PRIME
FOCUS

Deadline Scaling

Sean Cody
Managing Director of IT, North America

Table Of Contents

Introduction

Executive Summary

Deadline is a distributed and hassle free render farm and job management system. Deadline is designed to manage product agnostic render farms for distributed computing and isn't necessarily restricted to a visual effects or media pipeline. The distributed design of deadline is centered around a farm of discrete compute elements rather than the traditional High Performance Computing (HPC) 'single instance' cluster. Getting Deadline up and running is pretty easy. Though as your business grows so much your rendering infrastructure. The purpose of this white-paper is to describe the various ways you can design a Deadline based farm and some approaches in both networking, equipment and system tuning to allow for scaling a farm from a few nodes to a few thousand.

Overview

In this paper we will talk about the various ways one can approach setting up a render farm to maximize performance with respect to deadline.

We will start out by discussing how we define, measure and approach the concept of performance followed by an overview of the challenges faced when trying to optimize a diverse platform like a Deadline render farm. We will then go over a few network design configurations which can be tuned to how Deadline works.

Once we get the network out of the way we will talk about the hardware and system configuration choices to maximize the performance of a deadline repository in various environments.

Performance

Challenges

Network Design

Network design can have a drastic impact on how your render farm performs. As a network grows decisions made early on can either facilitate or hinder scaling and good network design isn't always obvious.

A Deadline repository is essentially a file server based queueing system. Since state and queuing are handled via simple files and the Slave application behaves accordingly there is no 'middle-man' issues or head-end controller failures (pulse is a special case such that is a pluggable meta-data proxy/accelerator and not required for queueing to work).

Simple

The most common network architecture is a simple flat network with a Deadline repository, some workstations, a few render nodes and some form of common storage. The Deadline repository itself is a simple server with an ethernet interface possibly attached to a Windows domain (style of domain is largely irrelevant with respect to Deadline). In this configuration the Deadline repository is a single point of contention and the link to the rest of the production devices needs to be some factor of oversubscription for network capacity.

For instance, in the network depicted in figure 1, we have 100 devices and a repository with a single link to that network. If the devices were all using gigabit ethernet and the

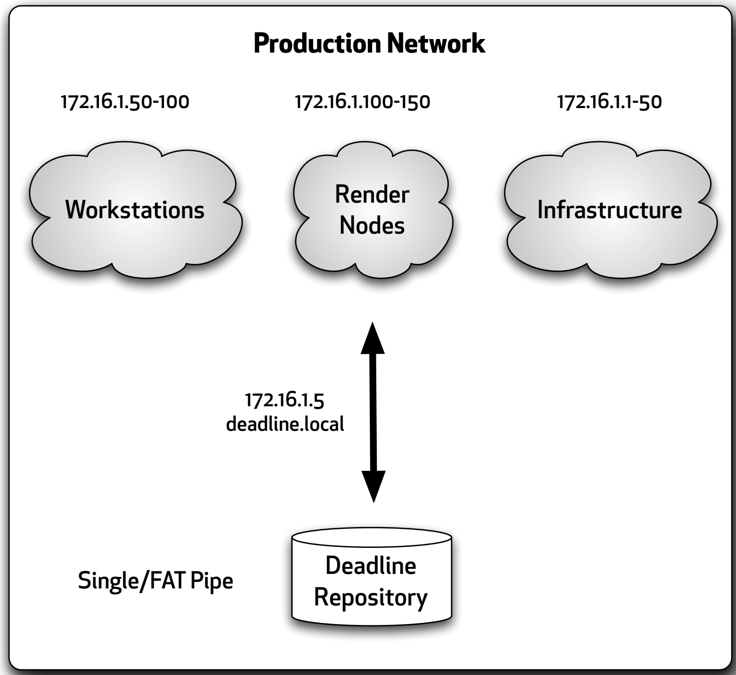


Figure 1. Simple Network

repository was also using gigabit ethernet we have a 100:1 oversubscription ratio (not counting infrastructure). In this configuration it would make more sense to lock the workstations and render nodes to 100 megabit reducing the oversubscription rate to a more manageable 10:1. Of course you can also just use a 10 gigabit interface and uplink on the Deadline repository server as well to achieve the same effect. As a general rule, the minimum suggested link speed for a Deadline repository is at least a single gigabit-ethernet (full-duplex) link. This may be an obvious point, but in reality the Deadline repository is just a 'simple file server.' So design decisions used to build and scale SAN or NAS are worth considering.

Link Aggregation

Even with a manageable oversubscription rate you can come to the situation where the Deadline server becomes overloaded (such as constant 100% farm utilization). In this scenario the Deadline Monitor application will become sluggish and job render times will incur a lag penalty. Deadline Slave is smart enough to minimize this lag, but under extreme work loads even the best optimization becomes strained.

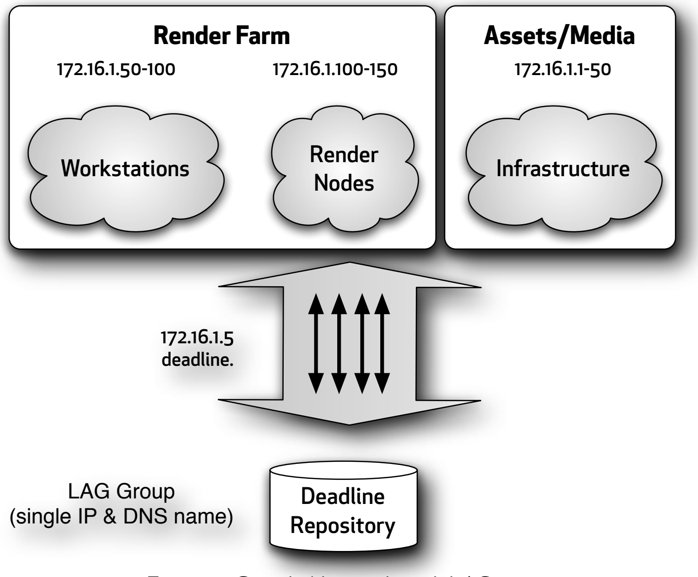


Figure 2. Simple Network with LAG grouping.

One way to simply increase capacity is to trunk or bond multiple network interfaces on the repository server (as depicted in figure 2). There are many ways to accomplish this. Though if your network switching fabric supports it, a LAG (link aggregation group) is the preferred method. A LAG is a layer-2 protocol where traffic is balanced across multiple interfaces and is addressable by a single IP address. The load balancing of traffic across the aggregated link is best determined by the device routing the traffic (which is usually a switch). Most, if not all, server operating systems support link aggregation in some form with the most common being 802.3ad LACP (link aggregation control protocol). If you do not have a managed switch which will allow for LACP trunking you can also opt for server based load balancing. This is not optimal from a networking point of view it is an option. With link aggregation you can reduce your oversubscription rate by a factor of the number of interfaces in the group. In the example shown in figure two we've taken the 100:1 oversubscription (assuming everything links at gigabit ethernet) and reduced it to 25:1. On paper this looks great though even with the best load sharing algorithms and a tuned LACP hashing scheme you will not get a perfectly balanced load across each interface.

Now if you are in the unfortunate scenario where you don't have an LACP supported switch and the repository host OS doesn't support any form of aggregation or load balancing at the interface level you can still use multiple interfaces. We will call this scenario "poor-man's trunking." To accomplish this you put a number of network interfaces into your repository server and assign each interface a unique address on the network. You then setup DNS on your network such that the network name of the repository points to each of the IP addresses assigned in a round-robin fashion (essentially multiple A records for the same host). In this configuration every lookup for the name assigned to the repository will return a different IP address in the set in a round-robin fashion. This lookup will persist for the cacheable lifetime of the DNS lookup but it will definitely distribute hosts between the interfaces. The frequency of lookups and the client name cache timing will define the level of load balancing. This is the least preferred configuration, though in a pinch it can make a significant performance enhancement while better options are investigated.

VLANs

When dealing with large numbers of hosts and a high oversubscription rate another option to consider is segmenting the network into virtual networks or VLANs. Segmenting a network into VLANs can be done many ways, but a method we've seen much success with is by segmenting the network by host function. In the simple render farm examples we've seen so far we can segment the network into 3 pieces. We have workstations, render nodes and infrastructure (as in servers and storage systems). We can then create three separate VLANs which groups of hosts (corresponding to those three classifications) into VLANs which we will assign a class C block of IP addresses as depicted in figure 3. This will or could (depending on

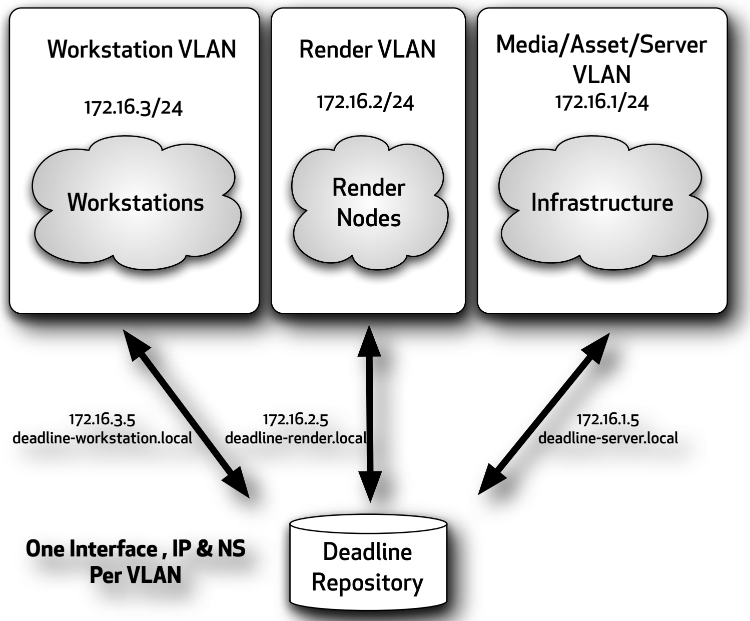


Figure 3. Segmented VLANs.

configuration) isolate broadcast and unicast traffic which would also reduce switch processing load.

In this configuration you can setup multiple interfaces on the repository server, (one for each VLAN) and assign local addresses to each interface accordingly. This is a rather crude but explicit technique to isolate and load balance traffic on a network.

This method requires your switching infrastructure to support VLAN tagging and if that isn't available this can be accomplished by using separate switches for each of the network segments and having a router to move traffic between the physically separate networks. VLANs are a means of virtually doing this and reducing equipment needs.

Pulse

After a few years of using Deadline for some rather intense projects/productions we grew faster than our infrastructure could handle which gave us an opportunity to find some of the 'low hanging fruit' and optimize communication between render nodes and the repository. Since the relationship between the slave instances and the repository is managed entirely with small XML data files this can put a lot of strain on disk and network resources on the repository directly correlative to the number of nodes in the farm. While development is progressing towards a relational database backed repository component this load would still need to be handled. At the time the obvious option was to abstract the repository from the file system adding a middle ware layer to keep render nodes from touching the repository directly.

This was a terrible option as that would introduce a 'server' or 'repository master' relationship completely invalidating the self-healing nature of the distributed queuing model.

The solution instead was to add support for a bolt-on proxy which would do that abstraction and proxy job management (and subsequently other management functions). This solution is called Deadline Pulse. The beauty of this option is that it is not required to have a functioning render farm. If it fails (which is rare) the farm continues to manage and execute jobs though while it is running it reduces the load on the repository back end (in the current case a file server).

Without Pulse running the CPU load of a Deadline repository sky rockets. Under light load the UNIX based load meter shows a load of less than 1 (which means there are less processes that can be serviced in the run queue than there is capacity to run, essentially the idle loop). Under heavy load (say >50 nodes rendering constantly) the UNIX load will jump to 4 or 5 (meaning there are 5 processes waiting in the run queue per scan). Once you get even larger we've seen the load spike to 50+. When the load of the machine is this high interactivity of the Deadline Monitor application gets sluggish, there is delay in job pickup and finalizing. The repository is still functioning through it is essentially overloaded. Using a Windows based repository you will see the CPU constantly at 100% under task manager with kernel time making up 70%+ of that total. The solution to this is just a simple as running a Deadline Pulse server on another host. Deadline Pulse will act as an intermediary proxy for job submission and interactive Deadline Monitor queries and will cut out the chattiness of the CIFS protocol (by using an XML based communication protocol) as well as reduce the easily cacheable file system queries (ie. the meta-data lookups).

Regardless of the type or scale of the farm it is highly recommended to run Deadline Pulse if you have the hardware to spare. For medium to large farms we would go so far as to say it is required for efficient operation and productivity of your resources.

Repository Server Design

Once you have your network design as optimized as your infrastructure allows you should consider how you are building your repository server itself. For the budget conscious, an extremely capable and performant repository can be put together for very little money. The hardware requirements for a Deadline repository is rather insignificant though spending a few extra resources on a specific set of key components can make a world of difference and allow for significant scaling with minimal effort.

Operating System

Since Deadline is hosted primarily off a network file share, the operating system of choice is largely open to whatever your environment and infrastructure can handle and support. The most popular operating system by far is Windows Server with a minimum suggested version of Windows 2000 Advanced Server. Linux with Samba and NFS is definitely another option as is FreeBSD, Solaris and Mac OS X. It is safe to say that any operating system with some form of network file share that clients can address via a UNC will be adequate for hosting a repository. At Prime Focus our North American facilities host our repositories on FreeBSD (currently 7.2 with plans to migrate to 8.1). For the purposes of this document we will cover Windows and FreeBSD though many considerations given to FreeBSD are applicable to other UNIX like operating systems since the application layer is essentially identical.

Windows Hosted

The first and only major concern with a Windows based repository is the maximum number of open sockets for the network file share. Assuming you have a enough CALs for all your render nodes and workstations there isn't too much tweaking needed (or even available) aside from tuning the network stack. The non-server class connection limit is the reason Windows Server is suggested, though if you only have a handful of nodes and clients (that is less than 10) you can use the desktop versions of Windows as well though your millage will vary.

We do not suggest multitasking (or hosting multiple services) on a deadline server due to the volume of sustained IO requests. Under a heavy load, it is not uncommon to see the CPU usage pegged almost totally in kernel calls. CIFS can be extremely chatty and under high numbers of simultaneous connections the network interface will generate Interrupt storms such that the machine will spend the majority of it's servicing the network controller driver. Jumbo frames may marginally help bring the packet rate (and therefore interrupt rate) down though we expect that gain to be marginal relative to volume.

That being said it is highly recommended that you run the Deadline Pulse service on a separate host than the repository. Just running Pulse will net 70-80% less load on the repository when job scheduling and repository scans (via Deadline Monitor) are substantial.

FreeBSD and UNIX Hosted

Just as in a Windows environment keeping the running services to a minimum is highly recommended. A stock install of FreeBSD and Samba 3 is all you need to get up and running. If you are wanting to run Pulse on a UNIX host you will need to also install Mono 2.6 and have a running X11 environment. If you are not using pre-built packages or are given a choice then we would recommend not building in CUPS support (for printing), ACL extensions or LDAP support. It does however help to build in support for the VFS modules especially the audit module which is **very** helpful when debugging file share issues. You don't need to enable these by default but having them on hand is invaluable in a pinch.

The Samba configuration is best kept to a minimum though tweaking the IP options to at minimum of TCP_NODELAY is suggested. If you are going to have Mac OS clients you will need to turn off UNIX extensions due to a bug in the Samba support OS X ships with. The single best feature to enable is sendfile as this will greatly simplify the IO path and maximize throughput.

Included below is a complete Samba configuration file (Figure 4, smb.conf) that is used in production on a heavily loaded network.

```
[global]
        workgroup = DOMAIN
        hosts allow = 172.16.0.0/255.255.255.0
        server string = Deadline Repository
        security = share
        log file = /var/log/samba.log
        max log size = 1024
        local master = No
        stat cache = yes
        strict sync = no
        use sendfile = yes
        domain master = No
        write cache size = 524188
        read raw = yes
        write raw = yes
        dead time = 15
        wins support = Yes
        name resolve order = hosts bcast
        dns proxy = No
        getwd cache = yes
        large readwrite = yes
        stat cache = yes

[repository]
        path = /mnt/ssd/repository_4
        guest ok = yes
        guest only = yes
        writeable = yes
```

Figure 4. /usr/local/etc/smb.conf

Once Samba is installed and running we can focus on the kernel optimizations. There are a few notable sections to consider. The first is the file descriptor and process sysctls. Under high simultaneous connection rates you can expect many open smbd processes and at a minimum they will require at least four file descriptors. While recent versions of FreeBSD is very good at auto tuning this it is something to keep in mind as the stock distribution is tuned for a general purpose multiple user server. The other section of note is the network stack sysctls. These are a bit harder to tune as you need to calculate values relative to expected and maximum load to avoid resource exhaustion. Below (in Figure 5), is a sample set of explicitly set sysctls (/etc/sysctl.conf) for FreeBSD 7.2.

```
# $FreeBSD: src/etc/sysctl.conf,v 1.8 2003/03/13 18:43:50 mux Exp $
#
#  This file is read when going to multi-user and its contents piped thru
#  ``sysctl'' to adjust kernel values.  ``man 5 sysctl.conf'' for details.
#

# Uncomment this to prevent users from seeing information about processes
that
# are being run under another UID.
#security.bsd.see_other_uids=0
kern.ipc.maxsockbuf=8388608
net.inet.tcp.sendspace=65535
net.inet.tcp.recvspace=65535
net.inet.tcp.delayed_ack=0
vfs.hirunningspace=5242880
vfs.lorunningspace=5242880
net.inet.tcp.local_slowstart_flightsize=65535
kern.maxfilesperproc=2048
kern.ipc.somaxconn=4096
kern.maxfiles=65536
kern.ipc.nmbclusters=65536
vfs.ufs.dirhash_maxmem=10485760
net.link.ether.inet.log_arp_wrong_iface=0
```

Figure 5. /etc/sysctl.conf

CPU

The CPU is the least important part of a Deadline Repository server. While you can technically use anything we recommend at least some form of multiprocessor support whether it be a multicore or multiprocessor system. Under a high load an SMP environment allows for at least one processor to be available for interactivity which is doubly important when hosting under Windows. In this load situation we find Windows on a single processor system to be sluggish and unresponsive but this effect is far less in an SMP environment. When the system task (or kernel) is heavily loaded it tends to monopolize on one processor, so the UI message pump is free to run on other processors, which is far easier to interact with. On UNIX hosts this isn't as much of a concern as the shell (and SSH session) is scheduled with the same or better priority as the many samba processes so it doesn't feel as sluggish, but if SMP support is decent as is in FreeBSD 7+ and Linux 2.6+ then the extra resources will be felt by better interactivity at the client end.

We do not recommend the budget style processors (such as the Celeron or Sempron lines) as they tend to skimp on L2 cache which is important for highly repetitive tasks like file serving. At the other end of the spectrum is the heavy bitters like the Intel Xeon line. These processors are overkill for this kind of server as CPU load will be **very** minimal under UNIX and nominal (on secondary processors/cores) under Windows.

When making a CPU choice we recommend using the midrange processors and choose high bandwidth chipsets to keep the memory path from being a bottle neck especially for bulk DMA transfers between storage and network interfaces.

Memory

Memory size and choice is more important in a Windows environment than under UNIX. As stated previously faster memory is worth more in this server scenario than quantity. ECC RAM is suggested if the budget allows. The amount and variety of DIMMs should be matched to the chipset chosen for maximum performance. For instance, for some Opteron multiprocessor systems four sticks of matched DDR will perform better than a single matched pair. As one can dedicate a DDR pair per processor. This doesn't necessarily hold true for Intel's i7 QPI platform. When designing a new system refer to the chipset documentation for recommendations on memory configuration.

For Windows we recommend 2 gigabytes of RAM as is usually recommended for Windows server platforms. In UNIX you can get away fine with a single gigabyte of RAM though like in Windows, there are a few scenarios such as tuning file system and buffer caches would make use of and perform better with more RAM. This is doubly so if ZFS is chosen as the file system. If you are in the *over clocker crowd*, you can setup a memory based file system strategy on top of a repository to maximize IOPS, though that is a very advanced topic.

Storage

Storage is definitely where we suggest the bulk of your budget. A slow system with fast storage will run circles around a fast system with nominal speed storage. Since a Deadline repository is essentially a file server the system should be tuned for high IOPS with small block random access. The highest hit rate files will be the 'slaveInfo' and 'job task' files which tend to be very small but are accessed very frequently. If storing scene files on the server (which is default) they can range in size from 10's of kilobytes to hundreds of megabytes. The access rate of scene files is far lower than that of the job queue and slave information files so optimizing for bulk file transfers will only net marginal performance gains for the average case.

Server grade SSD's are a fantastic choice for this kind of file system load though commodity SSDs with slow write performance are to be avoided. We use and recommend the Intel X25-E and X25-M drives with an OS that supports the TRIM SATA extensions. Other manufacturers and drive controllers like some of the higher grade Indilinx and SandForce controllers look to be a competing product though we have not tried or tested those internally as of yet so your mileage may vary.

If SSD's are outside your budget then you must stick with 'spinning metal' and we recommend at least a 10,000 RPM disk or better. We can't stress enough how disk speed will affect performance under load. In a pinch a stripe of 7200 RPM disks will suffice but once you factor in the cost of a decent RAID controller you come close to the opportunity cost of an SSD.

Finally it is recommended that the volume you store the repository is separate from the operating system install to dedicate and segregate IOPS to file serving over local disk traffic such as software startup or virtual memory management operations.

RAID

For server environments RAID is almost always a concern. In the case of a Deadline repository we do not recommend any of the RAID types that involve parity calculation. Parity calculation will floor maximum throughput on storage controllers as we've seen previously with 3ware controllers. For this environment RAID-10 is the recommended production configuration balancing speed with fault tolerance, though internally we have had a lot of success with RAID-0 stripes (of SSDs) that is periodically backed up to a different volume (such as a two disk RAID-1 mirror of 7200 RPM 'spinning metal') as this balances cost and performance.

As far as RAID controllers go, we heavily suggest using dedicated hardware with dedicated cache memory as opposed to software assisted or software RAID. That isn't to say it isn't a good choice but it will not be as performant or reliable. We have used a Windows software based RAID-10 with SCSI disks in production with a lot of success for years before we converted to using SSDs in our repositories.

Depending on the controller and the disk scheduling algorithm choosing a small stripe size (32 or 64 kilobytes) is preferred due to there being a very small average file size and high volume small block commit pattern to the disk systems.

SSD/SAS/SATA/SCSI

The kind of disk you choose will have a dramatic result on repository performance. While we suggest SSDs we understand that being a new and expensive technology it isn't always available. That being said we do not recommend or suggest IDE based storage systems due to their implementation and control protocols not being designed for the high IOP rate that you can expect.   At minimum we suggest a SATA disk and controller which supports Native Command Queuing (NCQ). If you have some fast SCSI disks lying around that is also a very good option.

SAN/NAS

Deadline repositories are one application that doesn't make sense to run from a SAN. While some SANs or similar style NAS (like Isilon or BlueArc) have some compelling load and caching systems, the rather low average file size and excessively random access do not suit a block based SAN scenario. The key to Deadline performance is fast small random access, not bulk transfer. If the SAN resources were dedicated to the repository it could be used effectively, but I would recommend putting your budget towards SSDs over SAN HBAs.

Suggested Configurations

Small Render Farm

For the purposes of this recommendation, we are going to define a small render farm as a total of 10 render nodes and workstations combined. In this special case the load will never be too extreme allowing us latitude on many of the performance variables.

Almost any 'off the shelf' PC you can find in a store would suffice though to be specific:

Intel Core 2 Duo

4 GB of RAM

A single 7200 RPM disk for the Operating System and tools. While we suggest FreeBSD or Windows server as the Operating System you can pretty much use whatever you have on hand though remember the 10 client maximum for non-server based Windows systems.

A secondary 7200 RPM disk on a separate controller for the deadline repository. Feel free to add another and if the motherboard supports it using another drive in a RAID mirror is a decent idea though I would heavily suggest testing and forcing a failure before putting into service (to practice a repair scenario should the worst happen).

On the networking side of things we can be just as flexible. A decent gigabit switch is all that is required though if you can spring for a managed switch that can be handy debugging network issues or monitoring state. Having a Deadline Pulse server is not explicitly required here though planning for it should things get a bit sluggish is a good idea.

Medium Render Farm

A medium farm would be between 10 and 100 render nodes and workstations combined. Load and contention can slow things down. The basic CPU and RAM and Operating System requirements are the same as in a small farm though we will need to focus on storage. The very first change we will make is to have a Deadline Pulse server. This will reduce your network load almost logarithmically just by running this proxy service. Most of the Deadline queue management and job delegation will be done via a non file-system line protocol (a specialized web service essentially). Using Deadline Pulse will also allow you to enable power management (assuming your machines are setup for Wake On LAN) and thermal management support (assuming you have an SNMP capable temperature sensor). Any basic server class system will be adequate for running a Deadline Pulse server though gigabit-ethernet is HEAVILY recommended.

On the storage end I would suggest a small but fast SSD. The Intel X25-E or X25-M SSD's are a fantastic choice. If you only use one that is totally fine though I would suggest adding a 7200 RPM disk to periodically backup the repository.

On the networking front, a decent switch will go a long way and if you have a managed switch which supports LAG a dual port LAG group would be a nice touch. You can expand that lag group if you find that two links are proving to be overloaded. Assuming you are starting from scratch, setting up a 'jumbo frame clean' network (that is a network in which EVERY host on that network supports and has jumbo frames enabled) will also reduce network load.

Large Render Farm

A large render farm consists of hundreds of render nodes and workstations. Surprisingly the CPU and RAM requirements still hold but yet again we need to focus on networking and storage to get the most performance out of the farm.

A high end storage system whose focus is on a high number of random access IOPs is essential. A striped RAID group of SSDs can accomplish this (up to the limit of the controller's bus bandwidth). If your budget can support it, specialized storage systems like the FusionIO Duo controllers are a fantastic choice. They are essentially giant (in terms of storage) and extremely fast SSDs attached to a PCI-Express bus. The advantage here is bypassing the SATA bottle neck at the cost of having to run a special driver for the device. In this case your limited to using Windows or Linux (we've been informed FreeBSD drivers for FusionIO devices are immanent) .

If you are going to use a RAID controller choose one that won't bottle neck the SSDs. RAID controllers are designed and optimized for 'spinning metal' drives and have timing and caching designed around that limitation. While is most cases this won't be an issue some tricks in use by some cheaper RAID controllers can lead to some odd and hard to debug throughput problems. The LSI/PERC and 3ware controllers are pretty decent choices in this category.

In production in Prime Focus VFX is to use a PERC based RAID controller with 2 Intel based SSDs in a striped configuration as well as two SAS 15,000 RPM drives mirrored and which is setup to backup the SSD stripe on an hourly basis (remember stripes have NO redundancy so some preventative maintenance is in order).

On the networking side, we have to pull out all the stops. As discussed in the Network Design section of this document a large render farm should be a hybrid of the trunked (LAG) and segmented networks (ie. Figure 3). At Prime Focus VFX the network is segmented to isolate workstation, infrastructure and render nodes. The Deadline repository is hosted on a 10 gigabit segment with the Pulse server hosted off the render node segment on single gigabit ethernet. The render nodes are trunked to a core switch via leaf switches with multiple 10 gigabit uplinks.

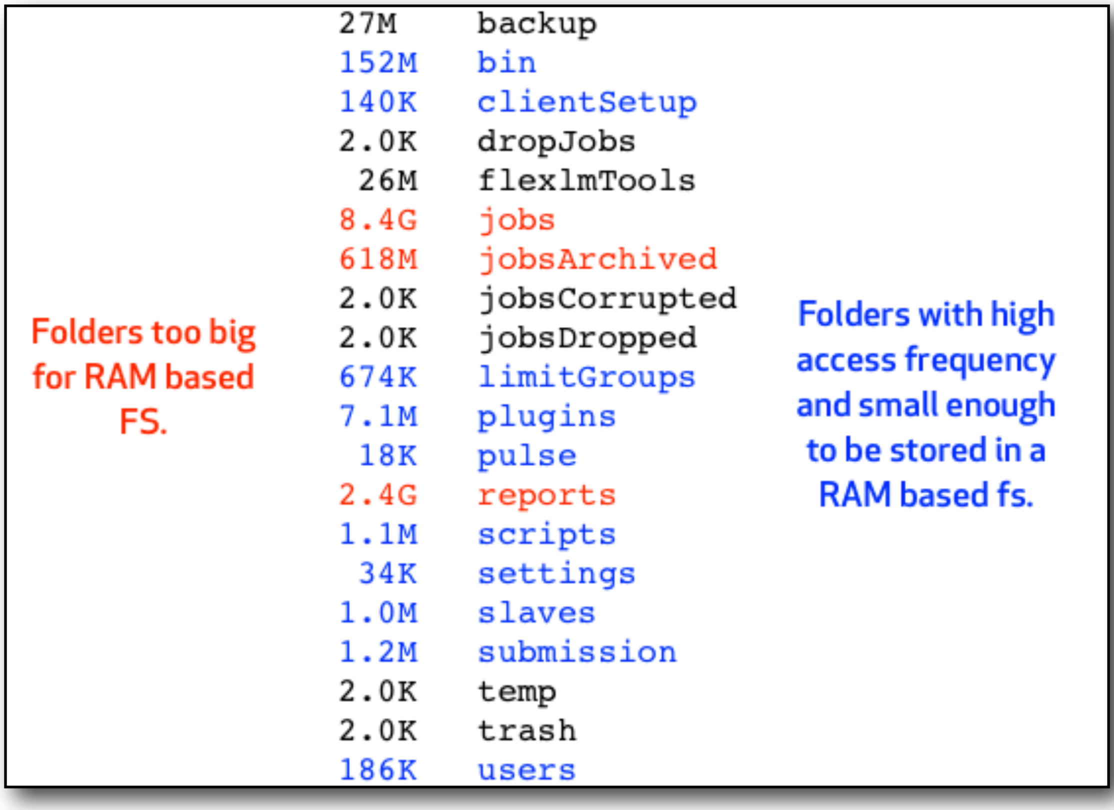
To go further you can implement multiple Deadline repositories such that a production render farm's repository can be completely isolated from workstations and artists and all of it's infrastructure optimized for that isolation. Render nodes (currently) can only be a member of one repository at a time so splitting a chunk of the render farm for production work and the rest for the bulk rendering is required though it will reduce to competition for resources on your network as a whole (not just deadline).

For the 'overclocking crowd'

For extra points (and some serious fun) you can build a hybridized repository where the bulk of your repository persists on an SSD (or RAID set) and the most frequently accessed meta-data is hosted off a memory based (that is in RAM) file system. This is more 'dangerous' than doing a simple RAID stripe so you'll have to plan for periodic backups and develop a means to pre-populate the memory based file system on system start up but in a pinch where specialized storage is not available (because it's pricey) this can be a life saver.

Doing this is left to an exercise for the reader. Note this is not easily done in Windows as you can't easily mount drives inside the file system of another existing volume. In UNIX based system this can be accomplished by setting the mount point of the memory based file system inside the mount point of the disk repository mount or 'union' mounting the file system on top of the existing repository mount point.

Let's look at the layout of a production Deadline Repository. Below (in Figure 6) is the contents of the repository.


Folders too big for RAM based FS.
Folders with high access frequency and small enough to be stored in a RAM based fs.

```
 27M    backup
152M    bin
140K    clientSetup
2.0K    dropJobs
 26M    flexlmTools
8.4G    jobs
618M    jobsArchived
2.0K    jobsCorrupted
2.0K    jobsDropped
674K    limitGroups
7.1M    plugins
 18K    pulse
2.4G    reports
1.1M    scripts
 34K    settings
1.0M    slaves
1.2M    submission
2.0K    temp
2.0K    trash
186K    users
```

Figure 6. Repository Distribution

The bulk of the repository is in the jobs, jobsArchived and reports folders. These aggregate over time and grow/shrink relative to the rate of production. Reports accumulate on every job as well if your repository is set to keep a copy of the scene files with the job meta-data (which is highly recommended) then the the jobs folder will grow and shrink quickly and usually unexpectedly. These folders need fast bulk access but they are too big for a memory based file system. However you will notice the folders highlighted in blue (namely bin, clientSetup, limitGroups, plugins, pulse, scripts, settings, slaves submission, users) are much smaller and can be moved into a memory based file system pretty reasonably. They do not grow and shrink to the same scale as the larger folders previously noted. The slaves and users folders are the most access folders in the repository (per unit of network throughput) so they are very advantageous to store in a RAM based file system.

In production your milage will definitely vary though in a pinch this could speed up user interactivity considerably.

Parting Words

It is our hope that this document gave you some insight on how to grow and manage the resources to build a high performance Deadline based render farm. This guide was intended to give our customers a better insight into how the product works, how to make it scale and chiefly to impart how to get the most of your investment in rendering infrastructure.

If you have any questions or comments regarding this document or the implementation of a Deadline Render farm please visit our forums at:

http://support.na.primefocusworld.com/

Manuals for our products as well as personal support for our products can be found at:

http://software.primefocusworld.com/software/support/deadline/

Thanks

A big thanks to the following people for helping review and build this material. Without them and their enthusiasm to get this information out it would still be a draft in a 'todo' list.

Ryan Russell - Senior Deadline Developer

Cody Lee - Deadline Developer

Ian Fraser - Director of Research and Development

Dan Rosen - International Chief Technology Officer

About Prime Focus

Who are we?

Prime Focus is a global Visual Entertainment Services group. We provide creative and technical services to the Film, Broadcast, Commercials, Gaming, Internet and Media industries. Visual Entertainment Services is a new definition for an industry where technology, visual delivery platforms and content are converging and evolving.

What do we do?

We offer a genuine end-to-end solution from pre-production to final delivery including pre-visualization, equipment hire, visual effects, video and audio post-production, Digital Intermediate, software, digital asset management and distribution.

Contact Us

North America: +1 323 461 7887
UK: +44 207 565 1000
India: +91 22 6715 5000
email: info@primefocusworld.com