

Response to Draft CDF SAM Requirements !DRAFT!

Adam Lyon, July 20, 2005

1 Introduction

CDF has released draft expectations for SAMGrid in the form of a requirements document (sent via e-mail on June 30, 2005 and reproduced verbatim in Appendix A). The summary of DC rate expectations is shown in Table 1.

| | |
|--------------------------------------|--|
| Input file delivery | 8/s |
| File stores | 50/s (includes a declaration and setting location) |
| Meta-data queries | 25/s |
| Project starts & stops | 1/s |
| Dataset queries | 1/s |
| Project dumps | 2/s |
| Project recovery dataset definitions | 1/s |

Table 1. CDF SAM Rate Expectations (DC)

In order to evaluate these draft requirements, we compare the expectations to the current load of SAM generated by DØ. SAM has been running very successfully at DØ, and therefore provides a sense of the load that SAM can currently comfortably handle. Extensive testing will be required if SAM is asked to handle a significantly increased load.

2 Responses to CDF Expectations

We examine each CDF expectation, providing a comparison to DØ and a response from the SAMGrid project.

2.1 Project start/stops

CDF expected a project starting every second (they actually expect 7-8/s, but reduced the requirement to one). In DØ parlance, a project is a group of parallel jobs (may be just one) that all pull from the same pool of input files. In CDF parlance, a project is one job with one or more parallel segments. In SAM parlance, a project may consist of one or more parallel *consumers*. A consumer then consists of one or more parallel *processes* all pulling from the same pool of input files. So a process is equivalent to a DØ job and a CDF

segment. Note that multiple consumers are rare, but multiple processes under a single consumer are commonplace.

2.1.1 DØ experience

The project start rate has been measured for the time period July 1, 2004 – July 1, 2005 using the central SAM database. See Appendix B.3 for details of the measurements. The DØ project start rate for SAM world wide for the mentioned period is shown in Fig. 1.

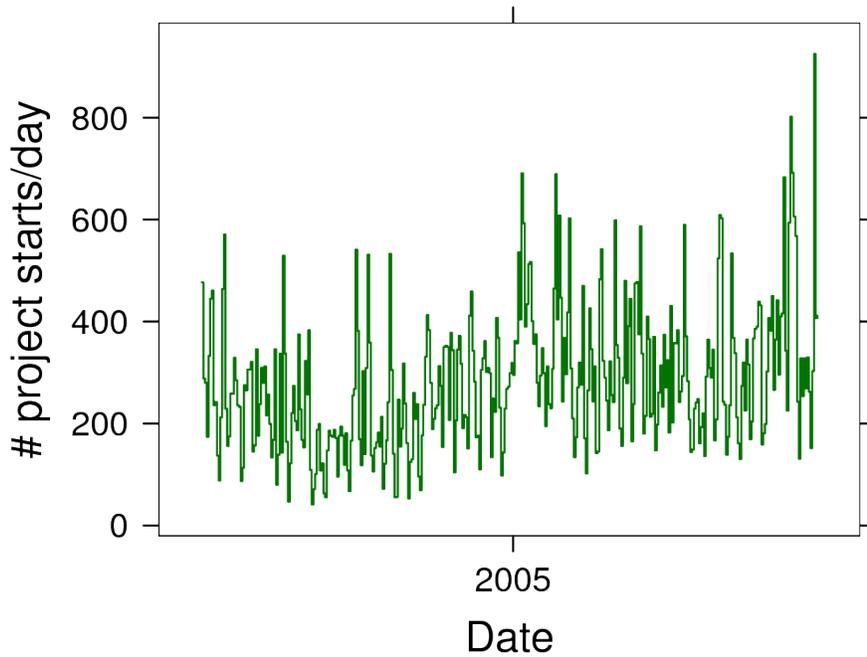


Fig. 1. DØ SAM world wide project start rate (starts per day) for 7/1/04-7/1/05.

The mean number of project starts per day is 282.8 per day or 0.0033 per second. The maximum number of starts per day is 925 per day or 0.011 per second. The maximum number of project starts is 93 times less than the claimed CDF DC expectation.

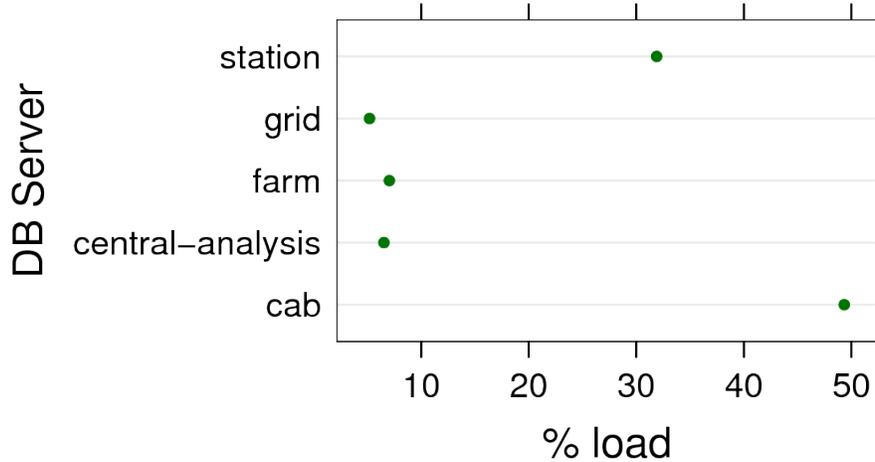


Fig. 2. Sharing the load among different DB servers for project starts.

DØ handles the load by distributing stations among different DB servers. In practice, the largest stations are on their own DB servers. For example, the two analysis stations share a DB server. The *station* DB server services the majority of stations.

2.1.2 Response

The expected number of DC project starts appears to be excessively high; 306 times the mean rate that DØ achieves world wide. Perhaps what was really meant was *process* starts - that is SAM servicing a job segment instead of a group of segments. Process starts will be examined next.

2.2 Process starts

A process starts for each parallel segment (or job in DØ parlance). The process corresponds to an instance of the executable that is expecting files.

2.2.1 DØ experience

The process start rate was measured from July 1, 2004 – July 1, 2005. See Appendix B.4 for details of the measurement.

The DØ process start rate for the period mentioned above is shown in Fig. 3.

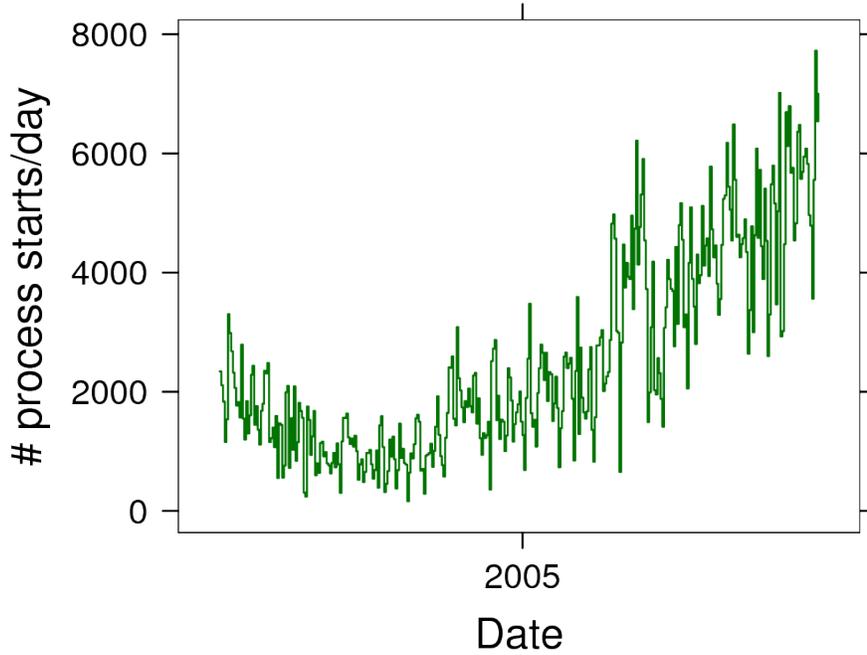


Fig. 3. The DØ process start rate for the period 7/1/04-7/1/05.

At around the start of 2005, the rate looks to be steadily increasing. To understand this effect the data are split by the servicing DB server. That plot is shown in Fig. 4. One clearly sees the start of the DØ reprocessing effort in late 2004 – early 2005. The Farm, Grid, and Station DB servers all service stations involved in the reprocessing effort. Note that the central-analysis station was discontinued at the start of 2005.

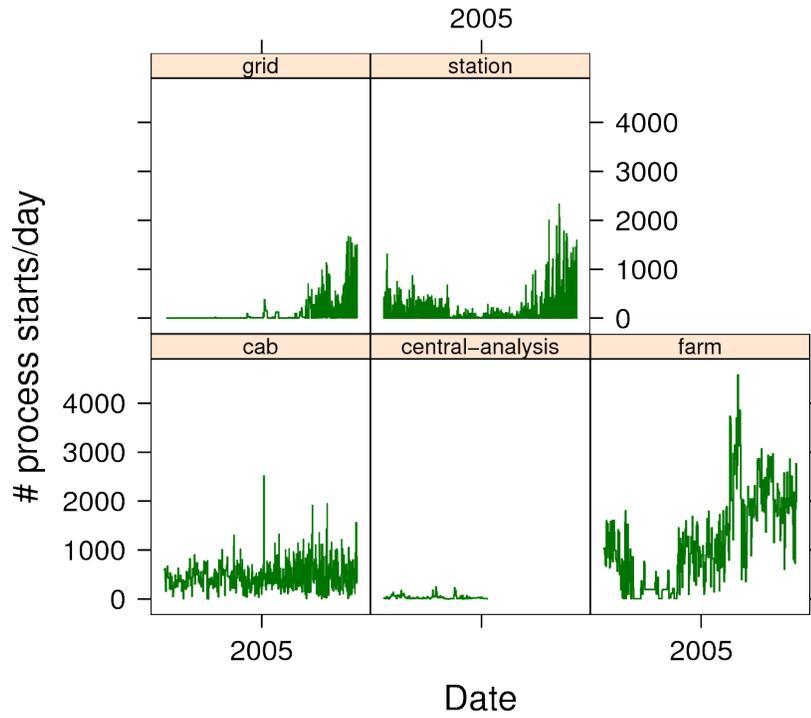


Fig. 4. Process start rates (#/day) shown for the different servicing DB servers. The name in the box indicates the DB server for the plot underneath.

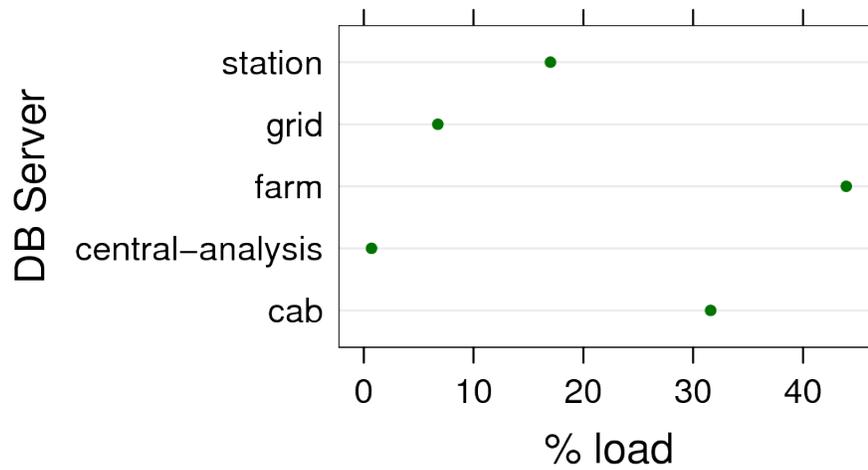


Fig. 5. Load sharing among DB servers for the total process start rate. To see how the DB servers share the total process start load, see Fig. 5.

2.2.2 Response

Assume that the CDF expected project start rate is really the *processs* start rate. The DØ world wide maximum process start rate is 7722 process starts per day, or 0.09 per second. The CDF expectation of one process start per second again seems excessively high (11 times higher than the DØ rate). We request that CDF produce a plot how many job segments actually start per second (not how many *can* start, but how many actually do, over the past year). We think that will give a better idea of the CDF load. Certainly the DØ load has increased significantly over the past year (about a factor of 4), and SAM has been able to keep up.

2.3 Input file delivery

SAM delivers a file to an awaiting executable. The executable then *consumes* the file, and is it marked as such in the SAM database. CDF expects to need a file consumption rate of 8 files per second.

2.3.1 DØ experience

The file consumption rate was measured for the period July 1, 2004 – July 1, 2005. See Appendix B.5 for the measurement details.

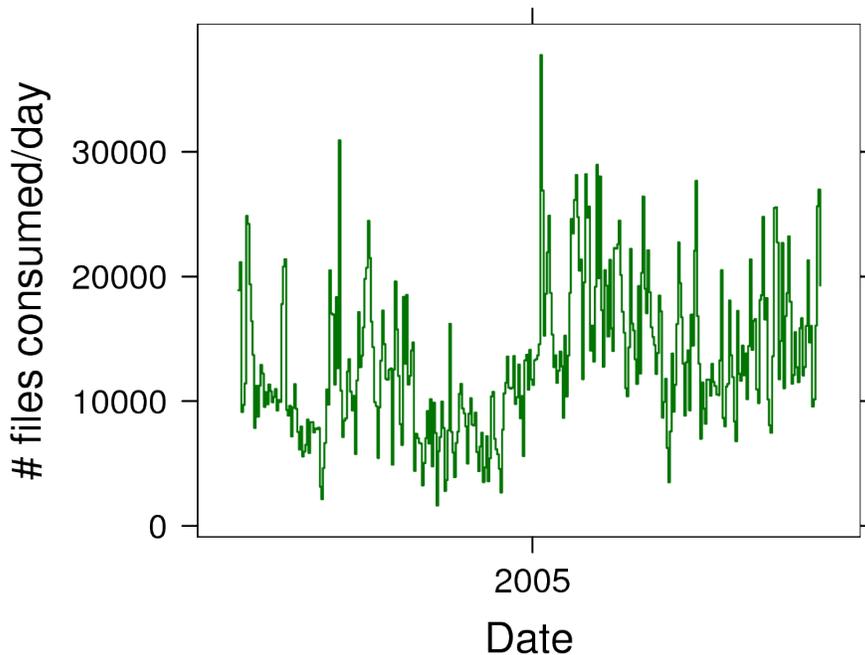


Fig. 6. Total number of files consumed per day for DØ for the period 7/1/04 – 7/1/05.

Fig. 6 shows the total world-wide file consumption rate (# files per day). The mean number of files consumed is 13192 per day (0.15/s) and the maximum is 37759 per day (0.44/s).

Fig. 7 indicates how the consumption rate is divided by the different servicing DB servers. The CAB analysis stations generate the most rate. Note the huge spike for the grid DB server during a reprocessing test.

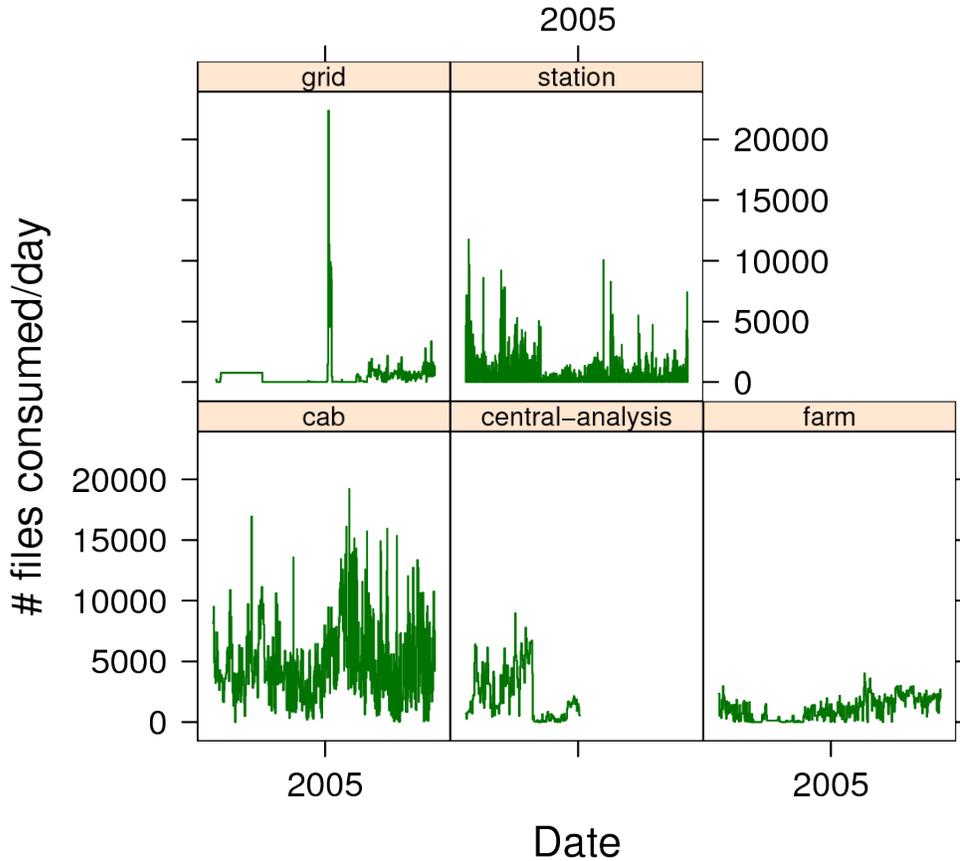


Fig. 7. File consumption shown for the different servicing DB servers.

The file consumption rate is shared by the DB servers. Fig. 8 shows how the load is divided. As suggested, CAB indeed creates the largest load and has its own DB server.

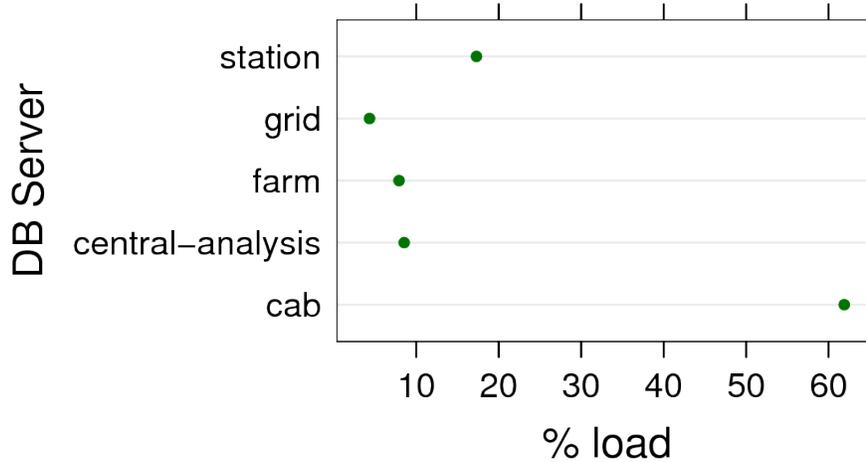


Fig. 8. File consumption load among the DB servers.

2.3.2 Response

The CDF expected file consumption rate is 52 times larger than the mean DØ rate and 18 times larger than the DØ maximum. Note that the DB server probably has very little to do with a limit on this rate -- rather it is the load on the station, disks, tapes, and network infrastructure. CDF obtained this expectation from the maximum dCache delivery rate. It would be more realistic to measure the CDF file input rate directly and compare.

2.4 File storage

File storage means declaring a file to SAM and then giving its location. CDF expects 50 file stores per second.

2.4.1 DØ experience

The DØ world wide file store rate was measured from July 1, 2004 – July 1, 2005. See Appendix B.6 for measurement details. This measurement includes files stored to durable locations in the process of merging.

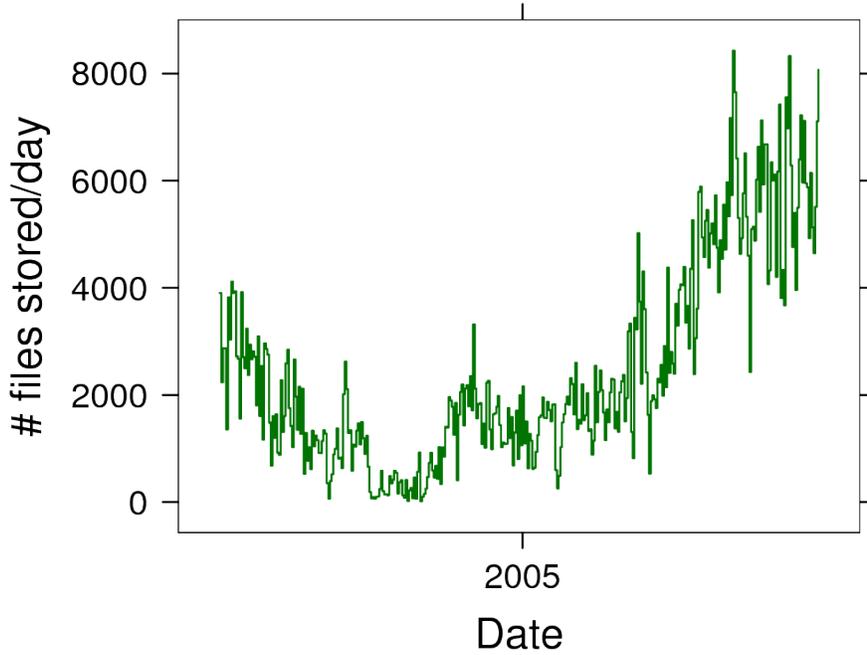


Fig. 9. DØ world wide file storage rate (including storing to durable locations). File stores/day.

Fig. 9 shows the world wide DØ file storage rate. As in the process rate, the plot is increasing steadily after the start of 2005. Looking at how the file stores are broken up among DB servers, it is clear that the DØ reprocessing is causing the rise. See Fig. 10. Note that dlsam is the online raw file storage.

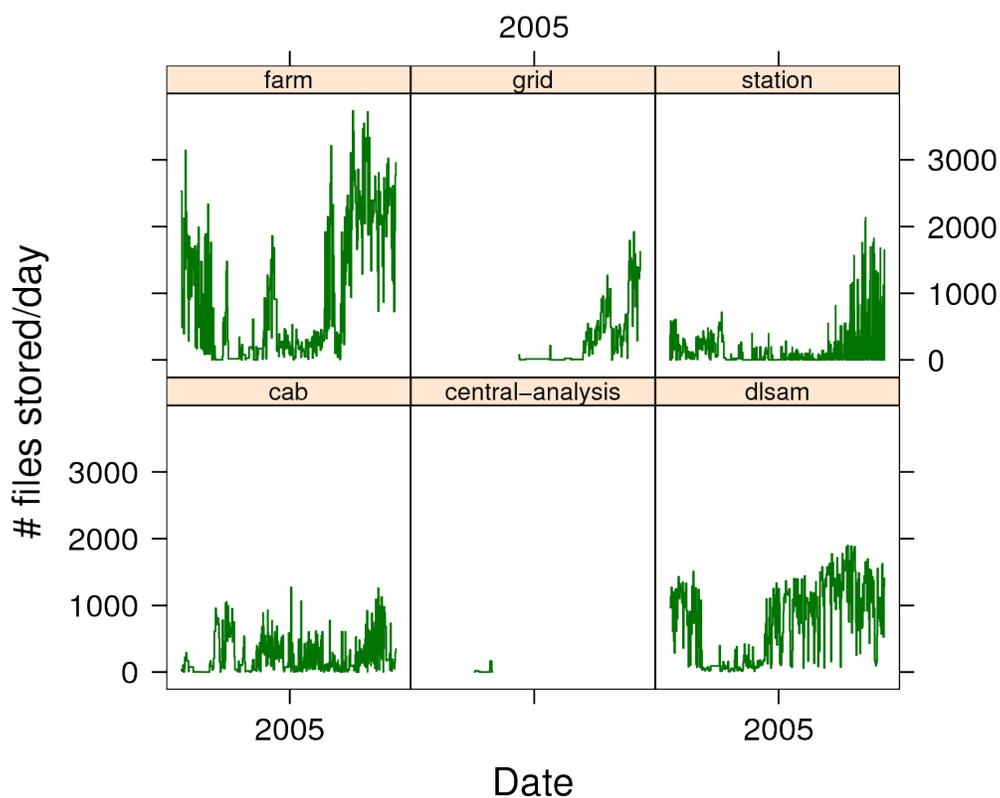


Fig. 10. DØ file store rate for the different DB servers. One sees the large increase from the farm, grid, and station DB servers as a result of the DØ reprocessing effort.

As in all cases before, having multiple DB servers helps to spread the load, as shown in Fig. 11.

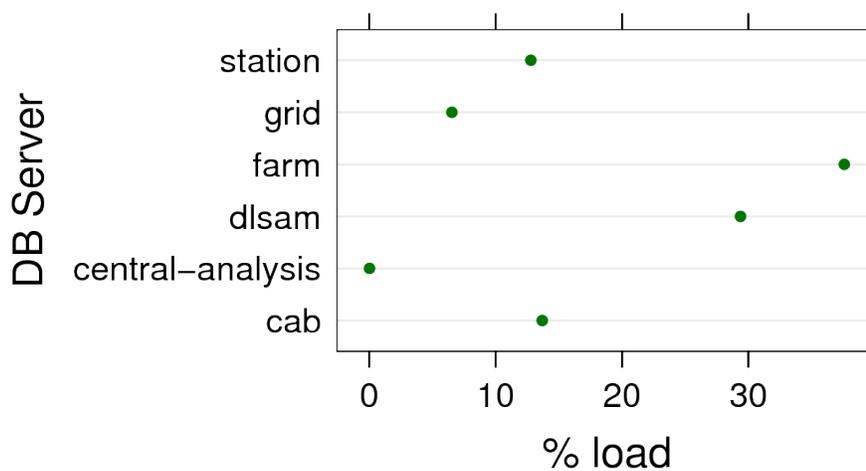


Fig. 11. Load sharing for total file store rate among the DB servers.

It is also important to look at file storage rates by bytes instead of numbers of files. See Fig. 12.

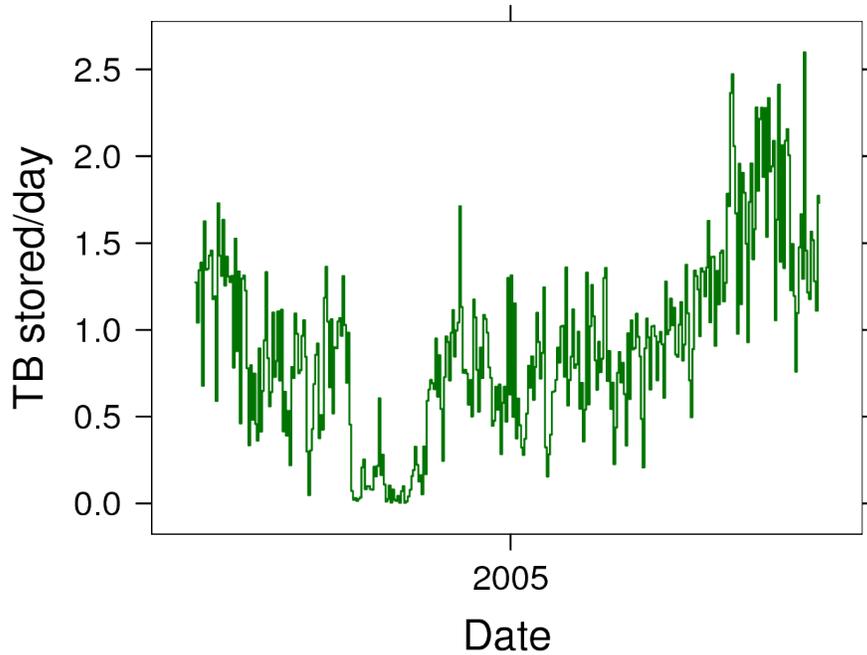


Fig. 12. DØ world wide file storage rates by Terabyts/day.

To understand how this load is generated, the same information is shown divided up by servicing DB server in Fig. 13. Note that the drop in Farm storage is due to the cessation of storing the large size DST files. The file stores from CAB are skims and analysis fixed thumbnails. DLSAM is the online system. The rise in the Farm, Grid, and Station DB servers represent the DØ reprocessing effort.

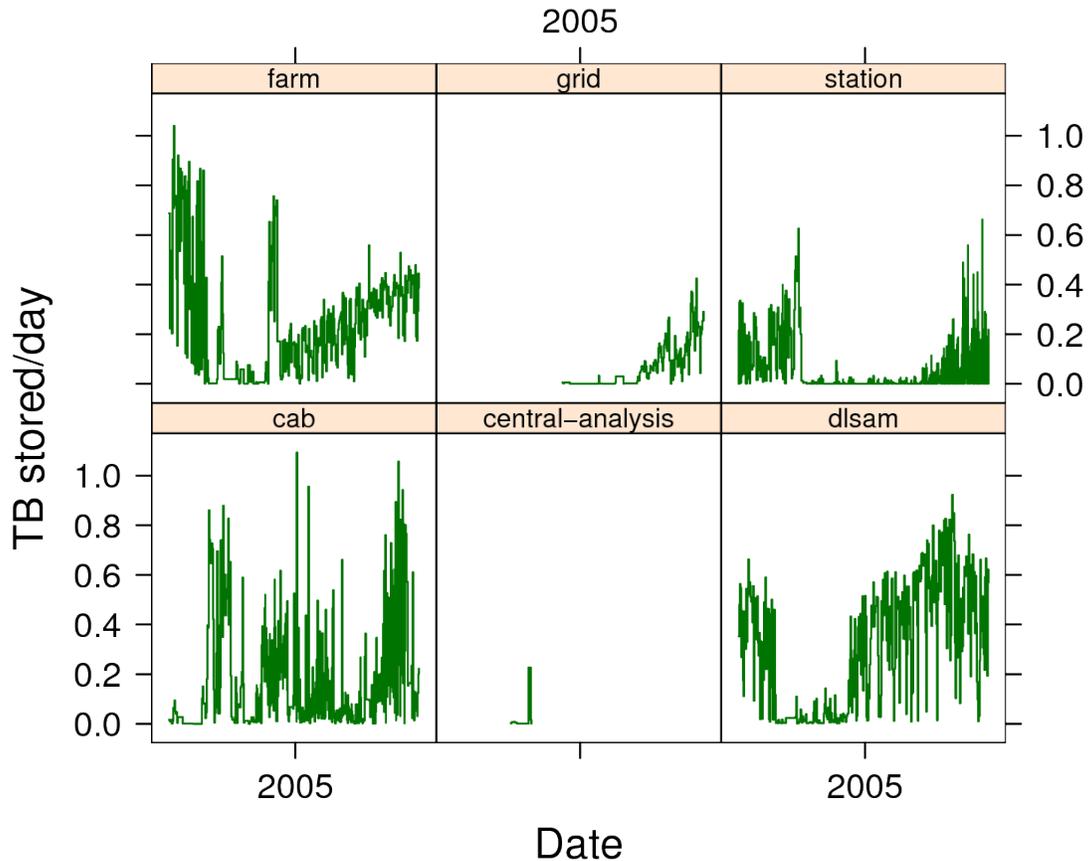


Fig. 13. File storage by DBServer. See notes in the text.

2.4.2 Response

The mean DØ file store rate is 2471 files per day (0.029/s) or 0.9 TB/day. The maximum DØ file store rate for the period is 8425 files per day (0.098/s) or 2.6 TB/day. The values include all MC, raw data, processing, reprocessing and user file stores, including intermediate MC and merging files. The CDF expectation of 50 file stores per second is 1748 times the DØ mean rate and is 513 times the DØ maximum rate. only caveat is that DØ tends to store files as the jobs go along, instead of waiting until the very end. If CDF plans to store files in bursts, they would be well advised to consider the DØ method.

3 Summary

| Quantity | CDF DC Expectation/s | DØ Mean/s | CDF is × times mean | DØ max/s | CDF is × times max |
|------------------|----------------------|-----------|---------------------|----------|--------------------|
| Project starts | 1 | 0.0033 | 306 | 0.011 | 93 |
| Process starts | 1 | 0.029 | 34 | 0.09 | 11 |
| File consumption | 8 | 0.15 | 52 | 0.44 | 18 |
| File storage | 50 | 0.029 | 1748 | 0.098 | 513 |

Appendix A CDF's Draft Requirements Document

Sent via e-mail on June 30, 2005, reproduced here verbatim...

.) From the dCache agreement:

up to 80 TB/day, 4 TB/hour.
assuming 1GB files

80000/day = 0.92 files/sec (if 100k files) 1.2 files/sec
4000/3600 = 1.11 files/sec 1.3 files/sec

dCache-PNFS transaction "bandwidth" of about 25-30k per hour

30k (open/close)/hour = 15000files = 4 input files/sec

.) Reconstruction farm:

Assuming Reconstruction farm is producing 5 TBytes of data per day
for the typical file size of 1 GByte this translates into ~5000 files

5000 input files/day = 0.0578 files/sec x2 safety = 0.116 files/sec

5000 input files x~10 for the intermediate files x2 for their
declaration and deletion of the tmp location plus the final files ~
100000 file declaration/day

1.2/sec x2 as a safety margin = 2.4 output/intermediate file declaration/sec

get metadata for each of the intermediate file 50000/day x2 = 1.2 files/sec

.) Donatella's example:

1GB file/10min x100 segments = 0.167 input files/sec; 0.6TB/h
(if x4000 segments = 6.7 input files/sec, or 24TB/h)
(if everyone did it it would exceed the dCache agreement,
so we would need to divide it by 6).

her group is producing 12 intermediate files (->24 declarations) per
file (I neglect here the merged output files as this is a small fraction
of the total)

0.042 file declarations/sec x100 segments 4.2 file declarations/sec
x4000 167 file declarations/sec

to stay within the dCache limits (assuming 4 input files/sec)
the maximum would be 100 file declarations per sec

trying to be realistic let us say that only 5 groups will do a split, so
I would say it can be reduced to 4.2x5x2~ 50

.) caf submission/number of projects estimate
jobs can be submitted every 0.1 sec shortest segment can run for 10min
(it is set to 40min now) so this means about 7-10 project starts/ends
per second at peak, but lets assume it is only one per second for now
given that there are very few one file projects; but each submission
does a dataset verification, so this is another query per second

.) MC production of 600 files/day (gives a negligible number of declarations)

In summary; current requirements for sam should be (assuming x2 safety

margin)

- 8 input files per second
- 50 output file declarations per second (this include an initial declaration plus a location modification);
- 25 one file metadata queries per second
 - 1 project start/stop per second
 - 1 dataset query per second
 - 2 project summaries per second
 - 1 project recoveries per second

the numbers are dc rate, we are not sure what the peak is, but the fact that there are ~4000 caf nodes should be taken into account.

this does not include remote dcafs; so, may be another factor of two is in order.

We also think we should have an expected command response time which should be under 5 seconds for most commands and 10 sec for the longer ones. Reliability is more important that the response time though.

Krzysztof & Doug for CDF DH

Appendix B 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.

B.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/samData/d0Usage
```

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:RemoterRSOAP>
```

Initialize the database connection using the public read account

```
<R5> library(ROracle)  
Loading required package: DBI  
<R6> db = dbConnect(Oracle(), db="d0ofprd1", user="d0read",  
                  pass="reader")
```

Make a little function to help in creation of sql strings

```
<R7> p = function(...) paste(..., sep="\n")
```

B.2 Get station information

We need information about the SAM stations

```
<R8> sql = p("select station_id, station_name from stations")
```

```
<R9> stations = dbGetQuery(db, sql)
```

Make the column names easier.

```
<R10> names(stations) = c("id", "name")
```

What did we get? Print the first 10.

```
<R11> stations[1:10,]
  id      name
0  1  protofarm
1  6 d0_main_analysis
2 11      d02ka
3 21 central-analysis
4 29  datalogger
5 31 d0-test-station
6 43  d0small-01
7 46 ccin2p3-analysis
8 51  pctestfarm
9 56  clued0-v5
```

Add a new column for the DBserver that handles the station (default is the station DBServer)

```
<R12> stations$dbServer = "station"
```

We know that the CAB stations belong to the CAB DBServer.

```
<R13> stations$dbServer[ grep("fnal-cabsrv", stations$name) ] = "cab"
```

The farm uses the farm DBServer

```
<R14> stations$dbServer[ stations$name == 'fnal-farm' ] = "farm"
```

The ccin2p3-analysis and ccin2p3-grid0 (but not grid1) stations use the Grid DBServer

```
<R15> stations$dbServer[stations$name == 'ccin2p3-analysis' |
                        stations$name == 'ccin2p3-grid0' ] = "grid"
```

When Central-analysis was around, it used its own DBServer

```
<R16> stations$dbServer[stations$name == 'central-analysis' ] =
      'central-analysis'
```

Online uses the dlsam DB server.

```
<R17> stations$dbServer[grep("d001", stations$name) ] = "dlsam"
```

One would think that the clued0 station would use the clued0 DBServer, but in fact it uses the station DBServer. The clued0 DBServer is for interactive stuff.

How many stations are on each DBServer?

```
<R18> table(stations$dbServer)
```

| | | | |
|------|------------------|-------|------|
| cab | central-analysis | dlsam | farm |
| 3 | 1 | 5 | 1 |
| grid | station | | |
| 2 | 93 | | |

B.3 Project starts

Do the query for the year July 1, 2004 - July 1, 2005

```
<R65> sql = p("select to_char(start_time, 'MM/DD/YYYY'), station_id, ",  
"      count(project_id) ",  
"      from analysis_projects ",  
"      where start_time > '1-jul-2004' and ",  
"      start_time < '1-jul-2005' ",  
"      group by to_char(start_time, 'MM/DD/YYYY'), station_id" )
```

```
<R66> ps = dbGetQuery(db, sql)
```

```
<R67> names(ps) = c("date", "stationId", "count")
```

```
<R68> ps[1:10,]  
      date stationId count  
0 01/01/2005      21    39  
1 01/01/2005      56     8  
2 01/01/2005     131    11  
3 01/01/2005     372    13  
4 01/01/2005     406    25  
5 01/01/2005     411   199  
6 01/02/2005      21    19  
7 01/02/2005      56    11  
8 01/02/2005     121     1  
9 01/02/2005     136     1
```

Fill in the db server and station name given the id

```
<R75> pss = merge(ps, stations, by.x="stationId", by.y="id", sort=F)
```

Let's sum up the counts by date for a total

```
<R77> pss.total = aggregate(pss$count, by=list(date=pss$date), sum )
```

```
<R82> names(pss.total) = c("date", "counts")
```

```
<R83> pss.total[1:10,]
```

```
      date counts
1 01/01/2005   295
2 01/02/2005   362
3 01/03/2005   357
4 01/04/2005   536
5 01/05/2005   404
6 01/06/2005   691
7 01/07/2005   593
8 01/08/2005   390
9 01/09/2005   434
10 01/10/2005   513
```

What are the statistics?

```
<R86> summary(pss.total$counts)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  41.0   177.0   265.0   282.8   357.0   925.0
```

Set some MSWord variables that we can refer to later

```
<R87> #var pss.max = max(pss.total$counts)
```

```
925
```

```
<R90> #var pss.mean = format(mean(pss.total$counts), digits=4)
```

```
282.8
```

```
<R29> #var pss.max.s = format( max(pss.total$counts)/24/60/60,
  digits=2)
```

```
0.011
```

```
<R30> #var pss.mean.s = format( mean(pss.total$counts)/24/60/60,
  digits=2)
```

```
0.0033
```

```
<R258> #var pss.mean.cdf = format(
  1.0/(mean(pss.total$counts)/24/60/60), digits=2)
```

```
306
```

```
<R259> #var pss.max.cdf = format( 1.0/(max(pss.total$counts)/24/60/60),
  digits=2)
```

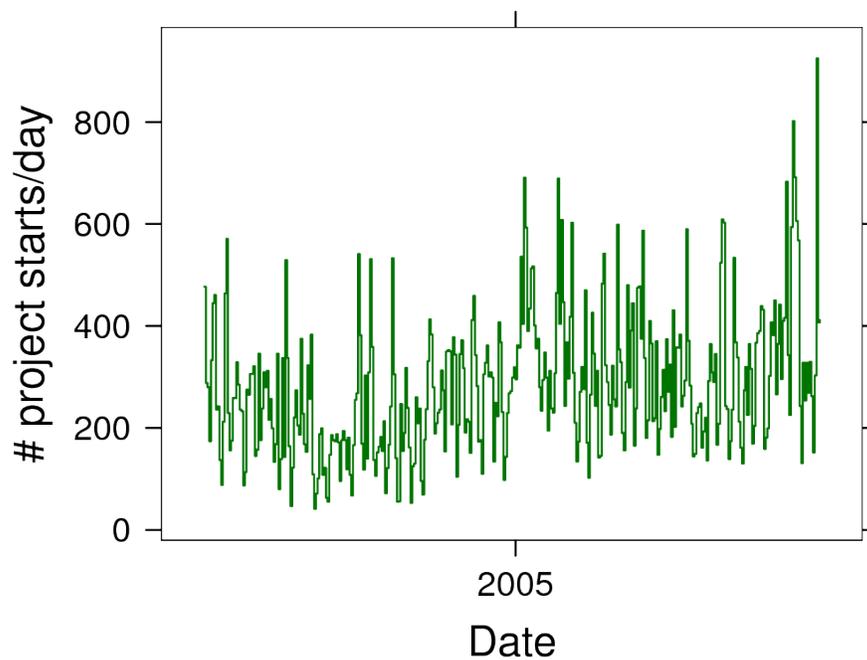
```
93
```

Convert the dates to POSIX dates

```
<R99> pss.total$pDate = as.POSIXct( strptime(pss.total$date,
"%m/%d/%Y") )
```

Plot the number of project starts per day

```
<R19> mp(
  xyplot( counts ~ pDate, data=pss.total, type="s",
    xlab="Date", ylab="# project starts/day"
  ),
  "pss.total.png", h=4, w=5
)
#with graphics pss.total.png
```



Split up by DBServers,

```
<R167> pss.dbServer = aggregate(pss$count,
  by=list(dbServer=pss$dbServer), sum )
```

```
<R168> names(pss.dbServer) = c("dbServer", "count")
```

```
<R169> pss.dbServer
```

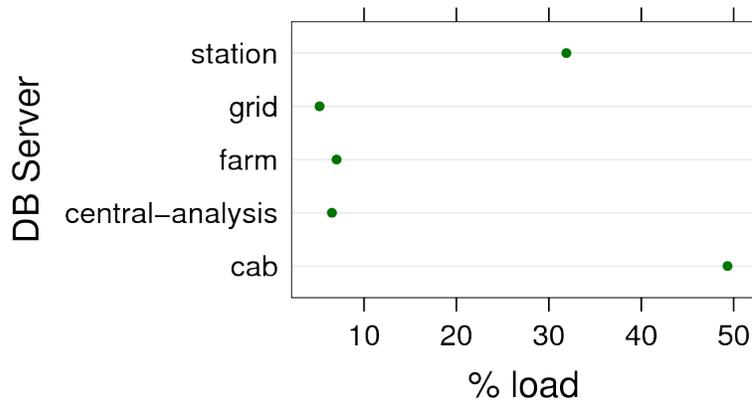
```

      dbServer count
1          cab 50924
2 central-analysis 6746
3          farm 7264
4          grid 5366
5          station 32923

<R170> pss.dbServer$perc = pss.dbServer$count / sum(pss.dbServer$count)
      * 100

<R171> mp(
      dotplot( dbServer ~ perc, data=pss.dbServer, xlab="% load",
              ylab="DB Server"),
      "pss.dbServer.png", h=3, w=5)
      #with graphics pss.dbServer.png

```



B.4 Process starts

Do the query for the year July 1, 2004 – July 1, 2005

```

<R57> sql = p("select to_char(pr.create_date, 'MM/DD/YYYY'),",
      "          p.station_id, count(pr.process_id)",
      "          from analysis_projects p, consumers c,",
      "          analysis_processes pr",
      "          where pr.create_date > '1-jul-2004' and ",
      "                 pr.create_date < '1-jul-2005' and",
      "                 c.consumer_id = pr.consumer_id and",
      "                 p.project_id = c.project_id",
      "          group by to_char(pr.create_date, 'MM/DD/YYYY'),",
      "                 p.station_id"
      )

```

```
<R59> prs = dbGetQuery(db, sql)
      #with timeout 120
```

```
<R61> names(prs) = c("date", "stationId", "count")
```

```
<R62> prs[1:10,]
      date stationId count
0 01/01/2005      21    39
1 01/01/2005      56     8
2 01/01/2005     131   885
3 01/01/2005     372    13
4 01/01/2005     406   100
5 01/01/2005     411   227
6 01/02/2005      21    19
7 01/02/2005      56     4
8 01/02/2005     121     1
9 01/02/2005     136     1
```

Fill in the db server and station name given the id

```
<R63> prss = merge(prs, stations, by.x="stationId", by.y="id", sort=F)
```

Let's sum up the counts by date for a total

```
<R64> prss.total = aggregate(prss$count, by=list(date=prss$date), sum )
```

```
<R65> names(prss.total) = c("date", "counts")
```

```
<R66> prss.total[1:10,]
      date counts
1 01/01/2005  1272
2 01/02/2005   687
3 01/03/2005  1898
4 01/04/2005  2555
5 01/05/2005  3477
6 01/06/2005  1641
7 01/07/2005  1414
8 01/08/2005  1529
9 01/09/2005  1079
10 01/10/2005  1982
```

What are the statistics?

```
<R67> summary(prss.total$counts)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   162   1222   1982   2522   3725   7722
```

Set some MSWord variables that we can refer to later

```
<R68> #var prss.max = max(prss.total$counts)
```

```
7722
```

```
<R70> #var prss.mean = format(mean(prss.total$counts), digits=4)
```

```
2522
```

```
<R71> #var prss.max.s = format( max(prss.total$counts)/24/60/60,  
  digits=2)
```

```
0.09
```

```
<R72> #var prss.mean.s = format( mean(prss.total$counts)/24/60/60,  
  digits=2)
```

```
0.029
```

```
<R256> #var prss.mean.cdf = format(  
  1.0/(mean(prss.total$counts)/24/60/60), digits=2)
```

```
34
```

```
<R257> #var prss.max.cdf = format(  
  1.0/(max(prss.total$counts)/24/60/60), digits=2)
```

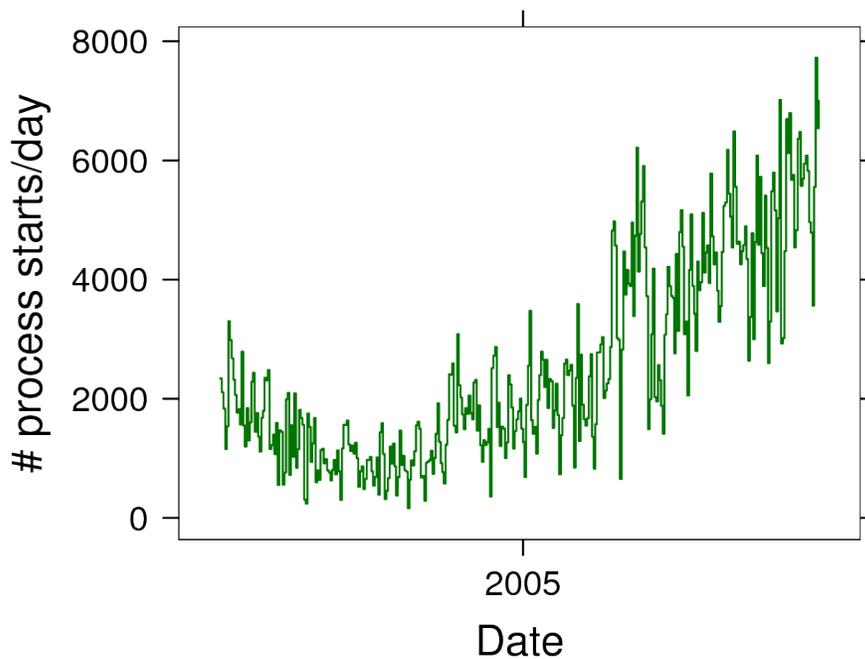
```
11
```

Convert the dates to POSIX dates

```
<R73> prss.total$pDate = as.POSIXct( strptime(prss.total$date,  
  "%m/%d/%Y") )
```

Plot the number of project starts per day

```
<R87> mp(  
  xyplot( counts ~ pDate, data=prss.total, type="s",  
    xlab="Date", ylab="# process starts/day"  
  ),  
  "prss.total.png", h=4, w=5  
  )  
  #with graphics prss.total.png
```



Split up by DBServers,

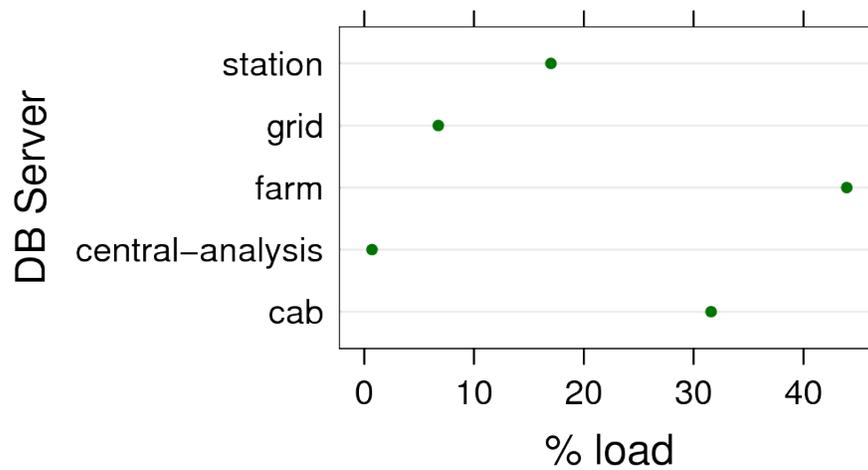
```
<R75> prss.dbServer = aggregate(prss$count,
  by=list(dbServer=prss$dbServer), sum )
```

```
<R76> names(prss.dbServer) = c("dbServer", "count")
```

```
<R77> prss.dbServer
      dbServer count
1          cab 290835
2 central-analysis  6587
3          farm 404541
4          grid  62171
5          station 156564
```

```
<R78> prss.dbServer$perc = prss.dbServer$count /
  sum(prss.dbServer$count) * 100
```

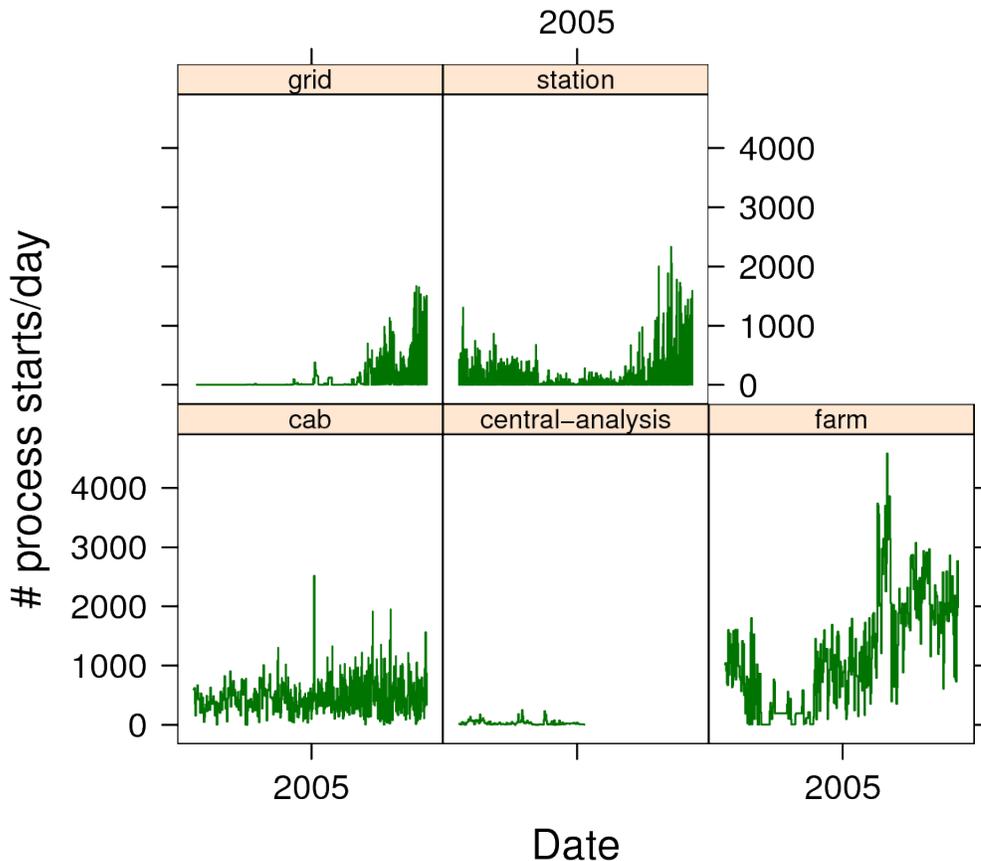
```
<R90> mp(
  dotplot( dbServer ~ perc, data=prss.dbServer, xlab="% load",
    ylab="DB Server"),
  "prss.dbServer.png", h=3, w=5)
#with graphics prss.dbServer.png
```



I wonder what the rise is in the total plot?

```
<R81> prss$pDate = as.POSIXct( strptime(prss$date, "%m/%d/%Y") )
```

```
<R89> mp(
  xyplot( count ~ pDate | dbServer, data=prss, type="s",
    xlab="Date", ylab="# process starts/day",
    par.strip.text=list(cex=0.6)
  ),
  "prss.splitByDbServer.png", h=5, w=6
)
#with graphics prss.splitByDbServer.png
```



So clearly it's due to the reprocessing!

B.5 File consumption

Do the query for the year July 1, 2004 – July 1, 2005

```
<R91> sql = p("select to_char(cf.create_date, 'MM/DD/YYYY'),",
"          p.station_id, ",
"          count(cf.proj_snap_id || cf.file_number ||",
"          cf.process_id) ",
" from analysis_projects p, consumers c, consumed_files cf,",
" where cf.create_date > '1-jul-2004' and ",
"       cf.create_date < '1-jul-2005' and ",
"       c.consumer_id = cf.consumer_id and ",
"       p.project_id = c.project_id ",
" group by to_char(cf.create_date, 'MM/DD/YYYY'), ",
"          p.station_id"
")
```

```
<R92> fc = dbGetQuery(db, sql)
      #with timeout 120
```

```
<R260> names(fc) = c("date", "stationId", "count")
```

```
<R94> fc[1:10,]
      date stationId count
0 01/01/2005      21 1499
1 01/01/2005      56   11
2 01/01/2005     131 1472
3 01/01/2005     201   10
4 01/01/2005     372  361
5 01/01/2005     406 2474
6 01/01/2005     411 5473
7 01/02/2005      21 1215
8 01/02/2005      56  111
9 01/02/2005     131   62
```

Fill in the db server and station name given the id

```
<R95> fcs = merge(fc, stations, by.x="stationId", by.y="id", sort=F)
```

Let's sum up the counts by date for a total

```
<R96> fcs.total = aggregate(fcs$count, by=list(date=fcs$date), sum )
```

```
<R97> names(fcs.total) = c("date", "counts")
```

```
<R98> fcs.total[1:10,]
      date counts
1 01/01/2005 11300
2 01/02/2005 13278
3 01/03/2005 13376
4 01/04/2005 13670
5 01/05/2005 14563
6 01/06/2005 37759
7 01/07/2005 26887
8 01/08/2005 15260
9 01/09/2005 18609
10 01/10/2005 21938
```

What are the statistics?

```
<R99> summary(fcs.total$counts)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 1620   9136   12020   13190   16650   37760
```

Set some MSWord variables that we can refer to later

```
<R100> #var fcs.max = max(fcs.total$counts)
37759
```

```
<R101> #var fcs.mean = format(mean(fcs.total$counts), digits=4)
13192
```

```
<R102> #var fcs.max.s = format( max(fcs.total$counts)/24/60/60,
  digits=2)
0.44
```

```
<R103> #var fcs.mean.s = format( mean(fcs.total$counts)/24/60/60,
  digits=2)
0.15
```

```
<R122> #var fcs.mean.cdf = format( 8.0 /
  (mean(fcs.total$counts)/24/60/60), digits=1)
52
```

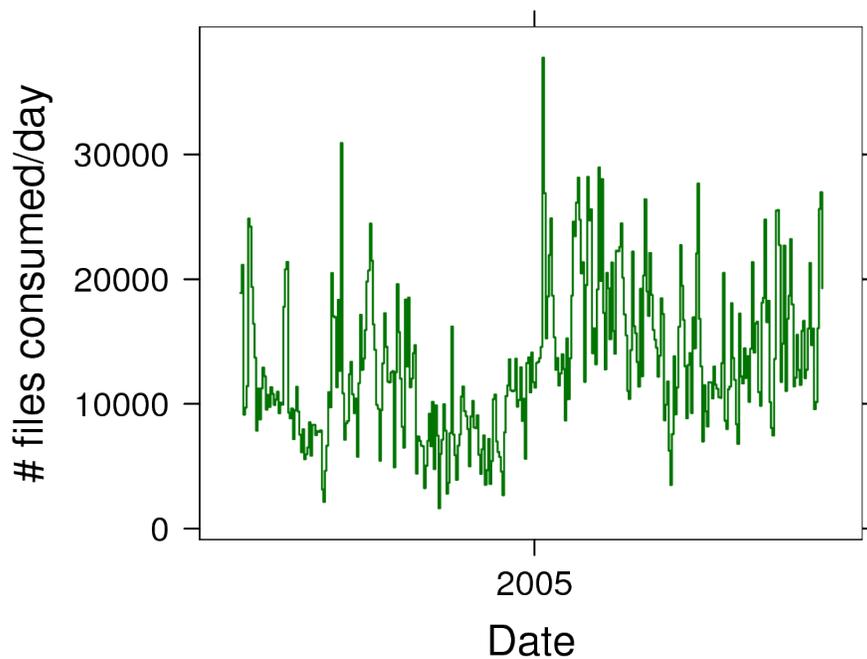
```
<R123> #var fcs.max.cdf = format( 8.0 /
  (max(fcs.total$counts)/24/60/60), digits=1)
18
```

Convert the dates to POSIX dates

```
<R105> fcs.total$pDate = as.POSIXct( strptime(fcs.total$date,
  "%m/%d/%Y") )
```

Plot the number of files consumed per day

```
<R106> mp(
  xyplot( counts ~ pDate, data=fcs.total, type="s",
    xlab="Date", ylab="# files consumed/day"
  ),
  "fcs.total.png", h=4, w=5
)
#with graphics fcs.total.png
```



Split up by DBServers,

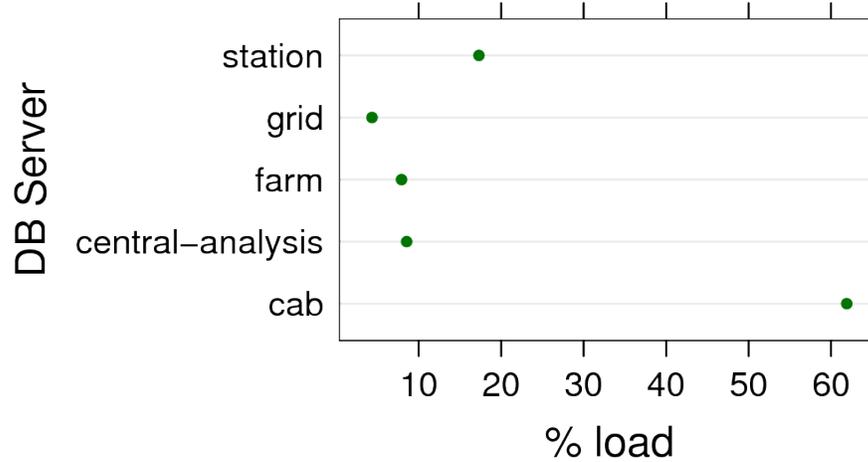
```
<R107> fcs.dbServer = aggregate(fcs$count,
  by=list(dbServer=fcs$dbServer), sum )
```

```
<R108> names(fcs.dbServer) = c("dbServer", "count")
```

```
<R109> fcs.dbServer
      dbServer  count
1          cab 2979874
2 central-analysis 411431
3          farm 381247
4          grid 209250
5        station 833329
```

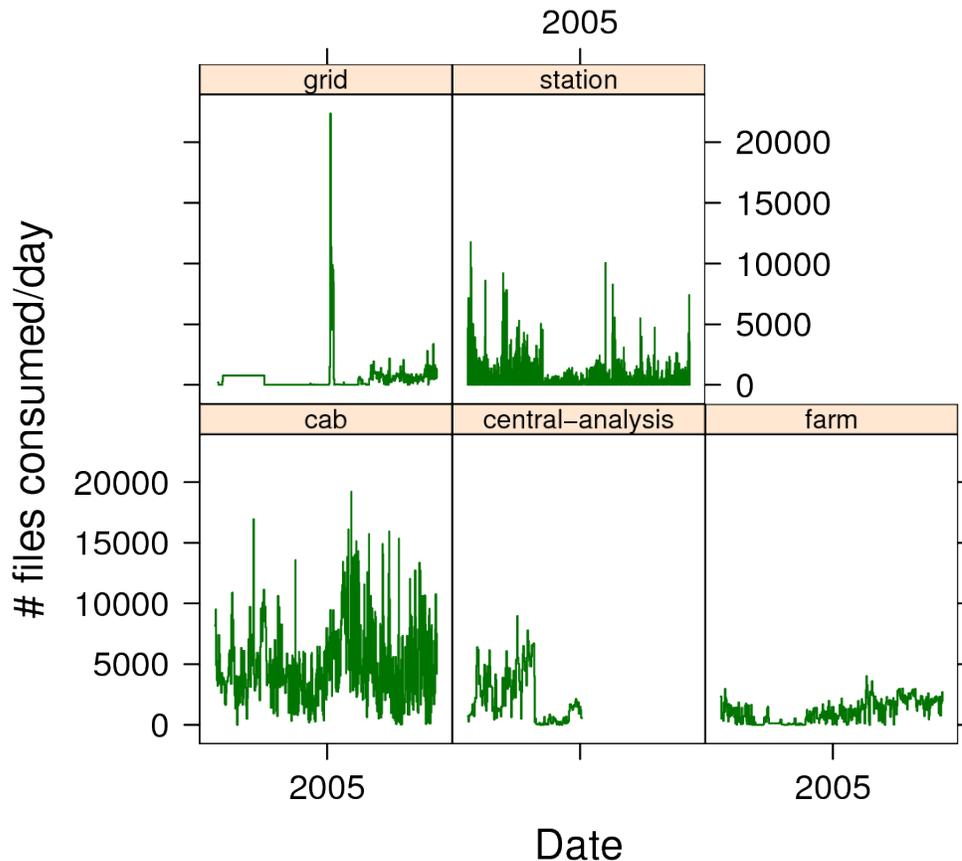
```
<R110> fcs.dbServer$perc = fcs.dbServer$count / sum(fcs.dbServer$count)
  * 100
```

```
<R111> mp(
  dotplot( dbServer ~ perc, data=fcs.dbServer, xlab="% load",
    ylab="DB Server"),
  "fcs.dbServer.png", h=3, w=5)
#with graphics fcs.dbServer.png
```



```
<R112> fcs$pDate = as.POSIXct( strptime(fcs$date, "%m/%d/%Y") )
```

```
<R113> mp(
  xyplot( count ~ pDate | dbServer, data=fcs, type="s",
    xlab="Date", ylab="# files consumed/day",
    par.strip.text=list(cex=0.6)
  ),
  "fcs.splitByDbServer.png", h=5, w=6
)
#with graphics fcs.splitByDbServer.png
```



B.6 File storage

Do the query for the year July 1, 2004 – July 1, 2005

```
<R20> sql = p("select to_char(df.create_date, 'MM/DD/YYYY'),",
"      p.station_id, count(df.file_id)",
"      sum(df.file_size_in_bytes)",
" from analysis_projects p, consumers c,",
"      analysis_processes pr, data_files df",
" where df.create_date > '1-jul-2004' and ",
"       df.create_date < '1-jul-2005' and",
"       pr.process_id = df.process_id and",
"       c.consumer_id = pr.consumer_id and",
"       p.project_id = c.project_id",
" group by to_char(df.create_date, 'MM/DD/YYYY'),",
"          p.station_id")
```

```
<R21> fs = dbGetQuery(db, sql)
#with timeout 120
```

```

<R22> fs[1:10,]
  TO_CHAR(DF.CREATE_DATE, 'MM/DD/ STATION_ID COUNT(DF.FILE_ID)
0          01/01/2005          131          194
1          01/01/2005          406          291
2          01/01/2005          411          428
3          01/02/2005          131           70
4          01/03/2005          131           83
5          01/03/2005          406           9
6          01/03/2005          411         1274
7          01/04/2005          131          223
8          01/04/2005          411          308
9          01/05/2005          131          530
SUM(DF.FILE_SIZE_IN_BYTES)
0          1.762065e+11
1          2.791566e+11
2          4.219895e+11
3          7.045787e+10
4          2.577005e+10
5          9.086288e+09
6          1.201856e+12
7          9.328890e+10
8          3.048394e+11
9          2.177248e+11

```

```

<R23> names(fs) = c("date", "stationId", "count", "bytes")

```

File stores from online don't show up here because they have a fake process ID. Just look for raw files.

```

<R24> sql = p("select to_char(df.create_date, 'MM/DD/YYYY'),",
"          count(df.file_id), sum(df.file_size_in_bytes)",
" from data_files df",
" where df.create_date > '1-jul-2004' and ",
"          df.create_date < '1-jul-2005' and",
"          df.data_tier_id = 9",
" group by to_char(df.create_date, 'MM/DD/YYYY')")

```

```

<R25> fsraw = dbGetQuery(db, sql)
#with timeout 120

```

```

<R26> names(fsraw) = c("date", "count", "bytes")

```

```

<R27> fsraw[1:10,]
      date count      bytes
0 01/01/2005 1247 567084952909
1 01/02/2005 1029 477629104477
2 01/03/2005  136 29874210828
3 01/04/2005  100 13977595986
4 01/05/2005  107 15204180560
5 01/06/2005  527 175065214569
6 01/07/2005  303 112926648528
7 01/08/2005  588 263002335778
8 01/09/2005  905 377378266664
9 01/10/2005 1123 523888321583

```

Figure out the id number for d0olc

```

<R28> stations$id[ grep('d0ol', stations$name) ]
[1] 116 117 118 456 466

<R29> stations$name[ grep('d0ol', stations$name) ]
[1] "datalogger-d0olc" "datalogger-d0olb" "datalogger-d0ola"
"datalogger-d0olf"
[5] "datalogger-d0oll"

<R30> fsraw$stationId = 116

```

Join the data frames

```

<R31> fsnoraw = fs

<R32> fs = rbind(fsnoraw, fsraw)

```

Fill in the db server and station name given the id

```

<R33> fss = merge(fs, stations, by.x="stationId", by.y="id", sort=F)

<R79> fss$tBytes = fss$bytes/(1024^4)

```

Let's sum up the counts by date for a total

```

<R36> fss.total = aggregate(fss[c("count", "bytes")],
  by=list(date=fss$date), sum )

<R37> fss.total[1:5,]

```

```

      date count      bytes
1 01/01/2005 2160 1.444438e+12
2 01/02/2005 1099 5.480870e+11
3 01/03/2005 1502 1.266587e+12
4 01/04/2005  631 4.121058e+11
5 01/05/2005 1276 6.634502e+11

```

```
<R38> names(fss.total) = c("date", "counts", "bytes")
```

```
<R39> fss.total[1:10,]
```

```

      date counts      bytes
1 01/01/2005 2160 1.444438e+12
2 01/02/2005 1099 5.480870e+11
3 01/03/2005 1502 1.266587e+12
4 01/04/2005  631 4.121058e+11
5 01/05/2005 1276 6.634502e+11
6 01/06/2005 1146 4.955848e+11
7 01/07/2005  619 3.548254e+11
8 01/08/2005  646 3.059922e+11
9 01/09/2005  939 4.088178e+11
10 01/10/2005 1253 5.690980e+11

```

Convert to GB

```
<R65> fss.total$tBytes = fss.total$bytes/(1024^4)
```

What are the statistics?

```
<R40> summary(fss.total$counts)
```

```

  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   15   1091   1841   2471   3627   8425

```

```
<R66> summary(fss.total$tBytes)
```

```

  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.003057 0.555600 0.882700 0.908600 1.235000 2.598000

```

Set some MSWord variables that we can refer to later

```
<R240> #var fss.max = max(fss.total$counts)
```

```
8425
```

```
<R241> #var fss.mean = format(mean(fss.total$counts), digits=4)
```

```
2471
```

```
<R242> #var fss.max.s = format( max(fss.total$counts)/24/60/60,
  digits=2)
```

```
0.098
```

```
<R243> #var fss.mean.s = format( mean(fss.total$counts)/24/60/60,
  digits=2)
```

```
0.029
```

```
<R244> #var fss.mean.cdf = format( 50.0 /
  (mean(fss.total$counts)/24/60/60), digits=1)
```

1748

```
<R245> #var fss.max.cdf = format( 50.0 /  
  (max(fss.total$counts)/24/60/60), digits=1)
```

513

```
<R70> #var fss.bytes.max = format( max(fss.total$tBytes), digits=3)
```

2.6

```
<R73> #var fss.bytes.mean = format( mean(fss.total$tBytes), digits=2)
```

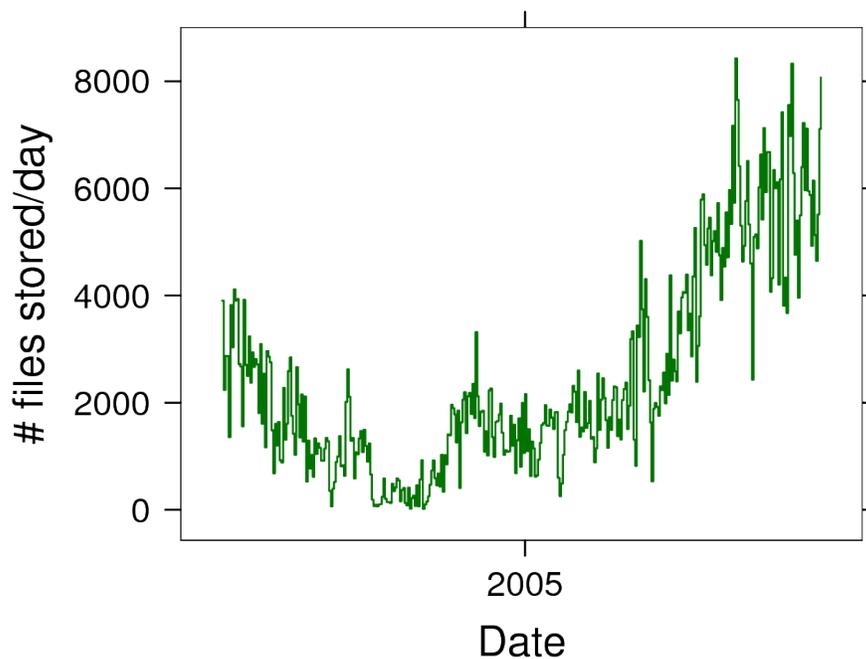
0.9

Convert the dates to POSIX dates

```
<R50> fss.total$pDate = as.POSIXct( strptime(fss.total$date,  
  "%m/%d/%Y") )
```

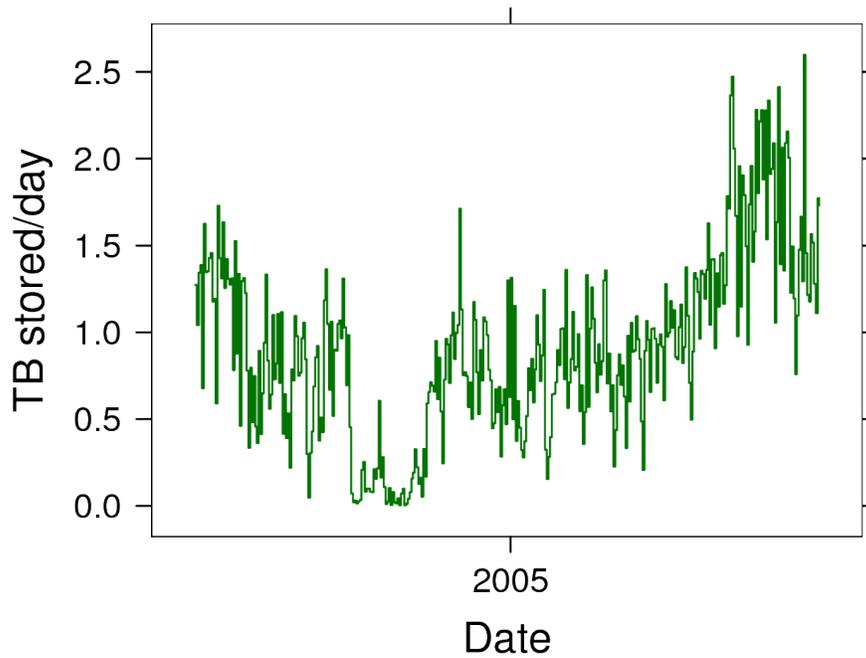
Plot the number of files stored per day

```
<R51> mp(  
  xyplot( counts ~ pDate, data=fss.total, type="s",  
    xlab="Date", ylab="# files stored/day"  
  ),  
  "fss.total.png", h=4, w=5  
)  
#with graphics fss.total.png
```



Do for bytes

```
<R75> mp(
  xyplot( tBytes ~ pDate, data=fss.total, type="s",
          xlab="Date", ylab="TB stored/day"
  ),
  "fss.gBtotal.png", h=4, w=5
)
#with graphics fss.gBtotal.png
```



Split up by DBServers,

```
<R56> fss.dbServer = aggregate(fss[c("count", "bytes")],
  by=list(dbServer=fss$dbServer), sum )
```

```
<R57> names(fss.dbServer) = c("dbServer", "count", "bytes")
```

```
<R58> fss.dbServer
```

```

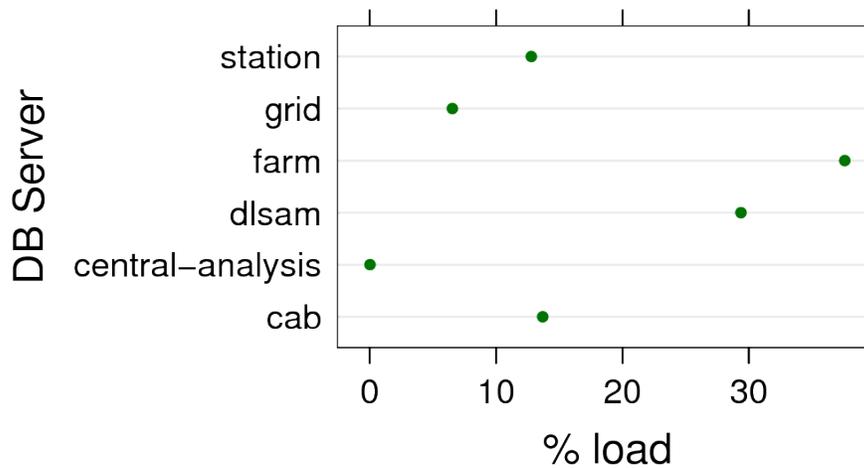
      dbServer count      bytes
1          cab 123095 9.603301e+13
2 central-analysis  268 3.004949e+11
3          dlsam 264164 1.202864e+14
4          farm 337995 9.494016e+13
5          grid  58849 1.325098e+13
6          station 115016 3.881195e+13

```

```
<R78> fss.dbServer$tBytes = fss.dbServer$bytes/(1024^4)
```

```
<R60> fss.dbServer$perc = fss.dbServer$count / sum(fss.dbServer$count)
      * 100
```

```
<R61> mp(
  dotplot( dbServer ~ perc, data=fss.dbServer, xlab="% load",
           ylab="DB Server"),
  "fss.dbServer.png", h=3, w=5)
#with graphics fss.dbServer.png
```

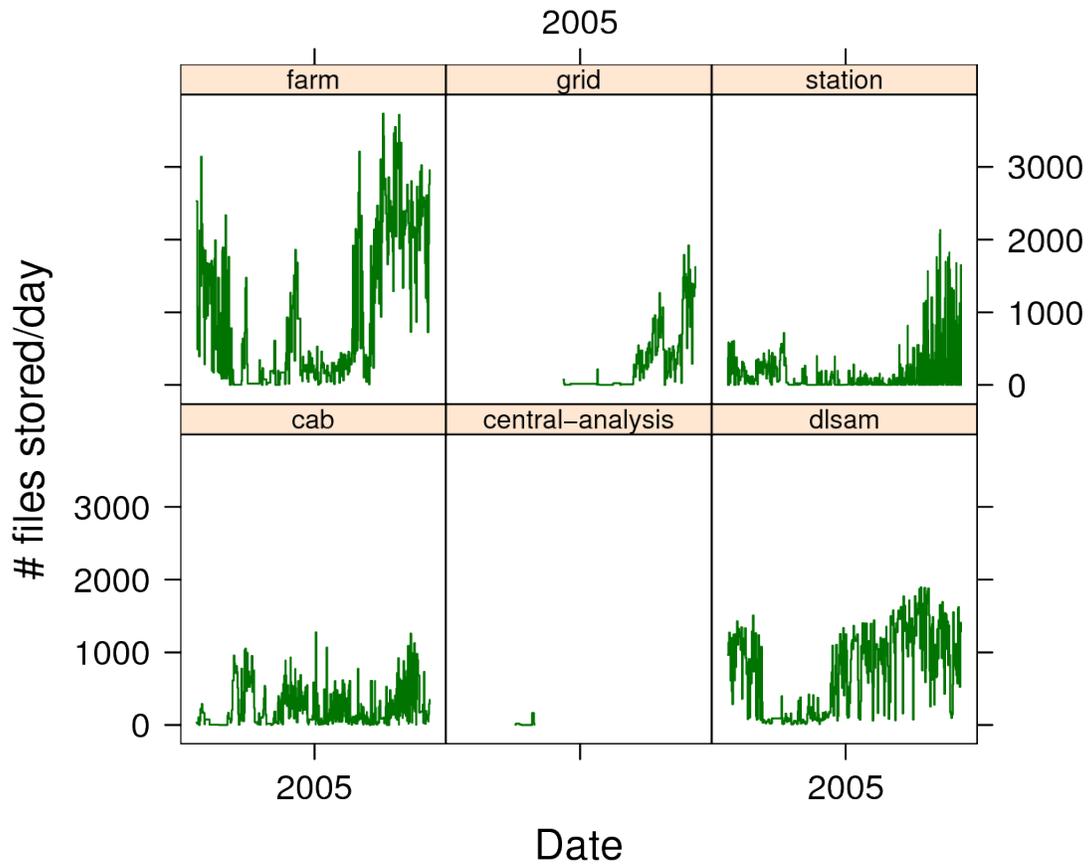


```
<R62> fss$pDate = as.POSIXct( strptime(fss$date, "%m/%d/%Y") )
```

```

<R63> mp(
  xyplot( count ~ pDate | dbServer, data=fss, type="s",
    xlab="Date", ylab="# files stored/day",
    par.strip.text=list(cex=0.6)
  ),
  "fss.splitByDbServer.png", h=5, w=6
)
#with graphics fss.splitByDbServer.png

```



```

<R80> mp(
  xyplot( tBytes ~ pDate | dbServer, data=fss, type="s",
    xlab="Date", ylab="TB stored/day",
    par.strip.text=list(cex=0.6)
  ),
  "fss.tBsplitByDbServer.png", h=5, w=6
)
#with graphics fss.tBsplitByDbServer.png

```

