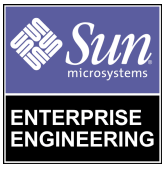
Managing NFS™ Workloads

By Richard McDougall, Adrian Cockcroft and Evert Hoogendoorn - Enterprise Engineering

Sun BluePrints™ OnLine - April 1999



http://www.sun.com/blueprints

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 806-3838-10
Revision 01, April 1999

Managing NFS™ Workloads

Applications often run across multiple systems, and a system often runs multiple applications. Application workloads can be defined in terms of the resources they consume on each system. This article explores how different resource management tools can be used to manage an NFS™ workload.

The unique characteristics of an NFS workload place different resource demands on the system. No single product can provide complete resource management of an NFS workload. The resources used by NFS and the different products or techniques that can be used to manage them are shown in TABLE 1.

TABLE 1 NFS Workload Resources

Resource Type	Management Product/ Method	Comments & Issues
CPU	Processor Sets / nfsd / BM	Limit by constraining the system processor set Control the number of nfsd threads nfsd(1M) indirectly limit the number of NFS ops by using bandwidth manager
Physical Memory	Priority Paging	Install and enable priority paging so that file systems will not consume all physical memory.
Swap Space	N/R	NFS uses very little swap space
Disk Bandwidth	Separate Disks / BM	Either put the NFS file systems on a different storage device or indirectly use bandwidth manager to control the NFS request rate
Disk Space	File System Quotas	Use File System Quotas to limit disk space allocation
Network Bandwidth	BM	Use bandwidth manager to control the rate of NFS ops

Controlling NFS CPU Usage

NFS™ servers are implemented in the Solaris™ Operating Environment kernel as kernel threads and run in the system class. You can control the amount of resource allocated to the NFS server in three ways:

- Limit the number of NFS threads with nfsd.
- Limit the amount of CPU allocated to the system class with psrset.
- Indirectly control the number of NFS ops with bandwidth manager.

You can control the number of NFS threads by changing the parameters to the NFS daemon when the NFS server is started. Edit the start-up line in /etc/rc3.d/S15nfs.server file:

```
/usr/lib/nfs/nfsd -a 16 (change 16 to number of threads)
```

The actual number of threads required will vary according to the number of requests coming in and the time each thread spends waiting for disk I/O to service the request. There is no hard tie between the maximum number of threads and the number of NFS threads that will be on the CPU concurrently. The best approach is to approximate the number of threads required (somewhere between 16-64 per CPU), and then find out if the NFS server is doing its job or using too much CPU time.

NFS is a fairly lightweight operation, so it is unlikely that the NFS server CPU usage is an issue. The CPU time consumed by the NFS server threads accumulates as system time. If the system time is high, and the NFS server statistics show a high rate of NFS server activity, then curtail CPU usage by reducing the number of threads.

A far more effective way to control NFS servers is to use the bandwidth manager product to limit the traffic on the NFS port, 2049, and indirectly cap the amount of CPU used by the NFS server. The disadvantage is that spare CPU capacity can be wasted because managing by bandwidth usage does not reveal how much spare CPU is available.

To understand if you have over constrained NFS's allocation of resources you can use the new NFS iostat metrics and look at the %busy column.

NFS Metrics

Local disk and NFS™ usage are functionally interchangeable, so the Solaris™ 2.6 operating environment was changed to instrument NFS client mount points the same way as disks. NFS mounts are *always* shown by `iostat` and `sar`. With automounted directories coming and going more often than disks coming online, that change may cause problems for performance tools that don't expect the number of `iostat` or `sar` records to change often.

The full instrumentation includes the wait queue for commands in the client (biod wait) that have not yet been sent to the server. The active queue measures commands currently in the server. Utilization (%busy) indicates the server mount-point activity level. Note that unlike the case with simple disks, *100% busy does not indicate that the server itself is saturated*; it just indicates that the client always has outstanding requests to that server. An NFS server is much more complex than a disk drive and can handle many more simultaneous requests than a single disk drive can.

The following is an example of the new `-xnP` option, although NFS mounts appear in all formats. Note that the `P` option suppresses disks and shows only disk partitions. The `xn` option breaks down the response time, `svc_t`, into wait and active times and puts the device name at the end of the line so that long names don't mess up the columns. The `vold` entry automounts floppy and CD-ROM devices.

```
% iostat -xnP
                    extended device statistics
  r/s  w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0 vold(pid363)
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0 servdist:/usr/dist
  0.0  0.5    0.0    7.9  0.0  0.0    0.0   20.7   0   1 servhome:/export/home2
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0 servmail:/var/mail
  0.0  1.3    0.0   10.4  0.0  0.2    0.0  128.0   0   2 c0t2d0s0
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0 c0t2d0s2
```

NFS Physical Memory Usage

NFS™ uses physical memory in two ways: in the kernel for each thread it is running and indirectly through the file system. The amount of kernel memory used by NFS is a reasonable static and easily manageable because its size doesn't change greatly and the amount of memory required is so small.

The amount of memory used by the file systems for NFS servers is, however, very large and much harder to manage. By default, any non sequential I/O and non-8k I/O uses memory at the rate of data passed through the file system. The amount of memory used grows continuously, and when there is no more free memory, memory is taken from other applications on the system.

By using priority paging you can apply a different resource policy to the memory system to prevent this from happening. With priority paging, the file system can still grow to use free memory, but cannot take memory from other applications on the system. Priority paging should be a mandatory checkbox item for any system that has NFS as one of the consolidation applications.

NFS and Swap Space

NFS uses a very small amount of swap space, and there should be no inter-workload swap space issues from NFS.

Controlling NFS with Solaris™ Bandwidth Manager

You can control amount of resources consumed by NFS indirectly by throttling the amount of network bandwidth on port 2049. The Solaris™ Bandwidth Manager product provides the means to do this.

First assess the network interfaces that need to be controlled. If clients come in over several network interfaces, all of these interfaces will have to be brought under control by the Solaris Bandwidth Manager software.

When defining interfaces in the Solaris Bandwidth Manager software, you must specify whether incoming or outgoing traffic needs to be managed. In the case, of NFS software, network traffic could go in both directions (reads and writes). In the Solaris Bandwidth Manager software configuration, this would look as follows:

```
interface hme0_in
    rate      100000000         /* (bits/sec) */
    activate  yes

interface hme_out
    rate       100000000
    activate   yes
```

Next, correctly define the service you want to manage. the Solaris Bandwidth Manager software already has two pre-defined classes for NFS:

```
service nfs_udp
    protocol udp
    ports 2049, *
    ports *, 2049

service nfs_tcp
    protocol tcp
    ports 2049, *
    ports *, 2049
```

Put in place a filter that can categorize network traffic in NFS and non-NFS traffic:

```
filter nfs_out
    src
        type    host
        address servername
    dst
        type    subnet
        mask    255.255.255.0
        address 129.146.121.0
    service
        nfs_udp, nfs_tcp
```

The filter in above example is for managing outgoing NFS traffic to the 129.146.121.0 network. You could decide to leave the destination part out, to manage NFS traffic to all clients, from wherever they come.

Create another `nfs_in` filter for NFS traffic in the opposite direction. Only the `src` and `dst` parts need to be reversed.

Lastly, create a class that will allocate a specific bandwidth to this filter:

```
class managed_nfs
    interface     hme_out
    bandwidth     10
    max_bandwidth 10
    priority       2
    filter        nfs_out
```

This class sets a guaranteed bandwidth of 10 percent of the available bandwidth (10 Mbytes in case of fast Ethernet). Control the maximum bandwidth by setting an upper bound to the CPU resources that the NFS software consumes on the host. The key variable is `max_bandwidth`; it specifies an upper bound to the consumed bandwidth that never will be exceeded. You could even set the `bandwidth` variable to 0, but this could lead to the NFS software starvation if other types of traffic will be managed as well.

The priority variable is less important. It will be a factor if other types of traffic are being managed. Generally, higher priorities will have lower average latencies, because the scheduler gives them higher priority if it has the choice (within the bandwidth limitations that were configured).

It is not easy to find a clear correlation between NFS network bandwidth and NFS server CPU utilization. It depends very much on the type of NFS workload for your server. A data-intense NFS environment will be very different from an attribute-intense environment. Experimentation will determine what's good for you. Your

administrator could even develop a program that monitors NFS CPU utilization, and if it is getting too high, use the the Solaris Bandwidth Manager APIs to dynamically limit the bandwidth more, all automatically and in real time.

Summary

This article shows that a combination of products can be used to manage an NFS server. The Solaris Bandwidth Manager software however, has the most direct control over the resource usage of an NFS server, and provides the simplest method of constraining the workload when combined with other workloads.

Author's Bio: Richard Mc Dougall

Richard has over 11 years of UNIX experience including application design, kernel development and performance analysis, and specializes in operating system tools and architecture.

Author's Bio: Adrian Cockcroft

The author of Sun Performance And Tuning, Adrian is an accomplished performance specialist for Sun Microsystems and recognized worldwide as an expert on the subject.

Author's Bio: Evert Hoogendoorn

Evert is a Networking and Security specialist for the Enterprise Engineering Group.