Managing FreeBSD @ Scale

Reclaiming Control of Large Infrastructure Deployment

Allan Jude

ScaleEngine Inc.

Introductions

Allan Jude

- 11 Years as FreeBSD Server Admin
- Architect of the ScaleEngine CDN (HTTP and Video)
- Host of the weekly TechSNAP.tv Podcast
- Former Professor @ Mohawk College (2008-2011) teaching Network Engineering and Security Analysis
- Extensive work with Puppet to manage our 80 odd servers in 28 data centers in 10 countries
- Lots of work with ZFS to manage large collections of videos as well as extremely large website caches (15+ million objects in ff*ff directories)

Overview

- What is scale?
- Why I don't trust "the cloud"
- What is ScaleEngine?
- Acquiring scale on the open market
- Advanced Puppet
- Lessons learned
- Where to go from here

Scale

What Is Scale?

- "Scalability is the ability of a system, network, or process to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth." [1]

[1] André B. Bondi, 'Characteristics of scalability and their impact on performance', *Proceedings of the 2nd international workshop on Software and performance*, Ottawa, Ontario, Canada, 2000, ISBN 1-58113-195-X, pages 195–203

You Cannot Predict How It Will Scale

Things never turn out the way you think they will

Design For **Eventual** Scale

- Trying to **predict** failure is a losing game
- Design systems with the **expectation** that they will need to scale out.
  - Tackle current, observable issues

  Example: we adopted and enforced a **geographic naming system** long before we had servers in other countries, or even other cities

Planning ...


GOOGLE
CLASSIC
QUERY:
IMAGES
NEWS
VIDEO
MAPS
OTHER
SEND YOUR QUERY TO: GOOGLE INC., 1600 AMPHITHEATRE PARKWAY, MOUNTAIN VIEW, CA 94043, UNITED STATES
PLEASE ALLOW 30 DAYS FOR SEARCH RESULTS

How Do You Scale?

Four usual **methods** once pressure points are observed

- Optimize
- Scale "up"
- Scale "out"
- Use "Cloud"

Optimize

- Make each task take less work so more tasks can be completed with the same resources, or tuning the system to have more resources.
- Often done too early, for no benefit
- Expertise dependent: no expertise, no optimization
- Oftentimes performance tuning is limited and time consuming
  - Example: it's rare to tune the kernel these days
  - In the 2000s, it was common to spend time stripping the kernel

Scale "Up"

You can scale 'Up'

Attempt to make your existing system handle more work

- Scale Up (Vertical) - Buy bigger resources (faster CPUs, more RAM, more Spindles), eventually limited by available hardware or budget
- Which component is limiting your performance?
- Might work
- Might not

Scale "Out"

- You can scale 'Out', adding more nodes to your application, and spreading the work across them
- Scale Out (Horizontal)
- Buy more servers (if your app can be scaled this way)
  - increases management cost and application complexity
  - requires load balancing
  - requires market awareness

Use "Cloud"

Offers Potentially **Immediate Horizontal Scale**

How?

Automagically spinning up additional **virtual machines**

Still introduces additional management complexity, uncapped costs, the uncomfortable fact that you don't get something for nothing...

And...

Additional Potential Fail

Important: Expect To Scale

A business not built to scale, is built to fail.

Many times businesses think they can predict how growth and usage will happen, and build misguided capacity to support these predictions at great expense

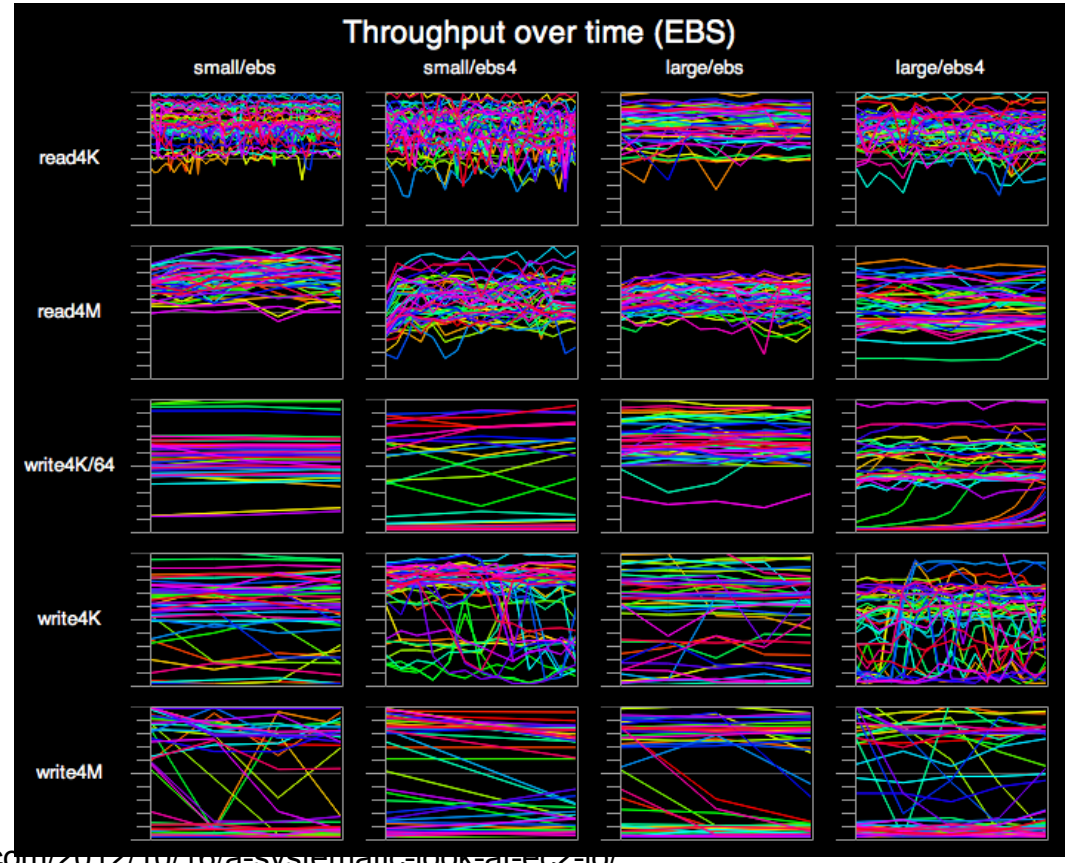
Scale built based on predictions rather than measurements is usually mistargeted and offers little ability to react to actual growth and scalability issues

What Is Wrong With "The Cloud"

- Opacity
  - No visibility inside the cloud -- what is happening?
  - If there is a problem, how do you find it?
  - Following @cperciva is nice but not optimal
- Vendor Lock-in
  - There are many clouds, each is different
  - Different APIs, different abstraction layers
- Price
  - Marketing hype says it is cheaper
  - For sustained usage, usually much more expensive
  - How much is the loss of visibility worth?
  - How much is the loss of control worth?
  - How much is the additional risk worth?

TheCloudIsALie.com

The cloud is ultimately a shared resource, meaning what other people are doing affects your performance



[1] http://blog.scalyr.com/2012/10/16/a-systematic-look-at-ec2-io/

Why I Don't Use "The Cloud"

Sysadmins will jump to cloud providers if immediate scale is required, **reducing autonomy and choice**, and ceding control over many important components to large corporate providers, such as Amazon or Rackspace.

- Many clouds do not offer FreeBSD
- Virtualization performance is often lacking vs bare metal
- Network IO and latency can suffer under Virtualization
- Cloud "hardware" **and network** is more expensive for sustained usage than the open market

Amazon 1st Gen dual core, 7.5gb ram: $262/30 days

Amazon 2nd Gen quad core, 15gb ram: $360/30 days

E3-1230 quad core, 16gb ram: $189/month

Dual E5630 quad core, 256gb ram: $389/month

ScaleEngine?

ScaleEngine is a Global CDN specializing in HTTP Delivery and Video Streaming

- 80 non-virtualized hosts
  - 28 different data centers
  - 10 countries
  - Aggregate 1600GB of ram
  - Aggregate 50 gigabits/sec usable capacity
- All running FreeBSD 9.x - Transitioning to Root-on-ZFS
- Managed by Puppet
- Extensive use of Jails (w/ ezjail)
- HTTP powered by Varnish and NGINX
- The last year (May 1 2012 - May 1 2013)
  - HTTP: 80.8 B requests, 566 TB
  - Video: 257 M requests, 1915 TB

Puppet

- Allows us to quickly scale up and out **on our own terms at the best market rate**
- Deploying puppetmaster for scale
- Creating and using custom facts (freebsd specific facts)
- Advanced configuration files with templates
- Managing packages (with portupgrade)
- Deploying jails with puppet (with ezjail)

What Is Puppet?

Puppet is a configuration management engine. Describe what your server should look like, and Puppet brings the additional server into the desired state.

- Driven by simple declarative configuration files called manifests
- Manages Packages, Users, Files and other configurable objects
- Collects 'Facts' about your nodes
- Uses SSL certificates to verify clients and control access

Starting With Puppet

- This talk does not cover the basics of getting started with puppet
- See "Config Management in FreeBSD using Puppet - Edward Tan, EuroBSDcon 2012" http://youtu.be/nto9HPes8Ko
- Or Edward Tan's Article in BSDMag - http://bsdmag.org/magazine/1784-freebsd-get-up-to-date
- We use the shell_config and ports_conf macros from http://projects.puppetlabs.com/projects/1/wiki/Puppet_Free_Bsd

Replacing The Default Web Server

- WEBRick - Default, Ruby library, single threaded, only good for testing

Use **nginx** to get SSL offloading; a real web server is much better at scaling SSL than a ruby script

- Mongrel - Partially deprecated in Puppet 2.7, fixed size pool of workers, can be distributed across hosts
- **Passenger** - module for Apache or **nginx** to run ruby apps, dynamic workers (can be distributed with nginx or mod_proxy)

I originally used Mongrel because it was closer to the PHP-FPM type workflow I was used to, but when I upgraded to Puppet 3 I switched to **Passenger**

Bypassing Ruby Entirely

Part of our puppet deployment involves installing some basic jails on every host using **ezjail**

This requires puppet to pull some rather large jail archives from the puppet master

To accelerate this and avoid the overhead caused by reading the file in ruby and copying it to the socket, we have **nginx deliver the files directly**, bypassing ruby entirely

nginx verifies the SSL client certificate to ensure that only authorized clients can access the files

nginx Config

```
server {
        listen 8140 ssl;
        ssl_certificate /var/puppet/ssl/certs/puppetmaster.scaleengine.net.pem;
        ssl_certificate_key /var/puppet/ssl/private_keys/puppetmaster.scaleengine.net.pem;
        ssl_client_certificate /var/puppet/ssl/ca/ca_crt.pem;
        ssl_crl /var/puppet/ssl/ca/ca_crl.pem;
        ssl_verify_client on;
        root /usr/local/etc/puppet/rack/public;
        passenger_enabled   on;
        passenger_set_cgi_param    HTTP_X_REAL_IP $remote_addr;
        passenger_set_cgi_param    HTTP_X_CLIENT_DN $ssl_client_s_dn;
        passenger_set_cgi_param    HTTP_X_CLIENT_VERIFY $ssl_client_verify;
        passenger_set_cgi_param    HTTP_X_SSL_SUBJECT $ssl_client_s_dn;
        types { }     # make sure we serve everything as raw
        default_type application/x-raw;
        ##location blocks go here (next slide)
}
```

Serve Files Directly

```
# serve static file for the [files] mountpoint
location /production/file_content/files/ {
        alias /usr/local/etc/puppet/files/;
}

# serve modules files sections
location ~ /production/file_content/[^/]+/files/ {
        alias /usr/local/etc/puppet/modules;
        # rewrite /production/file_content/module/files/file.txt
        # to /module/file.text
        rewrite ^/production/file_content/([^/]+)/files/(.+)$ $1/$2 break;
}
```

New Puppet Agents

In the previous config, **nginx** will only talk to you if you have an SSL client certificate (you are already an approved puppet agent)

There are two ways around this:

- Configure ssl_verify_client to **optional** and add checking for the SSL certificate **before files are served directly**
- Configure nginx to **listen on a second port** with ssl_verify_client optional and **pass everything to ruby**, then puppet agents use --ca_port 8141
  - Also helpful if you have **multiple puppetmasters**, as you can only have a single CA, but the additional puppetmasters can proxy the CA port to the real CA server

Templates Instead Of Configs

- Rather than just deploying the same config file to every server, we use Puppet templates to customize the config file for each server
- We use the 'facts' that puppet collects about each system to manage the configuration
- We create our own 'facts' that are distributed from the puppetmaster to the nodes, allowing us to collect specific information
- We define variables in our node configuration based on role, geography, hardware and policy

Node Configuration

```
node 'Chicago2.CHI1.ScaleEngine.net' {
    $se_host = 'Chicago2'
    $se_fqdn = 'Chicago2.CHI1.ScaleEngine.net'
    $se_location = 'CHI1.ScaleEngine.net'
    $se_identity = 'CHI1-2'
    $se_varnish_ips = ['23.19.138.19','23.19.138.18']
    $se_wowza_ip = '23.19.138.21'
    $se_nginx_ip = '23.19.138.22'
    $se_max_mbps = 600
    $se_norm_mbps = 400

    include edgeserver
    include videohost
}
```

Custom Facts

```
Facter.add("sysctl-hw_physmem") do
  confine :kernel => :freebsd
  setcode do
      Facter::Util::Resolution.exec('/sbin/sysctl -n hw.physmem').to_i
  end
end

Facter.add("sysctl-hw_ncpu") do
  confine :kernel => :freebsd
  setcode do
      Facter::Util::Resolution.exec("/sbin/sysctl -n hw.ncpu").to_i
  end
end
```

Varnish Template

```
<% varnish_ips = Array.new -%>
<% se_varnish_ips.each do |vip| -%>
<%   varnish_ips << vip.concat(":80") -%>
<% end -%>
varnishd_listen="<%= varnish_ips.join(",") -%>"
varnishd_config="/usr/local/etc/varnish/scaleengine.vcl"
varnishd_storage="malloc,<%= (se_memory.to_i * 0.25).to_i -%>M"
varnishd_identity="<%= se_identity -%>"
varnishd_params="-p thread_pool_add_delay=5 -p thread_pools=<%= sysctl-
hw_ncpu.to_i -%> -p thread_pool_min=48 -p thread_pool_max=<%= (8192 /
sysctl-hw_ncpu.to_i).to_i -%>"
varnishd_flags="-P ${varnishd_pidfile} -a ${varnishd_listen} -f
${varnishd_config} -s ${varnishd_storage} -u ${varnishd_user} -g
${varnishd_group} -T ${varnishd_admin} -i ${varnishd_identity}
${varnishd_params}"
```

Installing Ports & Packages

```
Package {
        provider    => portupgrade,
        ensure    => installed,
        require    => File["/usr/local/etc/pkgtools.conf"],
}
file { "/usr/local/etc/pkgtools.conf":
                owner           => root,
                group           => wheel,
                mode            => 0644,
                source          => "puppet:///files/pkgtools.edge",
}
```

Creating Package Definitions

```
class bsdpkg::denyhosts {
    package { "security/denyhosts": }
    file { "/usr/local/etc/denyhosts.conf":
        source      => "puppet:///files/denyhosts.conf",
        require     => Package['security/denyhosts'],
    }
    ports_conf { "denyhosts_enable":
        port        => "denyhosts",
        key         => "denyhosts_enable",
        value       => "YES",
    }
    service { "denyhosts":
        ensure      => running,
        name        => "denyhosts",
        enable      => true,
        subscribe    => File['/usr/local/etc/denyhosts.conf'],
    }
}
```

Installing EZJail

```
class bsdpkg::ezjail($os_version = "9.1-RELEASE") {
    package { "sysutils/ezjail":
        provider    => portupgrade,
        ensure      => present,
    }
    exec { "ezjail_setup":
        provider => "shell",
        command => "/usr/local/bin/ezjail-admin install -mp -r ${os_version}",
        creates => "/usr/jails/basejail",
        require => Package['sysutils/ezjail'],
    }
    ports_conf { "ezjail_enable":
        port        => "ezjail",
        key         => "ezjail_enable",
        value       => "YES",
    }
}
```

Creating Jails

```
class ezjail($jail_name, $jail_ip, $jail_archive = undef, $jail_root = "/usr/jails") {
    exec { "create_jail_${jail_name}":
        provider     => "shell",
        command   => "/usr/local/bin/ezjail-admin create -a ${jail_archive} ${jail_name} ${jail_ip}",
        creates      => "${jail_root}/${jail_name}",
        require      => [
                Package["sysutils/ezjail"],
                Exec["ezjail_setup"],
                File["${jail_root}/${jail_archive}"],
         ],
        notify       => Service["ezjail_${jail_name}"],
    }
    file { "${jail_root}/${jail_archive}":
        ensure       => file,
        source       => "puppet:///archives/${jail_archive}",
        backup       => false,
    }
```

Creating Jails (Continued)

```
    service { "ezjail_${jail_name}":
        ensure      => running,
        name        => "ezjail",
        enable      => true,
        hasstatus   => false,
        start       => "/usr/local/bin/ezjail-admin start ${jail_name}",
        stop        => "/usr/local/bin/ezjail-admin stop ${jail_name}",
        restart     => "/usr/local/bin/ezjail-admin restart ${jail_name}",
        status      => "/usr/local/bin/ezjail-admin console -e /usr/bin/true ${jail_name}",
        }
}
#In your Node config:
class { 'ezjail':
    jail_name       => 'chi2-test.scaleengine.net',
    jail_ip         => '23.19.138.20',
    jail_archive    => 'video_jail.tar.gz'
}
```

Open Market Server Acquisition

Based on the current market for commodity server hardware and network transit, we can quickly add several nodes, with predictable, negotiated monthly cost, and can rapidly configure each node based on its disk size, memory configuration, and cpu

Some providers provision new servers w/ IPMI in as little as 10 minutes

Cloud Virtual Server Acquisition

Contrast this with a cloud provider, where we control less of the specifics of geography, server configuration and network transit cost

We are reclaiming large infrastructure deployment with commonly available open source tools, to deploy FreeBSD quickly and efficiently when we require additional scale

More to do

- Switch to PKGNG and build our own custom binary packages, faster installs, easier upgrades and more control
- Automate our Root-on-ZFS deployment, rather than booting stock .iso and partitioning with scripts, deploy a small image to the drives then gpart resize
- Augment deployment image to have puppet preinstalled for faster bootstrapping
- Auto-tune web and video disk cache sizes based on available space (ZFS dataset, maybe with a quota)
  - Based on size not free space so puppet rewriting the config and restarting the service constantly
- Auto-tune ZFS (ARC, vdev cache, etc) based on system specs gathered by facter

Going Forward

- FreeBSD patches for facter, some of our custom facts should be standard, many facts exist for OpenBSD but not FreeBSD even though they are collected the same way
- 'stored configs' was deprecated in 3.x, investigate puppetdb (needs porting), would allow hosts to know about each other
  - Using puppet to automatically configure nagios
  - Easier management of ssh known_hosts
- Investigate switching to augeas for rc.conf etc, rather than the shell method