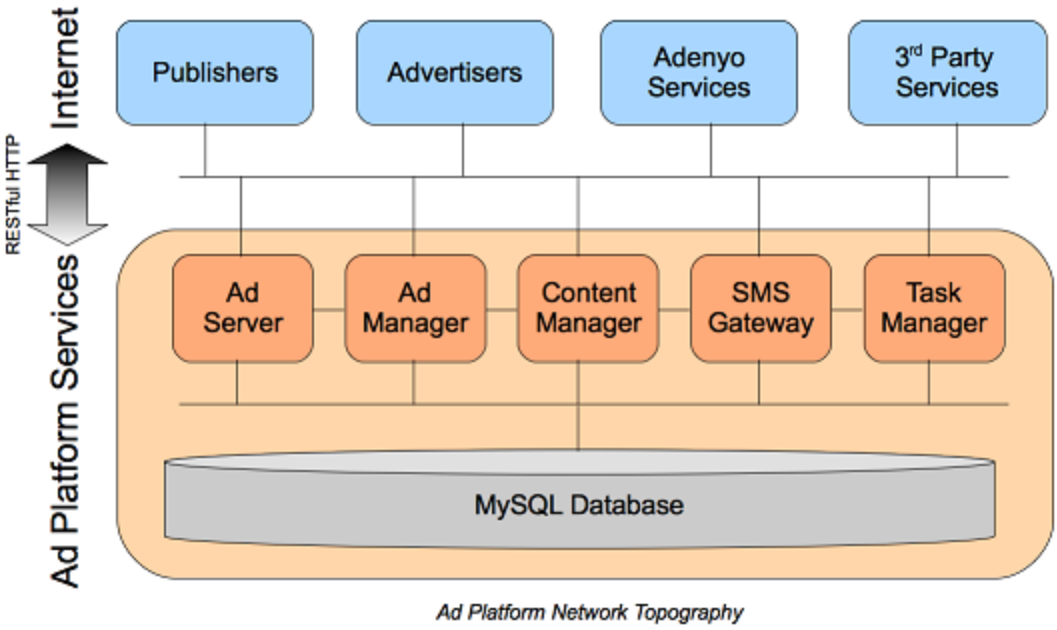
It would be interesting to experiment with storing and reading the data by a statistics script involving a key-value storage or NoSQL servers, currently gaining momentum.

Another approach is focused at reducing the number of queries required to display banners on the page (1, 2), which also improves the page loading speed. In this case, a query returns ad code for all banners shown on a page, rather than for each area separately

AM3



Ad Server Topography



Ad Platform Network Topography

Ad Server

This is our high traffic server for serving small payload ads in XML or some variation of XML. The ad server must be able to access Content Manager and the public via HTTP.

Ad Manager

This is our management application tool for configuring ad networks, campaigns, and ads. The Ad Manager must be able to access Content Manager and the public via HTTP.

Task Manager

This is our "jobs" application responsible for updating various application-specific statuses used on Ad Server and Ad Manager. The application has a small footprint with no public facing HTTP services. Task Manager only reads and writes to the database, and requires no HTTP connection to any services.

Content Manager

This application is like our file server with public facing HTTP service calls to execute read and write operations for putting and getting files. This is mainly used to store our creative assets for our ads, but it can really be used for storing and reading any binary file. Content Manager must be allow HTTP access from Ad Manager, Ad Server, and the public.

Database

We use MySQL 5.x as our relational database configured with a master/slave replication. We need MySQL to have the InnoDB plugin.

Datawarehouse

We have our production data ETL'd every 20 minutes to our Phoenix data center for our analytics product, Metrics.

Cache Proxies

We use the Squid Web Proxy to cache all our front end HTTP web requests (without a query string). http://www.squid-cache.org/

Load Balancer

We might be using HA Proxy for our load balancer, but not entirely positive. Either way, our application is load balancer implementation agnostic. http://haproxy.1wt.eu/

Ad Manager Application Specification

Admanager ad source Feed Test Cases
Ad Manager Controller URLs
Ad Manager JNDI Settings
Ad Mediums and Ad Units Specifications
Controller and Views Specification - Ad Management Application
DAO and ORM Layer - Ad Management Application
Import API Specification
Internationalization (i18n) Service
JSON Return Types
Services - Ad Management Application
Spring Security Service
Spring Security Specification
Status Definitions
User Interface and Experience Specifications

Ad Server Application Specification

Ad Server Cache Information
Ad Server Controller URLs
Ad Server JDNI Settings
AdServer v3.0 and v2.x API Specification
Ad Server v 3.1 html5 adunits ad responses specification
Ad Serving Rules
Code deployment and checkin proccedures
:Impression and Response Tracking Specifications
SDK Backwards Compatibility Service
SDK v3.0 Specifications

Content Manager Application Specification