

Results from the *read-30* test

Adam Lyon, August 1, 2005

1 Introduction

A SAM test was performed to determine the rate of file deliveries possible on a test CAF system.

2 Test Particulars

The test was started at 5pm on Friday July 29, 2005 and ended at 1:10pm on Saturday July 30, 2005.

The test consisted of submitting segments that looped 40 times; each loop consisted of a python `sam.getNextFile` python call, waiting for the file URL to be delivered, waiting 30 seconds to simulate processing, and finally releasing the file with a `sam.releaseFile` call. Since the "processing time" was 30 seconds, the name of this test is *read-30*.

The test CAF (`--farm=fncdf`) was used and consisted of 925 virtual machines. Twenty datasets were created, each containing about 1/20th of the `xbhd0d` golden dataset (full dataset contains 42,122 files). The script `/cdf/scratch/cdfopr/sam-testing/makeDefs.py` was used to make the datasets. Then 20 jobs were submitted (one per dataset), each consisting of fifty segments. So 1000 segments of 40 files means 40,000 files should have been requested.

The jobs were submitted using script `/cdf/scratch/cdfopr/sam-testing/doSubmit.py` which calls `submit-mimic-caf-usage-fncdf.sh` (written by Doug) that submits the job to the test CAF.

For these tests we used `sam v7_2_0` against the `user_int` universe. Note that we used our own `CafSubmit` script that turns off a very slow data set validation (meant to warn the user if the majority of files in their dataset are not cached - since we know all of our files are in the cache, this check is not necessary).

Jobs were run with job logs copied to `/cdf/scratch/cdfopr/sam-testing/read-30`. A script there called `parseLogs.py` opens the tar files, extracts the log file and processes it forming a new large file with a sum of information from each log. This large file contains the get next file request times, the time it took for the file to be delivered, and the time of the file release.

Other logs were used to monitor different aspects of the system. This will be covered when they are analyzed.

3 Results

The test was a success in that the vast majority of files were successfully delivered within about 20 hours. There are some interesting effects to note, however.

3.1 Results from job output logs

See Appendix A.2 for the code that produces the results in this section.

There were 39281 successful file deliveries over approximately 20 hours. Note that 14 segments crashed with an untrapped unknown exception generated by `sam.releaseFile`. This caused 191 files to be missed due to the crashed segments. I don't know why this doesn't add up to 40,000 [will need to check - maybe some segments never started].

The deliveries amount to about 50,000 files delivered per day. But note that an obvious 9 hours service outage makes this value smaller than it should have been.

3.1.1 Request and Delivery Rates

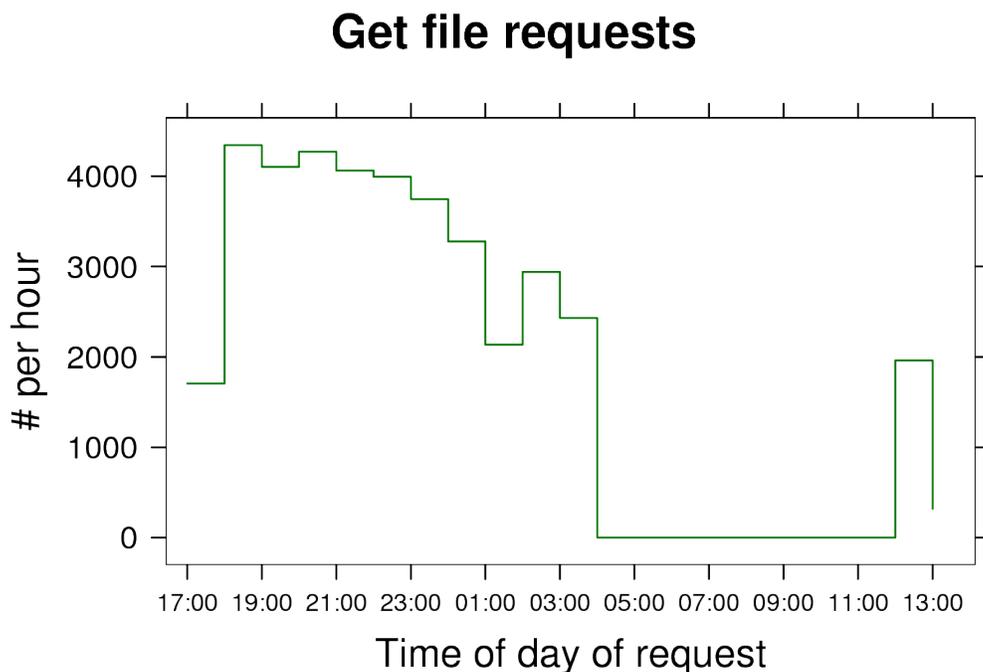


Fig. 1. Get Next File request rate

The rate of `sam.getNextFile` per hour is shown in Fig. 1. We were able to achieve a rate of between 2000 and 4000 requests per hour, or between 0.5 requests per second and a little more than one request per second.

The file delivery rate per hour (when the URL actually arrived to the waiting segment) is shown in Fig. 2.

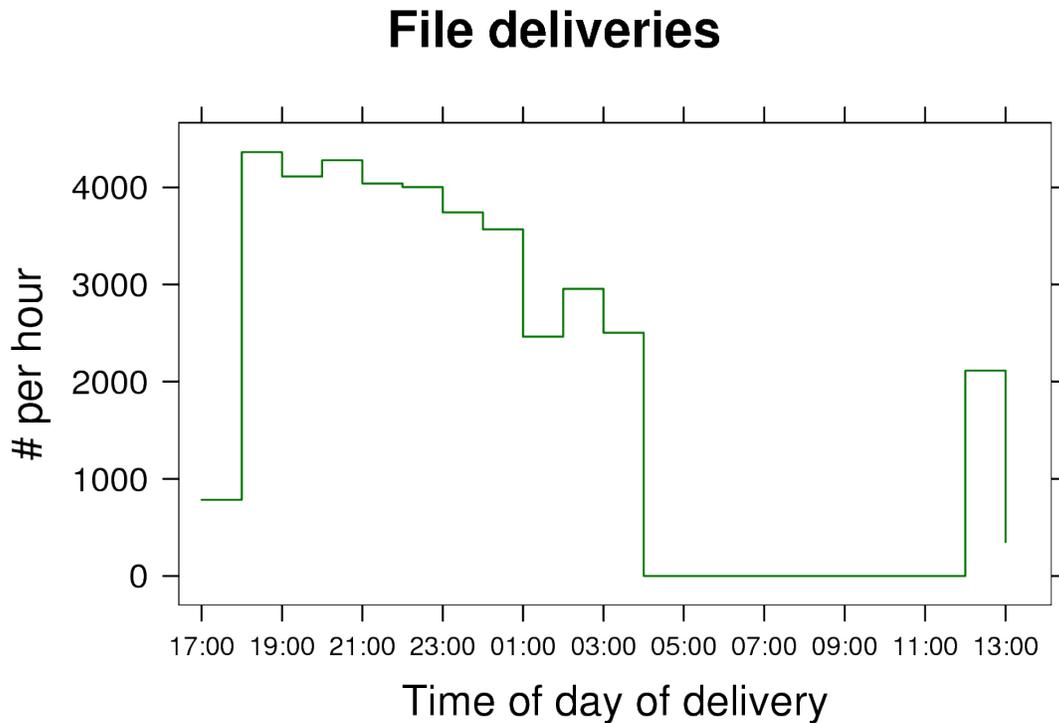


Fig. 2. File Delivery Rate

In both plots, one notices an obvious service outage between 4am and noon on Saturday, July 30, 2005. Must figure that out.

3.1.2 Delivery wait times

The job log files contain how long each Get Next File request had to wait for its file to be delivered. These results are shown in various ways below.

File delivery waits

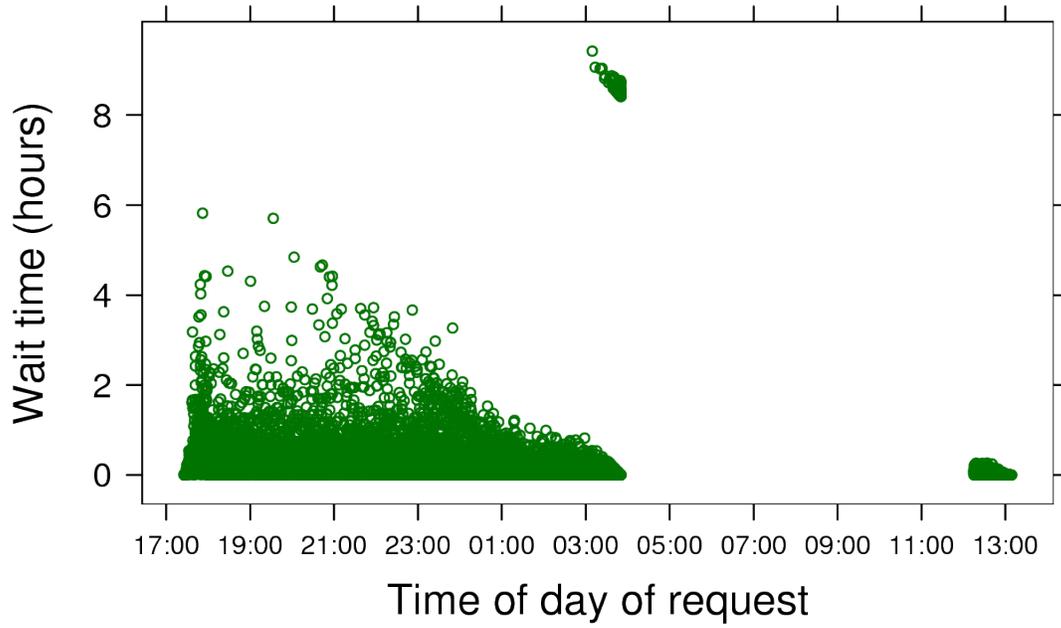


Fig. 3. The wait times for file deliveries vs. the time of the request.

File delivery waits

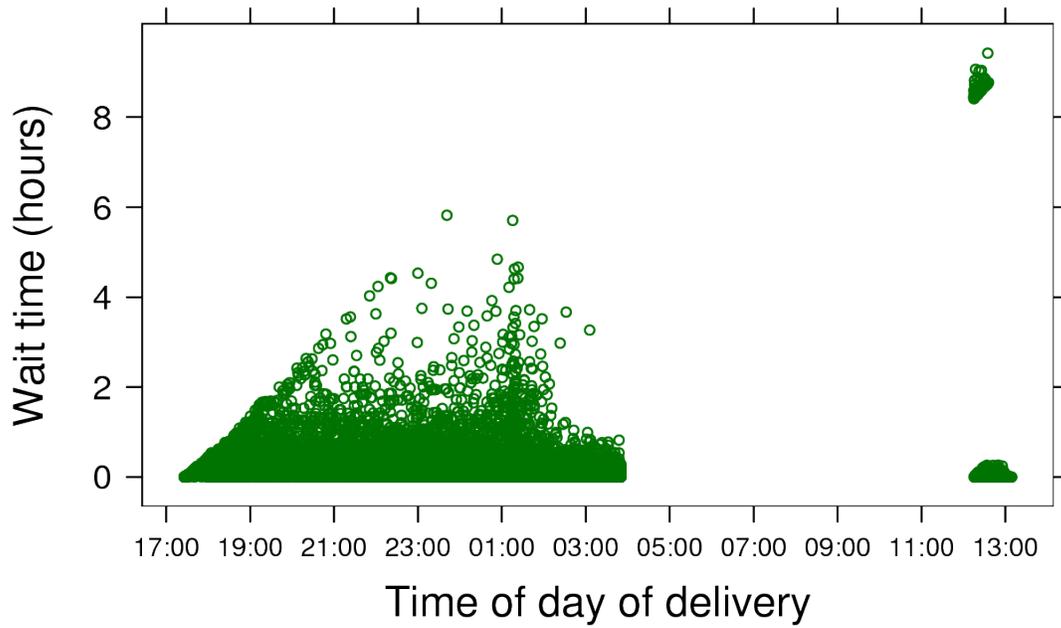


Fig. 4. The wait times for file deliveries vs. the delivery time.

Fig. 3 and Fig. 4 show the delivery wait time against the time of the request and time of the delivery respectively. Note that these plots really only show off the outliers and it is impossible to see the average wait times. But one can see that some of the outliers are quite large, even at the very beginning of the test. This needs to be understood.

A more useful plot is the median wait time in hour blocks as shown in Fig. 5. Here the median is used as outliers affect it less than a mean (but see the appendix for that plot). Note that the empty part of the plot is where there were no deliveries during the mystery outage.

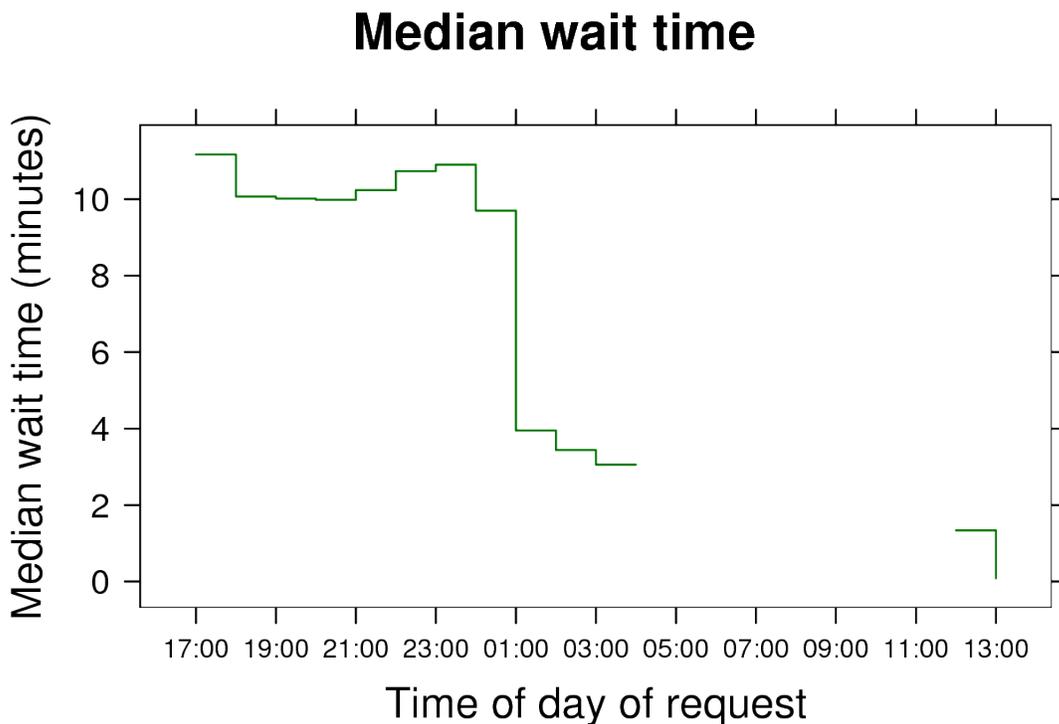


Fig. 5. Median file delivery wait time (minutes) in hour blocks.

One sees that the median wait time is a maximum of eleven minutes and improves dramatically at about midnight.

Fig. 6 shows the fraction of deliveries that took over an hour to complete. The maximum is about 8% and occurs just before the service outage.

Percent long wait times

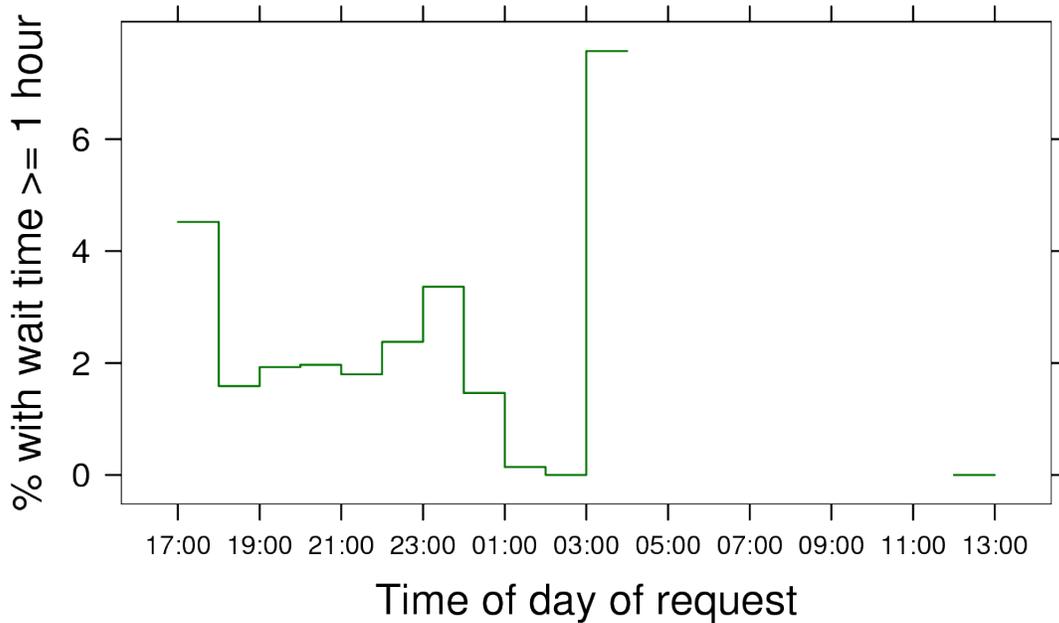


Fig. 6. The fraction of delivery times that took over an hour.

From the job log files, it is possible to determine the number of unfulfilled file delivery requests that were "queued" at any given time. This information is shown in Fig. 7. Had file delivery been very fast (faster than the get next file request rate), then this plot should have averaged a little above zero. But because file delivery is significantly slower than the rate of new requests, one sees them piling up. There comes a point, at about midnight, when the file delivery rate speeds up for some reason and causes the unfulfilled request count to fall. Would be interesting to figure out what caused that! And again one sees the service outage when no requests were fulfilled.

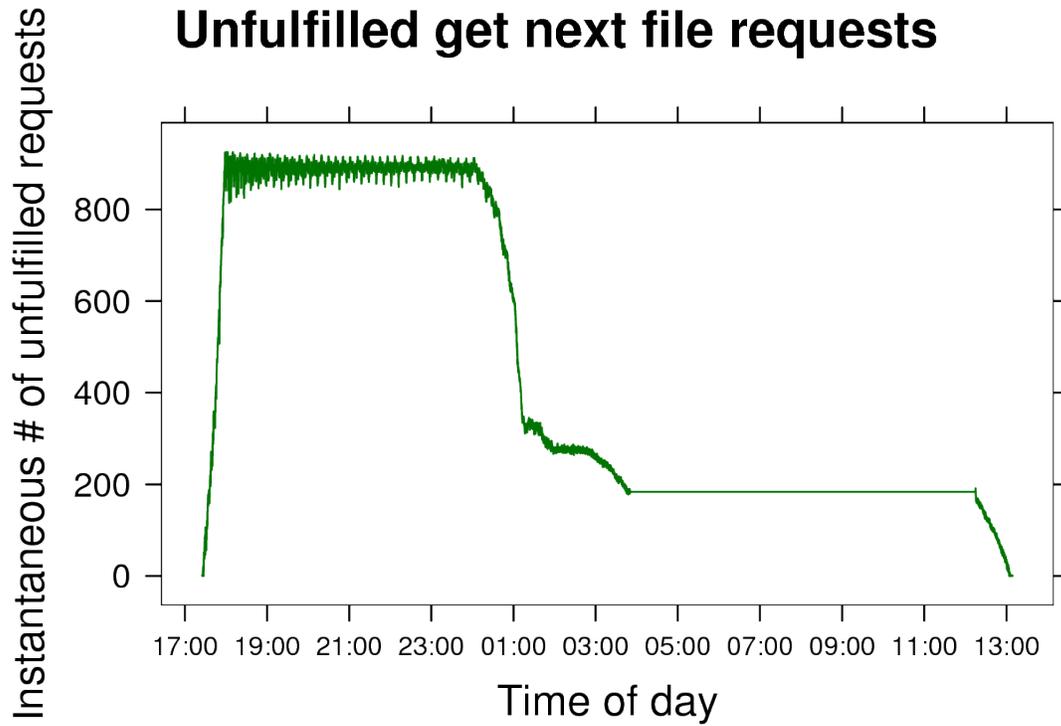


Fig. 7. The instantaneous number of unfulfilled file delivery requests.

More to come.

Appendix A R Session

R is an open source statistical analysis software package that allows for very easy analysis of data in databases and text files. I wrote a "notebook" style package that allows one to use R from within Microsoft Word. Below is the notebook providing all of the code and results for this document.

A.1 Connect to R and initialize

Connect to R running locally on my laptop.

```
<R0> #connect port 6101 timeout 20
R is using work directory /Users/adam/work/projects/cdfSamTests/read-30
```

Set up graphics

```
<R1> library(lattice)
```

```
<R2> trellis.par.set(col.whitebg())
```

```
<R3> fontsize = trellis.par.get("fontsize"); fontsize$text=16 ;
      fontsize$points=6 ; trellis.par.set("fontsize", fontsize)
```

I have a helper function that makes putting graphics into Word easy.

```
<R4> mp
function (plotExpr, file, height = 7, width = 7, res = 72 * 3)
{
  bitmap(file, "pngalpha", height = height, width = width,
         res = res, pointsize = 10)
  r = eval(plotExpr)
  if (class(r) == "trellis")
    print(r)
  invisible(dev.off())
}
<environment: namespace:RemoteRSOAP>
```

A.2 Running job output

Doug's python script loops over getting the next file, waiting thirty seconds to simulate processing, and then releasing the file. A log file records the time of the get next file, the duration of the command, the pnfs name of the file, and the time and duration of the release file command.

Let's read this information into R.

```
<R5> d = read.table("out.log", header=T)
```

```
<R6> d[1:5,]
```

```
  job segment   getDate   getTime getDur fileNum
1  245       1 2005-07-29 17:25:19 25.573      1
2  245       1 2005-07-29 17:26:15 36.284      2
3  245       1 2005-07-29 17:27:21 40.078      3
4  245       1 2005-07-29 17:28:31 66.103      4
5  245       1 2005-07-29 17:30:08 69.070      5
```

```
pnfs
```

```
1
```

```
dcap://cdfdca1.fnal.gov:25144/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ23/
GJ2365/GJ2365.0/xd025f02.03c6bhd0
```

```
2
```

```
dcap://cdfdca1.fnal.gov:25144/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ24/
GJ2422/GJ2422.0/xd026062.0380bhd0
```

```
3
```

```
dcap://cdfdca1.fnal.gov:25144/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ26/
GJ2657/GJ2657.0/xd0260f7.0093bhd0
```

```
4
```

```
dcap://cdfdca1.fnal.gov:25144/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ27/
GJ2791/GJ2791.0/xd026110.01fabhd0
```

```
5
```

```
dcap://cdfdca1.fnal.gov:25144/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ27/
GJ2791/GJ2791.0/xd026110.01debhd0
```

```
  relDate relTime relDur
1 2005-07-29 17:26:14 0.503
2 2005-07-29 17:27:21 0.091
3 2005-07-29 17:28:31 0.111
4 2005-07-29 17:30:08 0.095
5 2005-07-29 17:31:47 0.106
```

Convert dates and times into POSIX times that R can deal with...

```
<R7> d$getDateTime = paste(d$getDate, d$getTime)
```

```
<R8> d$relDateTime = paste(d$relDate, d$relTime)
```

```
<R11> d$getPTime = as.POSIXct( strptime( d$getDateTime, "%Y-%m-%d
%H:%M:%S" ) )
```

```
<R12> d$relPTime = as.POSIXct( strptime( d$relDateTime, "%Y-%m-%d
%H:%M:%S" ) )
```

We can make the delivery time from the get time plus the duration

```
<R15> d$delPTime = d$getPTime + d$getDur
```

What is the range of the test?

```
<R18> testEdges = c(min(d$getPTime), max(d$relPTime, na.rm=T)) ;
  testEdges
[1] "2005-07-29 17:25:19 CDT" "2005-07-30 13:09:22 CDT"
<R19> diff(testEdges)
Time difference of 19.73417 hours
<R20> prettyEdges = c(testEdges[1]-60*60, testEdges[2]+60*60)
```

A.2.1 Basics

How many files deliveries were attempted?

```
<R42> nrow(d)
[1] 39281
```

How many failed on the get end?

```
<R43> sum(is.na(d$getDur))
[1] 0
```

How many failed on the release end?

```
<R44> sum(is.na(d$relDur))
[1] 14
```

```
<R106> #var successfulDeliveries = nrow(d)
39281
```

How many files were missed?

```
<R107> lastNum = d$fileNum[ is.na(d$relDur) ]
```

```
<R111> sum(40-lastNum)
[1] 191
```

```
<R112> #var nMissed = sum(40-lastNum)
191
```

This means that these jobs with failures crashed -- meaning that some files never got asked for.

A.2.2 Number of gets and deliveries

Let's count up how many get file requests and deliveries there were per hour

```
<R37> getsPerHourCuts = cut( d$getPTime, "hours" )
```

```
<R38> delsPerHourCuts = cut( d$delPTime, "hours" )
```

```
<R24> getsPerHour = tabulate(getsPerHourCuts)
```

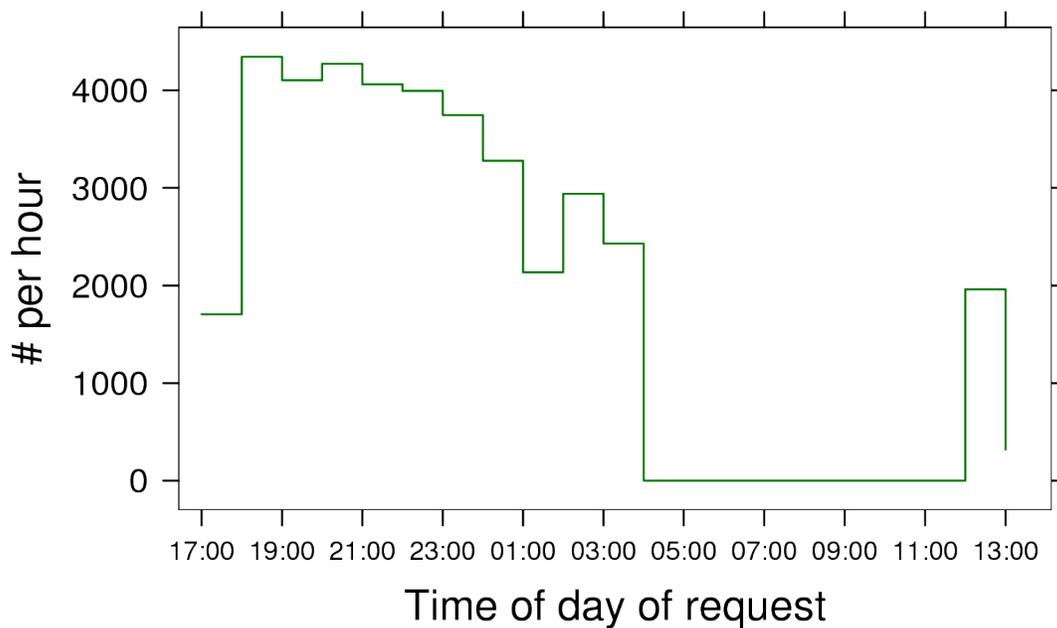
```
<R25> delsPerHour = tabulate(delsPerHourCuts)
```

```

<R47> mp(
  xyplot( getsPerHour ~ as.POSIXct(levels(getsPerHourCuts)),
    main="Get file requests",
    xlab="Time of day of request",
    ylab="# per hour", type="s", xlim=prettyEdges,
    scales=list(x=list(tick.number=10, cex=0.6))
  ),
  "getsPerHour.png", h=4, w=6
)
#with graphics getsPerHour.png timeout 60

```

Get file requests



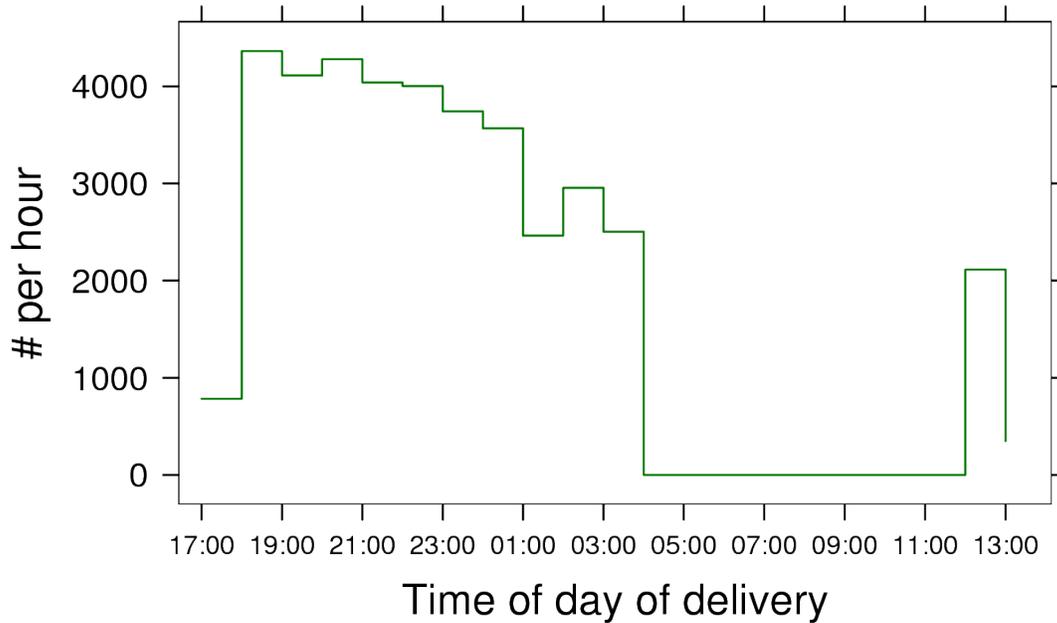
Hmmm - no file requests were made between about 4am and noon on Saturday.

```

<R45> mp(
  xyplot( delsPerHour ~ as.POSIXct(levels(delsPerHourCuts)),
    main="File deliveries",
    xlab="Time of day of delivery", xlim=prettyEdges,
    ylab="# per hour", type="s",
    scales=list(x=list(tick.number=10, cex=0.6))
  ),
  "delsPerHour.png", h=4, w=6
)
#with graphics delsPerHour.png timeout 60

```

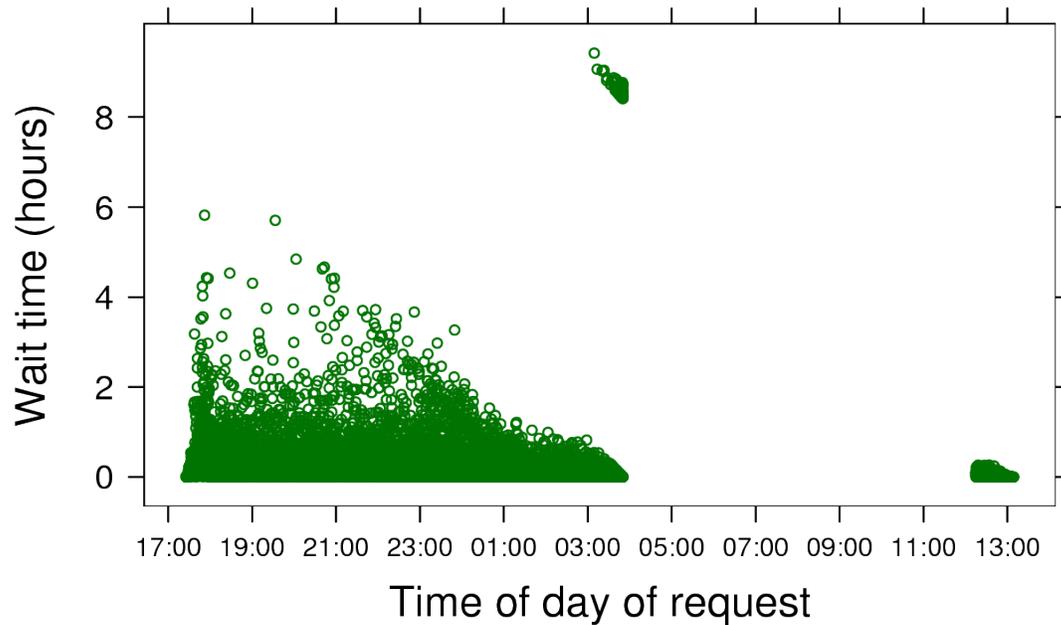
File deliveries



What do the wait times look like?

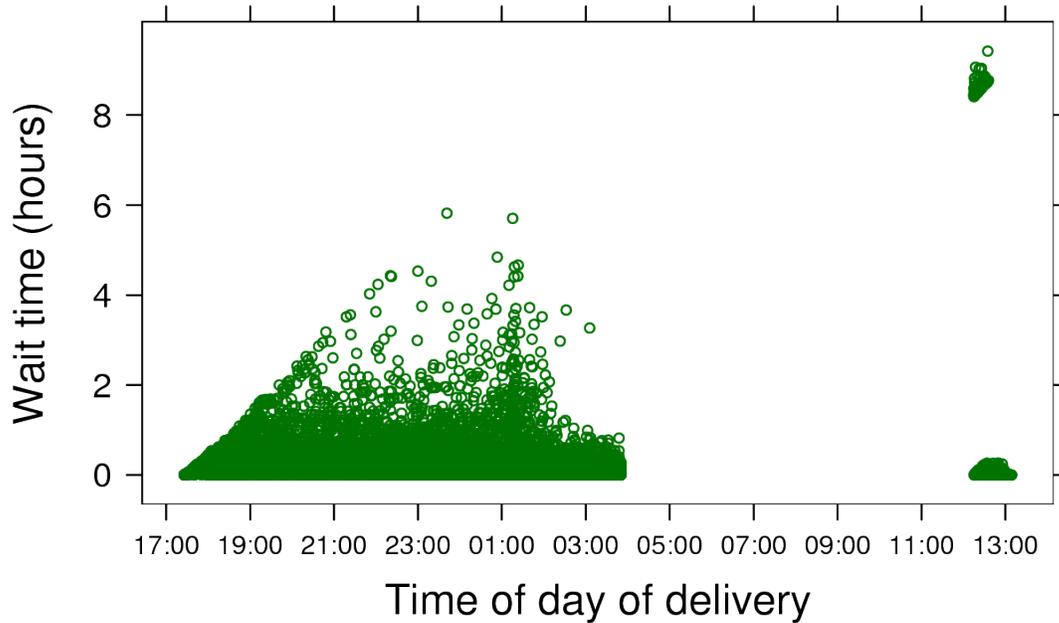
```
<R57> mp(  
  xyplot( getDur/60/60 ~ getPTime, data=d,  
    main="File delivery waits",  
    xlab="Time of day of request", xlim=prettyEdges,  
    ylab="Wait time (hours)",  
    scales=list(x=list(tick.number=10, cex=0.6))  
  ),  
  "getWaits.png", h=4, w=6  
)  
#with graphics getWaits.png timeout 60
```

File delivery waits



```
<R59> mp(  
  xyplot( getDur/60/60 ~ delPTime, data=d,  
    main="File delivery waits",  
    xlab="Time of day of delivery", xlim=prettyEdges,  
    ylab="Wait time (hours)",  
    scales=list(x=list(tick.number=10, cex=0.6))  
  ),  
  "delWaits.png", h=4, w=6  
)  
#with graphics delWaits.png timeout 60
```

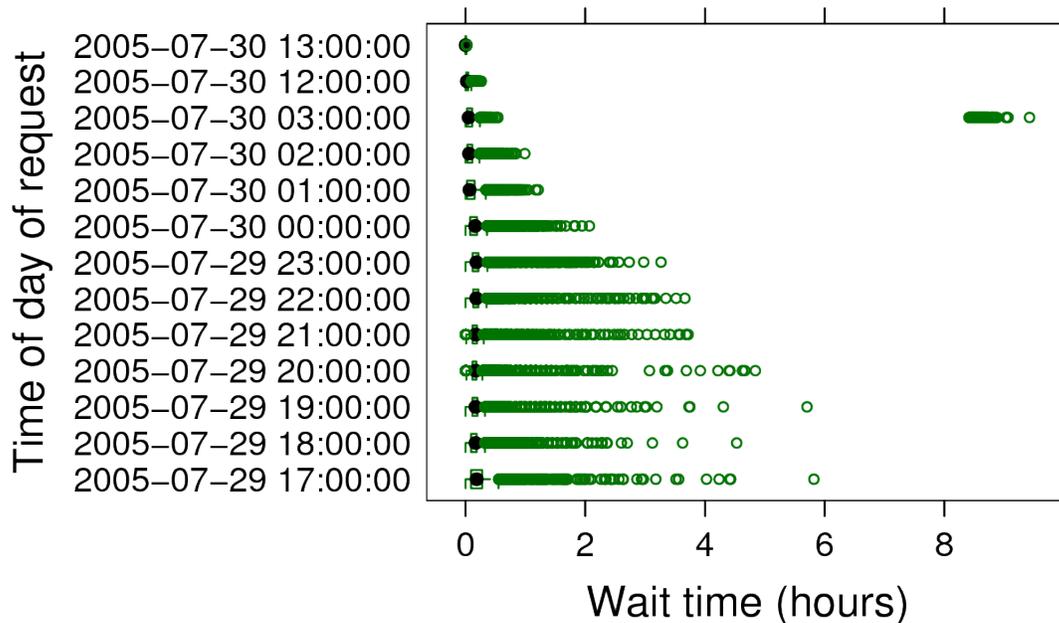
File delivery waits



These plots make the outliers really stand out and hard to see the mean wait times. Let's try to plot those...

```
<R60> mp(
  bwplot( getsPerHourCuts ~ getDur/60/60, data=d,
    main="File delivery waits",
    xlab="Wait time (hours)", ylab="Time of day of request"
  ),
  "waitsPerHourBw.png", h=4, w=6
)
#with graphics waitsPerHourBw.png timeout 60
```

File delivery waits



Again, the outliers dominate the plot. Let's just plot medians and how many are over an hour...

```
<R65> waitMedians = tapply(d$getDur/60, getsPerHourCuts, median)
<R80> waitMeans = tapply(d$getDur/60, getsPerHourCuts, mean)

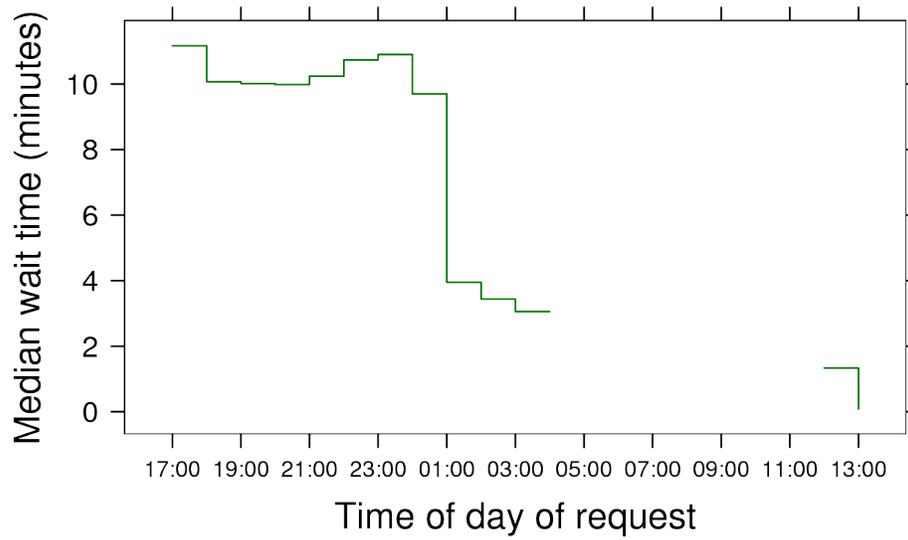
<R67> nOverHour = tapply(d$getDur/60/60 >= 1, getsPerHourCuts, sum)
<R71> percOverHour = nOverHour / getsPerHour * 100\ nOverHour =
<R85> nOverHalfHour = tapply(d$getDur/60/60 >= 0.5, getsPerHourCuts,
  sum)

<R86> percOverHalfHour = nOverHalfHour / getsPerHour * 100
```

Let's plot these things

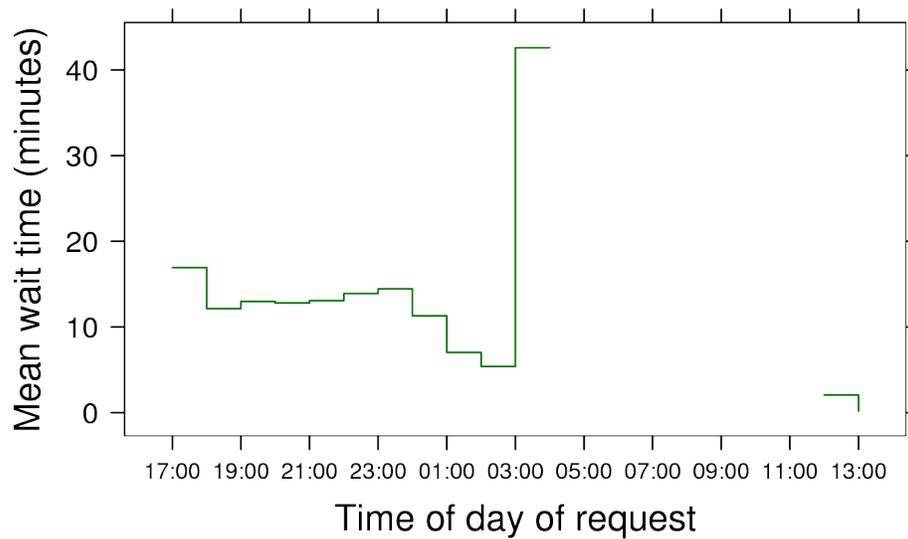
```
<R81> mp(
  xyplot( waitMedians ~ as.POSIXct(names(waitMedians)), type="s",
    main="Median wait time", xlab="Time of day of request",
    ylab="Median wait time (minutes)",
    scales=list(x=list(tick.number=10, cex=0.6))
  ),
  "medians.png", h=4, w=6)
#with graphics medians.png timeout 60
```

Median wait time



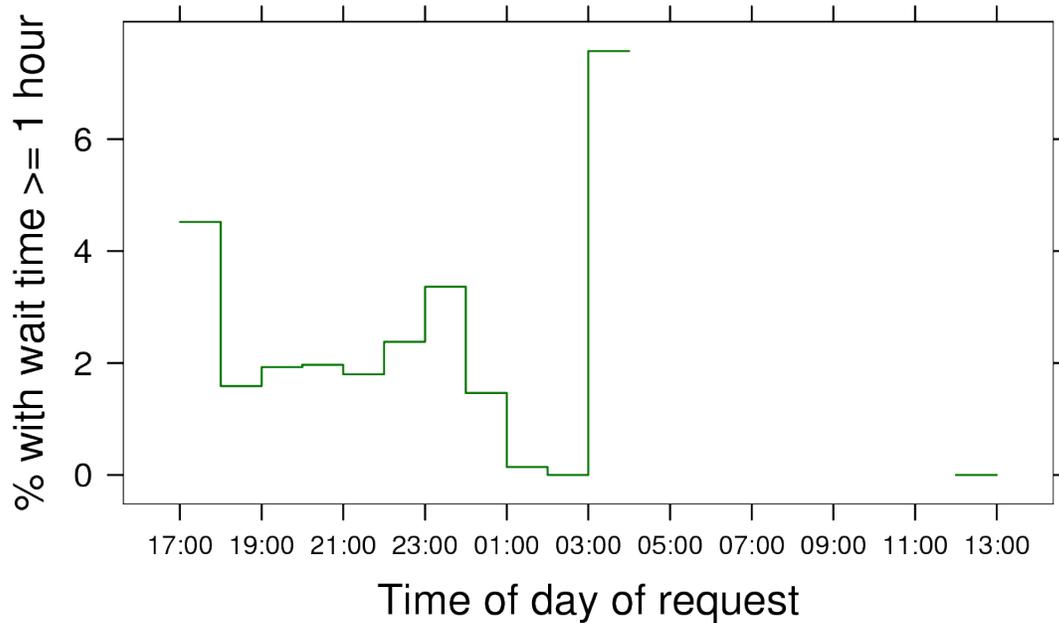
```
<R83> mp(  
  xyplot( waitMeans ~ as.POSIXct(names(waitMeans)), type="s",  
    main="Mean wait time", xlab="Time of day of request",  
    ylab="Mean wait time (minutes)",  
    scales=list(x=list(tick.number=10, cex=0.6))  
  ),  
  "means.png", h=4, w=6)  
#with graphics means.png timeout 60
```

Mean wait time



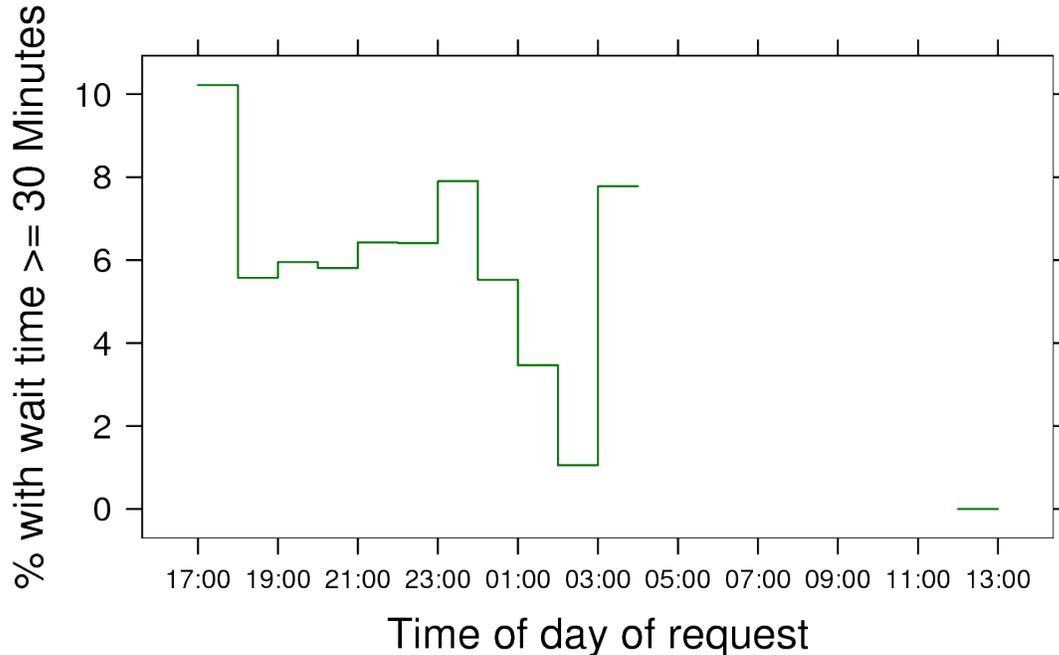
```
<R88> mp(  
  xyplot( percOverHour ~ as.POSIXct(names(percOverHour)), type="s",  
    main="Percent long wait times", xlab="Time of day of  
    request",  
    ylab="% with wait time >= 1 hour",  
    scales=list(x=list(tick.number=10, cex=0.6))  
  ),  
  "overHour.png", h=4, w=6)  
#with graphics overHour.png timeout 60
```

Percent long wait times



```
<R87> mp(
  xyplot( percOverHalfHour ~ as.POSIXct(names(percOverHalfHour)),
    type="s",
    main="Percent long wait times", xlab="Time of day of
    request",
    ylab="% with wait time >= 30 Minutes",
    scales=list(x=list(tick.number=10, cex=0.6))
  ),
  "overHalfHour.png", h=4, w=6)
#with graphics overHalfHour.png timeout 60
```

Percent long wait times



A.2.3 Look at number of open requests

Let's look at the number of instantaneous requests that are open at any given time.

Get next file requests...

```
<R89> gnf = data.frame( time=d$getPTime, adj=1 )
```

File deliveries (request fulfilled)

```
<R92> gnc = data.frame( time=d$delPTime, adj=-1 )
```

Join them,

```
<R93> gn = rbind(gnf, gnc)
```

Sort by time,

```
<R94> gn = gn[ order(gn$time), ]
```

Do a running count of # of unfulfilled get next file requests

```
<R95> gn$count = cumsum(gn$adj)
```

```
<R96> gn[1:10,]
```

```

                time adj count
1      2005-07-29 17:25:19   1    1
81     2005-07-29 17:25:42   1    2
201    2005-07-29 17:25:43   1    3
110000 2005-07-29 17:25:44  -1    2
81100  2005-07-29 17:25:46  -1    1
201100 2005-07-29 17:25:46  -1    0
121    2005-07-29 17:25:53   1    1
17448  2005-07-29 17:25:53   1    2
174481 2005-07-29 17:25:54  -1    1
121100 2005-07-29 17:25:55  -1    0

```

```
<R97> summary(gn$count)
```

```

  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0     330     876     696     891     925

```

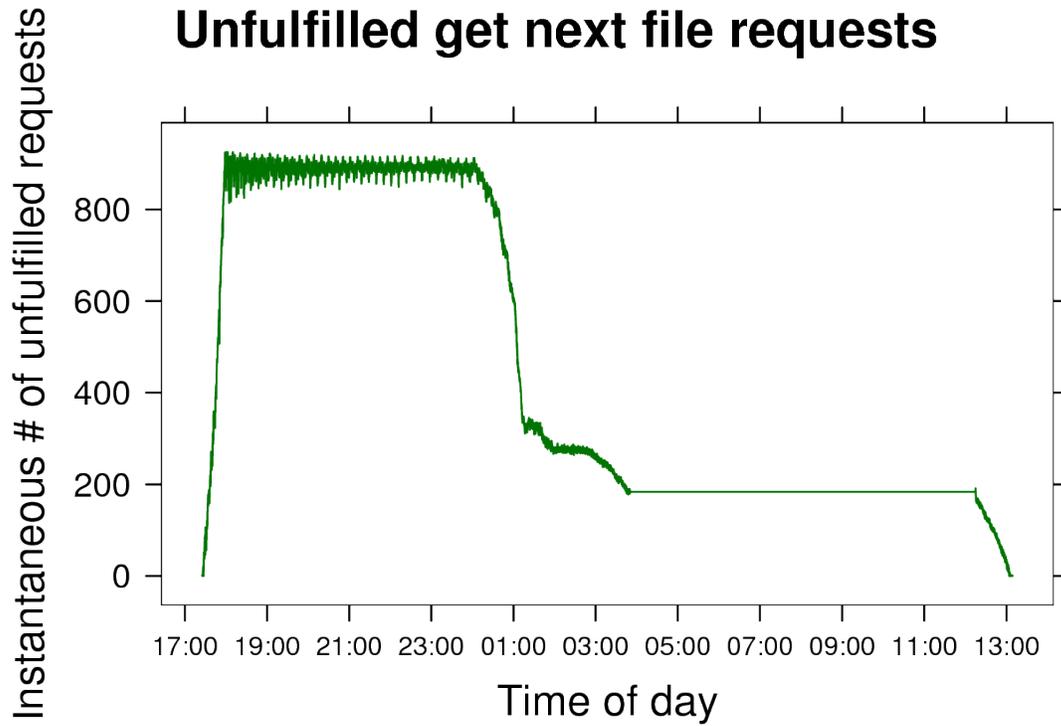
Plot it...

```

<R102> mp(
  xyplot( gn$count ~ gn$time, type="s",
    main="Unfulfilled get next file requests",
    xlab="Time of day",
    ylab="Instantaneous # of unfulfilled requests",
    scales=list(x=list(tick.number=10, cex=0.6)),
    xlim=prettyEdges ),
  "unfulfilled.png", h=4, w=6 )
#with graphics unfulfilled.png timeout 120

```

Unfulfilled get next file requests



I think this just shows the affect of the request wait time being longer than the request rate.