Benchmarking the Sun Fire X4500

James Thorne[1]

May 31, 2007

Abstract

The Sun Fire X4500, AKA “Thumper”, is a fast, high density storage unit. It provides 24 TB of storage in 4U. It does not have any RAID controllers and relies on the software RAID capabilities of the operating system.
The performance of the X4500, both local and NFS, under Solaris x86 and Scientific Linux 4.4 is examined. The methods of testing are described and the results are discussed. A problem was encountered when running the tests in Linux and the steps taken to investigate this problem are explained.
Conclusions are drawn regarding the performance of the two operating systems under test and the suitability of the machine for use as a disk server within the Tier1.

[1] Tier1, Science and Technology Facilities Council, Rutherford Appleton Laboratory, Harwell Science and Innovation Campus, Didcot, OX14 0QX

1 Introduction

1.1 History

RAL acquired a Sun Fire X4500 (AKA “thumper”) on loan from Sun Microsystems in order to test its suitability for use in the Tier1. The machine was designed for the media market to stream large quantities of video and audio but could be used to stream large amounts of data of any type. The testing involved not only load testing the machine to establish how high a load the machine can sustain but also testing the actual disk I/O speeds to measure the performance that can be expected from the box.

1.2 Hardware

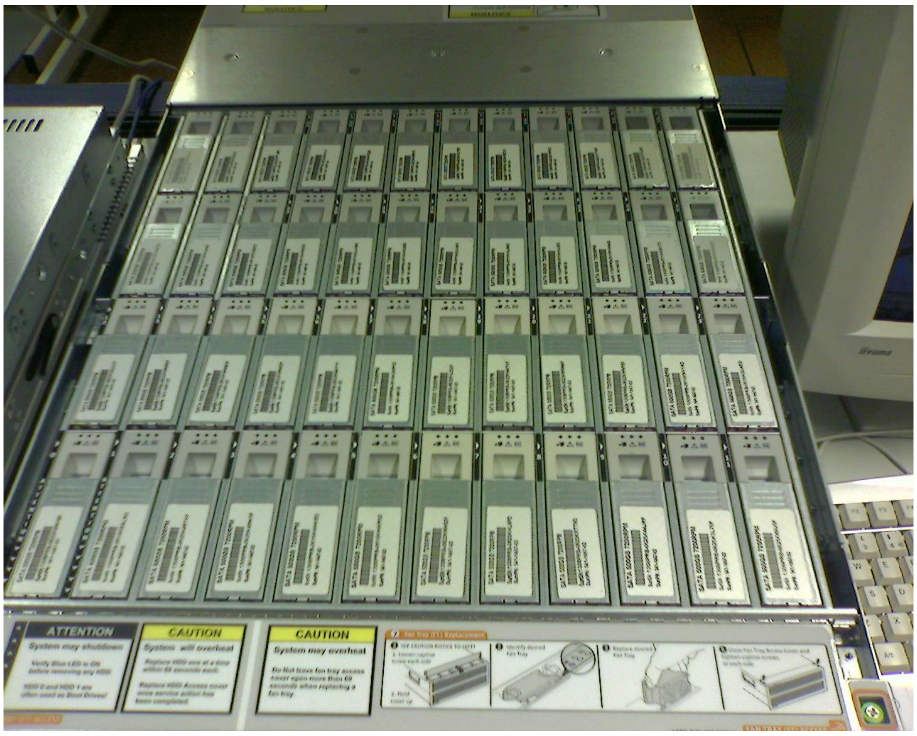

Figure 1. Disks in the X4500

The Sun Fire X4500 file server is a 4U disk server with the following features:

- 48 direct-attached hot-pluggable, 3.5-inch 7200RPM SATA II HDDs (Figure 1)
- 6 x SATA controllers
- 16GB RAM
- 2 x Opteron dual core processors
- 4 x gigabit Ethernet ports

There is no hardware RAID controller in the X4500; it relies solely on the software RAID capabilities of the installed operating system.

1.3 Objectives

The main aim of the testing was to determine the X4500's performance in the following areas:

- Local I/O to a software RAID device under Solaris

- Local I/O to a software RAID device under Linux
- NFS I/O to a software RAID device under Solaris
- NFS I/O to a software RAID device under Linux

2 Testing and results

2.1 Setup

Our test configuration consisted of two system disks for the machine, one containing Scientific Linux 4.4 and one containing Solaris 10 x86. One of these disks was inserted in disk slot 0 at any one time to ensure that the other is not corrupted by accident. The rest of the disks were then available for creating software RAID devices using the relevant operating system tools.

The testing was done using IOzone[2]. A custom suite of disk server benchmarks (section 2.2) was also used to see how the machine performed under high loads.

The tests were run in a number of different configurations in both Linux and Solaris but there were problems when running the tests in Linux. These are discussed in section 3.

2.1.1 Software RAID

2.1.1.1 Linux

Software RAID in Linux[3] is accomplished with the **mdadm** command:

```
# mdadm --create /dev/md0 --level=raid6 --raid-devices=12 device
list
```

/dev/md0 can then be used as the block device for a file system:

```
# mke2fs -b 4096 -E stride=16 /dev/md0
# mkdir /tst
# mount -o noatime,nodiratime /dev/md0 /tst
# df -h /tst
Filesystem Size Used Avail Use Mounted on
/dev/md0 4.5T 109M 4.5T 1 /tst
```

The file system **/tst** is now ready for testing. Note that software RAID devices such as **/dev/md0** above cannot be partitioned so one must use LVM or create multiple, smaller RAID devices.

For this testing we used 12 disks in a RAID6 configuration as in the example above.

2.1.1.2 Solaris

The “software RAID” system in Solaris 10 is called ZFS[4]. Rather than sitting between the file system layer and the raw disks (c.f. Linux software RAID), it is a file system that can span multiple disks and has features like RAID. ZFS is quick and easy to set up with the **zpool** command:

```
# zpool create -f tst raidz device list
```

As well as creating the raid device, it creates the mount point **/tst** and mounts the new zpool device at the mount point.

For this testing we used 9 drives to create **/tst**.

2 http://www.iozone.org
3 http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/sysadmin-guide/s1-raid-approaches.html
4 http://www.sun.com/2004-0914/feature/

2.1.2 NFS client

The client was a single machine with a standard Tier1 worker node installation (Scientific Linux 3.0.8). The X4500 and the client machine were connected via Gigabit Ethernet. **/tst** on the X4500 was exported and mounted at **/tst** on the client. The same test scripts were run as if **/tst** was a local file system.

2.2 Custom benchmark suite

The idea of this suite of programs is to emulate the load on a disk server created by typical physics applications. There are several programs that are run independently and access the disk in different ways, including *AIM VII*[5] and custom disk I/O tests.

The suite was used to generate a large sustained load.

2.3 IOzone

IOzone is a disk I/O testing suite which performs a variety of read and write tests. IOzone was run according to the IOzone documentation[6]. Scripts to run IOzone with the desired command-line options were created and are listed in appendix A.

Several IOzone tests were run using differing numbers of threads but small file sizes to check that IOzone would run OK:

```
    iozone -i0 -i1 -t1 -s512m -r1k
    iozone -i0 -i1 -t1 -s512m -r8k
    iozone -i0 -i1 -t1 -s512m -r16k
    iozone -i0 -i1 -t1 -s512m -r32k
    iozone -i0 -i1 -t2 -s512m -r32k
    iozone -i0 -i1 -t4 -s512m -r32k
    iozone -i0 -i1 -t8 -s512m -r32k
    iozone -i0 -i1 -t16 -s512m -r32k
    iozone -i0 -i1 -t32 -s512m -r32k
```

These tests ran successfully so the scripts in appendix A were used to run IOzone to collect the data for analysis.

5 http://sourceforge.net/projects/aimbench
6 http://www.iozone.org/docs/IOzone_msword_98.pdf

2.4 Results

2.4.1 Solaris, local I/O

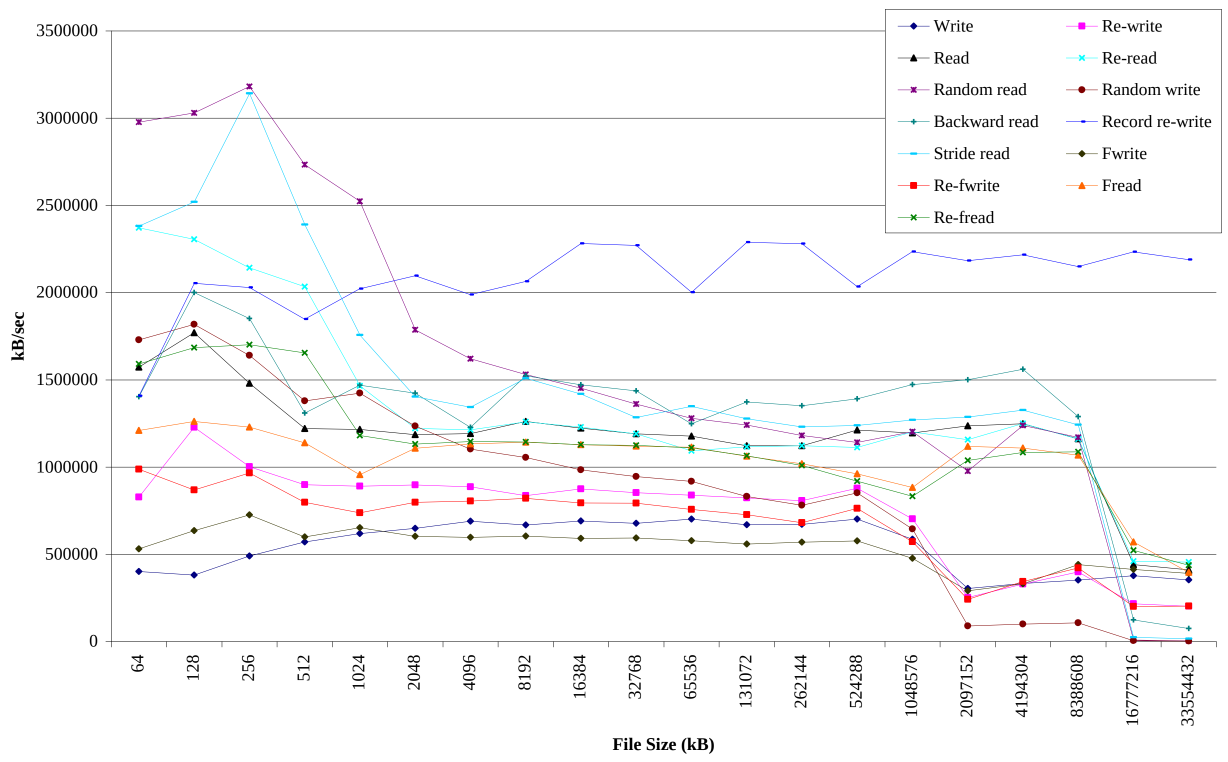

Figure 2. Solaris local data rates

Figure 2 shows how file size affects local data rates under Solaris. For each file size, data was written with records sizes from 4 kB to 64 kB. The data rate quoted is the mean of the rates achieved at each file size.

The best write performance was observed on files of between 4 MB and 512 MB. Within this range, the write speed remained fairly constant at about 700 MB/s. Above 512 MB, the write speed drops by to about 300 MkB/s. At the lower end, data rates start at about 400 MB/s for files of 64 kB and climb gradually to the maximum at a file size of 4 MB. All write tests follow a similar trend.

The read figures follow a similar pattern; rates increase to a maximum of 1.77 GB/s for files of 128 kB, fall again and level out at file sizes of 512 kB to 8 GB. Rates stay high for longer than write rates, only falling off for file sizes over 8 GB. The read rate between 512 kB and 8 GB is about 1.25 GB/s.

2.4.2 Linux, local I/O

The same tests were attempted on Linux but the time spent investigating the problems described in section 3 prevented this. The X4500 under Linux was tested using multi-threaded instances of IOzone to see how disk I/O was affected by the number of threads. A script (thumper-threaded.sh, appendix A.2) was used to run IOzone. The script keeps track of its state in case of a reboot.

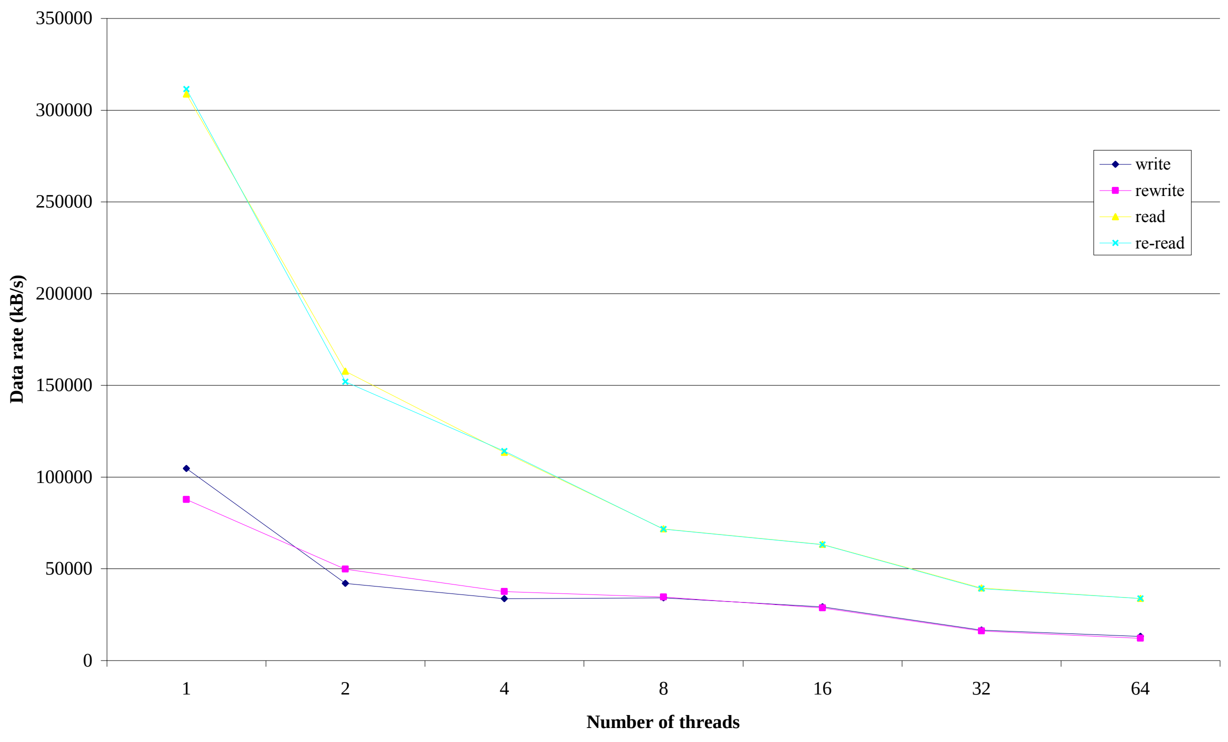

Figure 3. Linux local data rates

Figure 3 shows the results of the multi-threaded IOzone testing in Linux; 1 to 64 threads were running concurrently, each accessing a 32 GB file. The figures quoted are the measured per-thread rate.

As would be expected, the per-thread performance drops with increasing number of threads, both in read and write tests. With a single thread, the read and write rates are about 300 MB/sec and 100 MB/sec respectively. These figures halve for two threads and then start to flatten out. At 64 threads, the per-thread read rate is 33 MB/sec and the write rate is 12 MB/sec.

2.4.3 Solaris, NFS

Figure 4 shows the test results for IOzone via NFS to the X4500 running Solaris. Write rates start at about 8 MB/sec for files of 64 kB and increase to 45 MB/sec for files of 1 MB and remain fairly constant for all larger file sizes up to 8 GB.

Read rates show a very different pattern and are probably artificially high for lower file sizes due to the caches involved in NFS. Once the file size reaches 256 MB, the read rates fall dramatically, probably due to exceeding an NFS cache, either on the client or the server. For file sizes greater than 256 MB, the read rate drops to between 100 and 240 MB/sec.

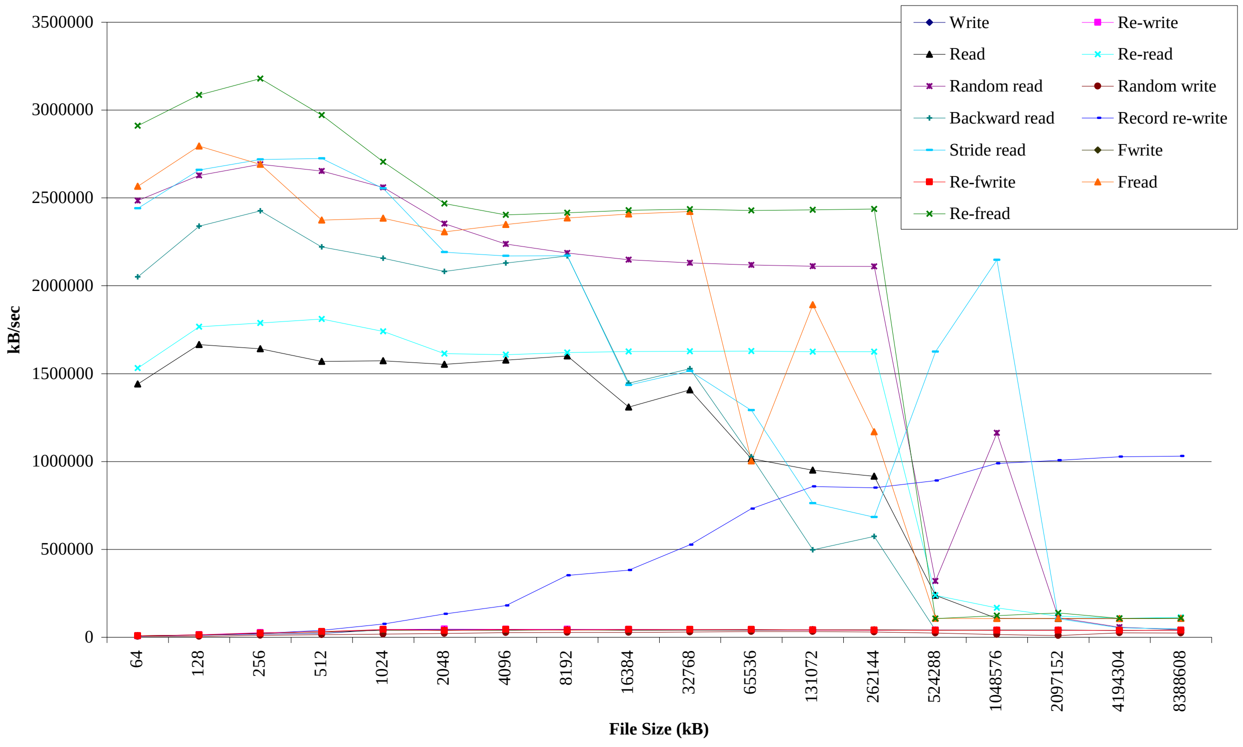

Figure 4. Solaris NFS data rates

2.4.4 Linux, NFS

Figure 5 shows the results of the Linux NFS tests. Per-thread read rates fluctuate around 40 MB/sec and write rates start at about 12 MB/sec for one thread and drop to 3 MB/sec per thread for 32 threads.

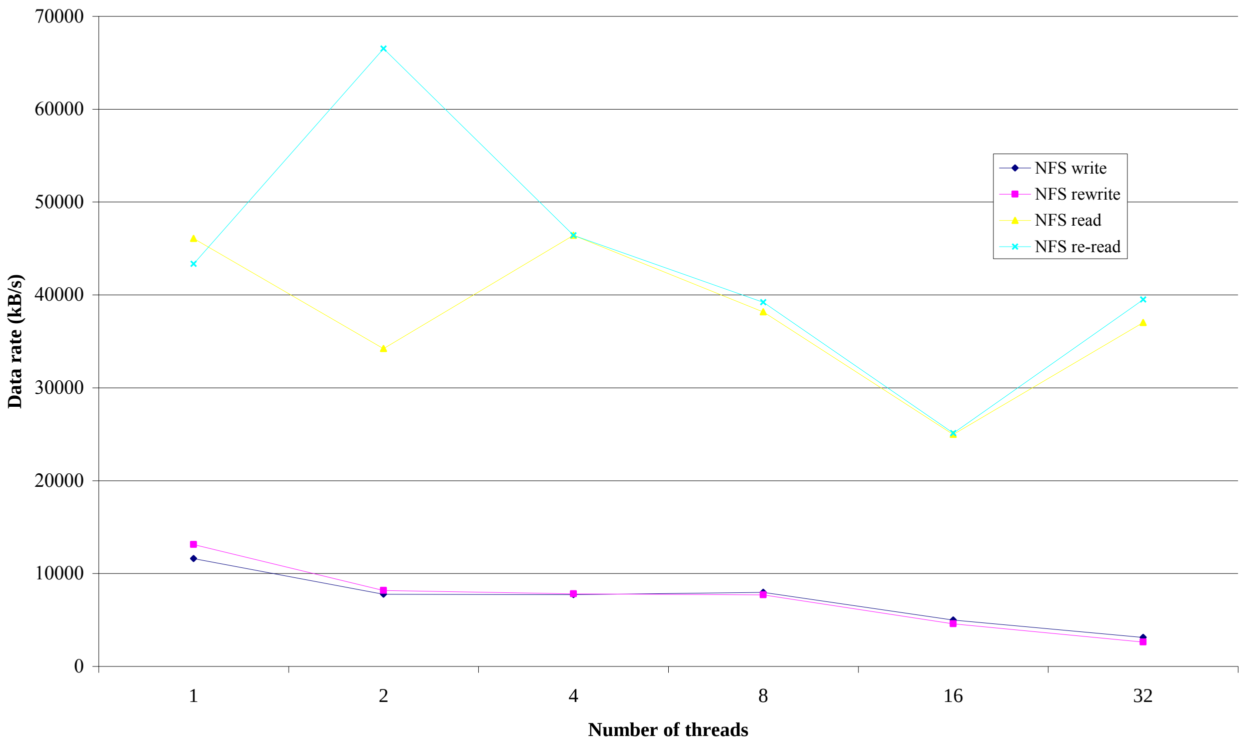

Figure 5. Linux NFS data rates

2.5 Comparisons

Time spent on investigating the problems under Linux and the fact that the tests run on Solaris did not produce results on Linux (the machine rebooted before completing them) meant that the tests run on the two operating systems differed. Comparison of Linux and Solaris is more difficult as a result but some comparisons can be drawn.2.5.1 Local I/O

The data rates for a single thread accessing a 32 GB file can be compared (Table 1). This suggests that for 32 GB files the X4500 is 46% faster on reads and 255% faster on writes, despite the I/O being split across nine rather than twelve disks.

Table 1

	Read (MB/sec)	**Write (MB/sec)**
Linux	309	105
Solaris	452	373

2.5.2 NFS

It is possible to compare the single thread, 8 GB file figure (Table 2). Again, the X4500 seems to perform better under Solaris with NFS reads being 132% faster than Linux and NFS writes being 233% faster. The Linux figures may not be wholly representative, however. They may be skewed by the client waiting for the server to reboot due to the problems described in section 3. Differences in the way that the two operating systems handle NFS caches may also have affected the results.

Table 2

	Read (MB/sec)	**Write (MB/sec)**
Linux	46	12
Solaris	107	40

Some NFS tests were run on both Linux and Solaris from the same NFS client. These measured the mean read and write rates for a number of file sizes up to 256 MB. Due to the small size of the files, the read figures were broadly similar due to caches on the client (figure 6) and so do not agree with table 2 which is for 8 GB files. It is worth noting that Solaris drops below Linux above files of 8 MB. The reason for this was not determined.

The write figures are more useful as they are not affected by caches (figure 7). Here Solaris clearly performs better than Linux with figures broadly similar to table 2.

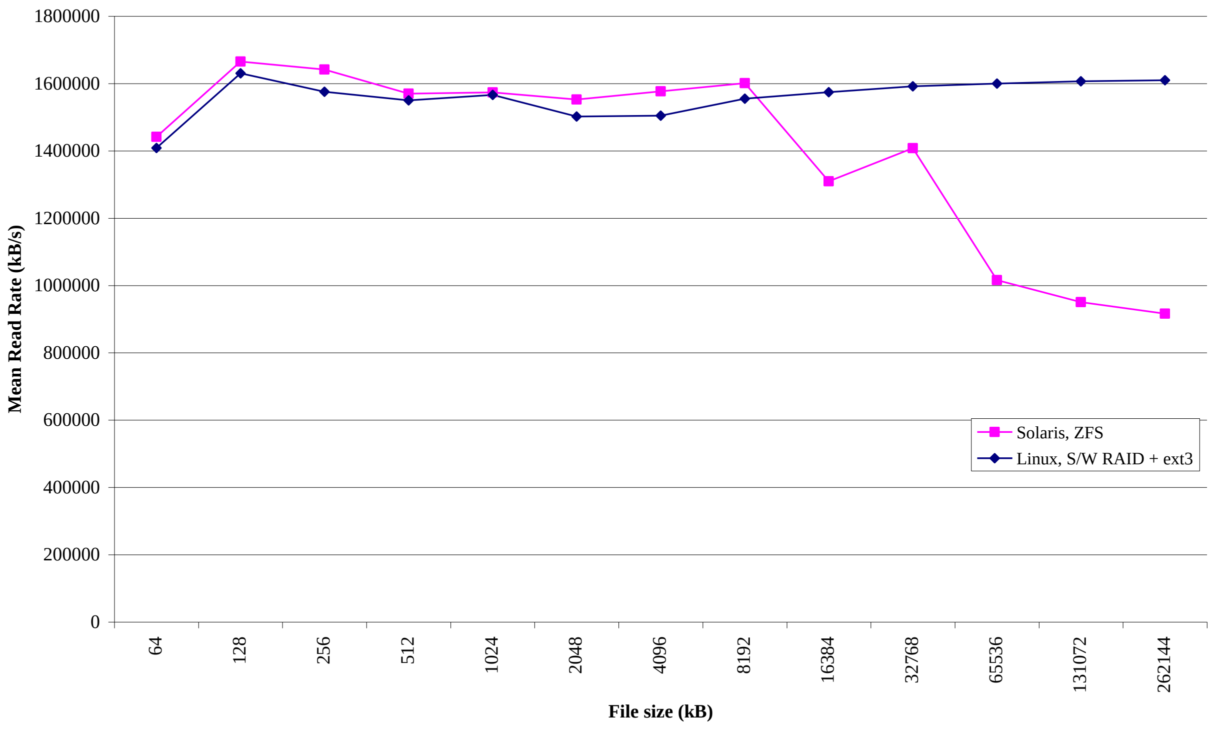

Figure 6. Mean NFS read performance

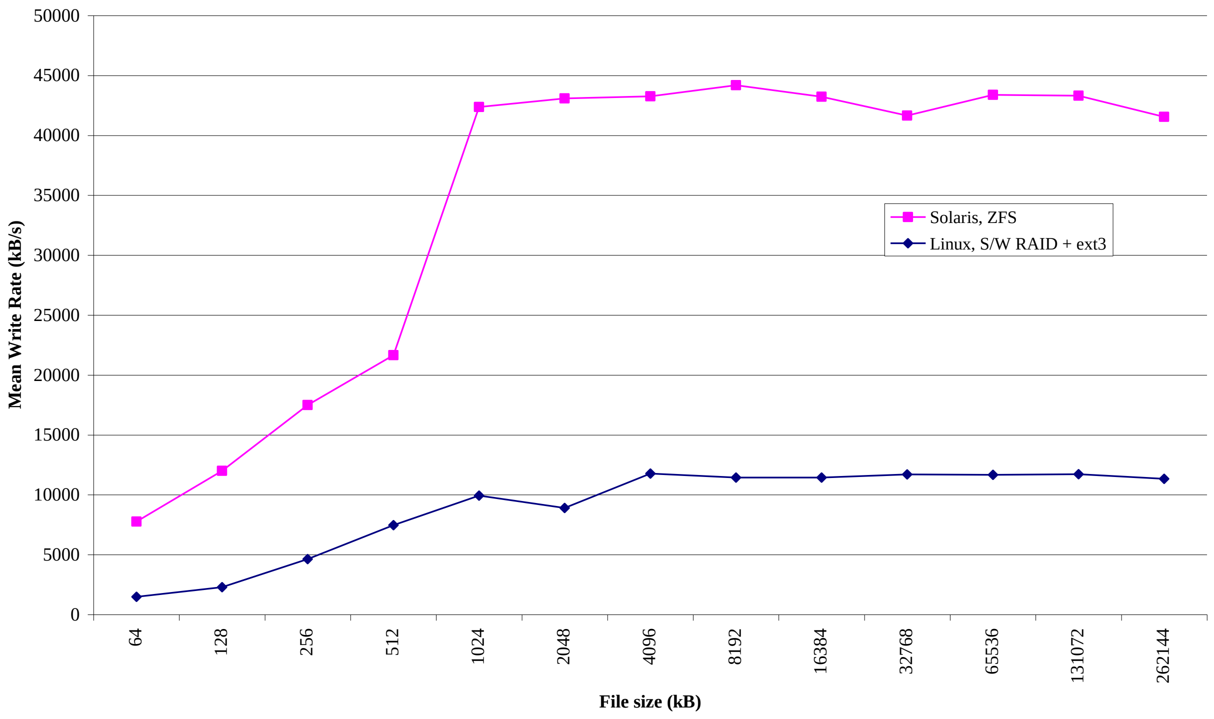

Figure 7. Mean NFS write performance

3 Problems

3.1 Spontaneous rebooting

There was a major problem when testing the X4500 under Linux. Early on, during load testing with the custom disk server test suite, the machine rebooted with no log messages in **/var/log/messages**.

These tests should have been easily accomplished by the X4500. During the POST, the message displayed was:

```
    Hyper Transport sync flood error occurred at last boot
```

Looking at the logs on the Integrated Lights Out Manager (ILOM) with **ipmitool** shows this problem logged just after system boot:

```
    # ipmitool -H IP address -U root -P password -I lanplus sel list
    a801 | 01/16/2007 | 14:16:01 | System Firmware Progress | System \
    boot initiated | Asserted
    a901 | 01/16/2007 | 14:38:43 | OEM #0x12 | | Asserted
    aa01 | OEM record e0 | 00000000040f0f0700000000f2
    ab01 | OEM record e0 | 010000000480c0800308000000
    ac01 | 01/16/2007 | 14:39:45 | System Firmware Progress | \
    Motherboard initialization | Asserted
    ad01 | 01/16/2007 | 14:39:45 | System Event | \
    Undetermined system hardware failure | Asserted
    ae01 | OEM record e0 | 00000000040f0f0700000000f2
    af01 | OEM record e0 | 00000100040f0f0700000000f2
```

This happened only once during testing.

3.2 Kernel panic

On one occasion, there were some errors early in the morning, followed by a kernel panic. There was the following in **/var/log/messages**:

```
    Feb 11 04:02:04 csflnx357 kernel: PCI-DMA: Out of IOMMU space for \
    131072 bytes at device 0000:06:01.0
    Feb 11 04:02:04 csflnx357 kernel: ata25: status=0x50 \
    { DriveReady SeekComplete }
    Feb 11 04:02:04 csflnx357 kernel: ata25: error=0x50 \
    { UncorrectableError SectorIdNotFound }
    Feb 11 04:02:04 csflnx357 kernel: Info fld=0x505050, \
    Current sdy: sense key No Sense
```

3.3 Investigation

The *Error Handling* section of the *Sun Fire X4500 Server Diagnostic Guide*[7] suggest that the HyperTransport sync flood error message can be caused by one of several things:

- The CPU detected an un-correctable multiple-bit DIMM error.
- A CRC or link error on one of the HyperTransport links.
- A system or parity error on a PCI bus.

The testing tools do not appear to be the cause as the machine rebooted in this way even when idle. Also, the fault does not appear when running Solaris which leads to the possibility that the fault is the way in which Linux is interacting with the hardware or the hardware itself.

Memtest86+ was run on the box for over 24 hours and after several passes had found no faults with the memory. Other memory testing tools were tried too, including Stream[8] and Memtester[9]. Neither of these tools revealed any problems.

The kernel panic suggested that the available IOMMU space may need to be increased or that there was a problem with **/dev/sdy** (the system disk). **/dev/sdy** passed an **fsck** check. The amount of IOMMU space available can be changed using kernel options. The IOMMU was disabled in the machine BIOS and the machine was booted with the kernel parameter **iommu=memaper=4** and the custom disk testing suite was run.

7 http://docs.sun.com/source/819-4363-10/error_handling.html#88475
8 http://www.cs.virginia.edu/stream/
9 http://pyropus.ca/software/memtester/

With this parameter, the machine stayed up for four days with load averages of 600 to 900, with a peak at 1500. The setting of the IOMMU aperture seemed to have fixed both the panic and the spontaneous reboots. The machine was rebooted, **/tst** was recreated and the tests were started again (section 2). The X4500 rebooted after just over a week. The kernel parameter **iommu=memaper=5** was tried. Although the machine stayed up for nearly three weeks using this parameter it rebooted again.

After the X4500 had been returned, an article was found on a Sun Internet forum[10] which described the IOMMU problem on the X4500. It suggests replacing the operating system provided *sata_mv* driver with *mv_sata*[11] or using the *sata_mv* included in the latest 2.6 kernel.

A source RPM for the *mv_sata* driver was provided by Sun during the testing. This was built and tried but did not seem to work. It is likely that this was due to a subsequent kernel update without rebuilding the *mv_sata* driver for the new kernel. With hindsight, it seems insufficient time was spent researching this driver at the time. If one of these machines were to be tested again then more time should be spent building and researching this driver.

4 Conclusions

The performance of the X4500 was generally good; however the problems encountered while running Scientific Linux 4.4 caused difficulties during testing. There is a fix for this which would allow better testing of the X4500 if the Tier1 was able to obtain one again for further testing.

From the results that were obtained, it appears that Solaris and ZFS are much quicker than Linux and Software RAID6. The Linux figures may have been adversely affected by repeated background rebuilds of the software RAID device due to the repeated rebooting of the machine. Using Solaris rather than Linux would, however, require major changes to the Tier1 fabric and configuration system, as well as provision of support for the disk server software on Solaris x86. The Tier1 may not have the resources to do this.

If the problems encountered under Linux are fixed by the different SATA driver then the X4500 would be a candidate for use as a disk server in the Tier1. With 16 GB RAM and powerful CPUs, the system was very responsive under high load and may be able to perform other tasks, such as calculating checksums, while serving data.

10 http://forum.java.sun.com/thread.jspa?threadID=5105877&tstart=120
11 http://www.keffective.com/mvsata/

A Scripts

A.1 thumper-solaris.sh

```
#!/bin/sh
# thumper-solaris.sh
#
# A script to run iozone on the thumper.
#
# this is where our executable is
#
cd iozone.solaris/
# run iozone. See below for a translation
#
./iozone -azRQ -f /tst/iozonefile -U /tst -b \
/root/thumper-iozone-local-solaris.xls \
-q 64m -s 32g > /root/thumper-iozone-local-solaris.log
# -a = "automatic mode"
# -z = Use record sizes below 64k for large files too.
# -R = Produce Excel report
# -Q = Create offset/latency files
# -f = The filename to use for the tests
# -U = The filesystem to unmount and mount between each test
# -b = The Excel file to write the output to
# -q = The maximum record size (64MB here)
# -s = The maximum file size (2xRAM = 32 GB here)
#
# The output to STDOUT is tabular data for plotting if necessary.
```

A.2 thumper-threaded.sh

```
#!/bin/bash
# thumper-threaded.sh
#
# A script to run iozone on the thumper.
#
# Before a new run you might want to delete the state file defined
for
# $STATEFILE below.
#
# On the Thumper, this should go in /etc/rc.local so that it starts
on boot
# if we have a reboot.
#
# Do you want debugging? 0 = no, 1 = yes
DEBUG=1
# Setup $PATH, just in case
#
PATH=$PATH:/bin
# Set up some variables
#
# FILESIZE should ideally be twice the RAM of the machine.
FILESIZE=32g
IOZONEDIR=/root/iozone.thumper/iozone3_281/src/current
TESTDIR=/tst
IOZONE_FILENAME=$TESTDIR/iozonefile
OUTFILE=/root/thumper-iozone-threaded-20070330
STATEFILE=/root/thumper-iozone-threaded-20070330.state
#STATEFILE=/home/ron/thumper-iozone-threaded.state
# this is where our executable is
#
cd $IOZONEDIR
# Run iozone. Translation of options.
# -a = automatic mode
# -R = Produce Excel report
# -i0 = run write/rewrite tests
# -i1 = run read/re-read tests
# -F = The filenames to use for the tests
# -U = The filesystem to unmount and mount between each test
# -b = The Excel file to write the output to
# -t = The number of threads to run
# -s = The test file size (2xRAM = 32 GB here)
#
# The output to $LOGFILE is tabular data for plotting if necessary.
#
# This doubles the number of threads with each iteration. The
setting of
# $IOFILE is necessary as we're remounting the filesystem between
tests to clear
# the caches.
# Touching the state file ensures that it exists and avoids errors
from cat.
# Decalre $STATE as an integer variable
touch $STATEFILE
declare -i STATE=`cat $STATEFILE`
# Loop through the different numbers of threads that we want to run
for THREADS in 1 2 4 8 16 32 64 128; do
        # Check how far we got the last time we were run. We don't
want to run
        # all the tests again. If you do, remove the state file and
re-run
        if [ $THREADS -gt $STATE ]; then
                # Set $IOFILE to contain a list of filenames for the
threads
                # to use as temp files and remove them.
                IOFILE=""
                for i in `seq 1 $THREADS`; do
                        IOFILE="$IOFILE ${IOZONE_FILENAME}${i}"
                        rm ${IOZONE_FILENAME}${i}
                done
                # Print out the command if we're running with
debugging
                if [ $DEBUG -eq 1 ]; then
                    echo "DEBUG: iozone -R -i0 -i1 -t${THREADS} \
-F $IOFILE-b ${OUTFILE}-${THREADS}.xls -s $FILESIZE 2>&1 \
> ${OUTFILE}-${THREADS}.log"
                        echo
                fi
                # Run iozone
                ./iozone -R -i0 -i1 -t${THREADS} -F $IOFILE \
-b ${OUTFILE}-${THREADS}.xls -s $FILESIZE 2>&1 \
> ${OUTFILE}-${THREADS}.log
                # Update the state file
                echo $THREADS > $STATEFILE
        fi
done
## END ##
```

B Preparing for return

Before returning the X4500 to Sun the disks needed to be blanked. Time was limited so it was decided to blank as many disks as possible in parallel:

```
# for i in a b c d e f g h i j k l m n o p q r s t u v w x z; do
> dd if=/dev/zero of=/dev/sd${i}
> done
```

However, this method caused the same hypertransport error discussed in section 3.1 and was abandoned. Instead, the first gigabyte of each data disk was overwritten along with the whole of both system disks. The machine was collected for return to Sun on Tuesday 10th April 2007.