

Timing Studies for SAM getMetadata command

Tests to measure the capabilities of a v7_3_0 SAM dbserver were performed. These tests are documented in this note. This testing occurred on the CDF Phase I SAM farm worker nodes. To perform these tests, single CAF jobs were run on fncdf CondorCAF, a varying number of jobs sections were run with between 10 and 1000. These jobs used the SAM DB server user_int (v7_3_0) running on cdfsam05.fnal.gov. Table 1 lists the number of sections, type of test, start and stop times. During most of the tests there was no attempt to ensure that the getMetadata commands were synchronized between job sections; the python code for these tests is shown in appendix A. Two of the tests were designed to have at least the first getMetadata command in each section to occur at the same time; the python code for this test is in appendix B. It should be noted that in order to try to have the getMetadata commands to occur concurrently many of the job sections had to sleep for a significant period of time in order to account for the behaviour of the Condor job scheduler.

Table 1 List of SAM getMetadata tests performed

	Type of test	# of job sections	Submit time	End time
1	getMetadata	10	Sun Aug 21 15:48:11 2005	Sun Aug 21 16:08:53 2005
2	getMetadata	100	Sun Aug 21 20:25:02 2005	Sun Aug 21 21:25:08 2005
3	getMetadata	100	Thu Sep 1 07:02:40 2005	Thu Sep 1 07:42:44 2005
4	getMetadata	399	Thu Sep 1 08:05:13 2005	Thu Sep 1 10:24:17 2005
5	Concurrent getMetadata	10-50 jobs	Sun Sep 4 22:01:00 2005	Mon Sep 5 00:54:00 2005

Test 1: – 10 CAF job sections running on Phase I farm nodes

This section presents the results from the tests reading the metadata from the integration database using the SAM DB server with 10 job sections running at once. There was a 10 second sleep between getMetadata commands.

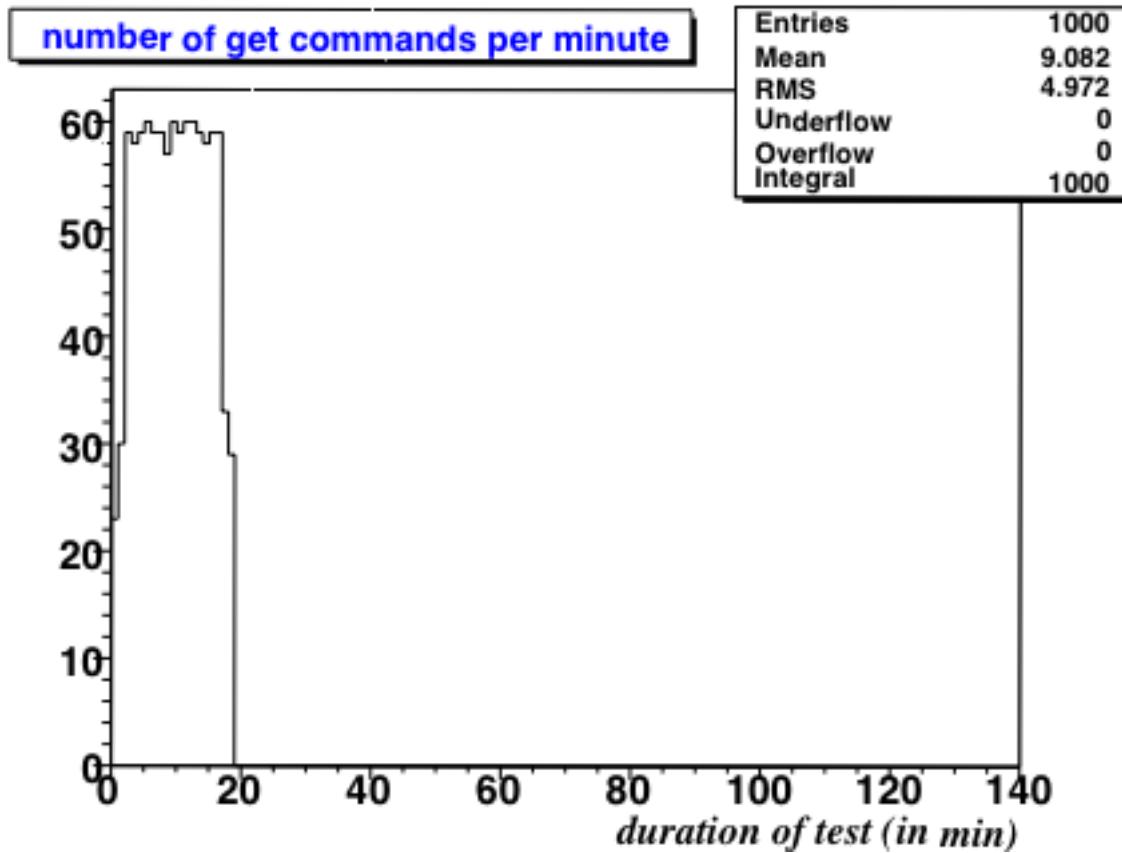


Figure 1 - Number getMetadata commands per minute. The plateau is an artifact of having 10 second sleep between getMetadata commands. The time during the test is along the x axis. There were 10 job sections running at once.

Figure one shows the number of getMetadata commands executed per minute. Within a single job there is a 10 second wait between getMetadata commands. With 10 concurrent job sections running the maximum rate can be up to 60 jobs per minute as indicated by the plot.

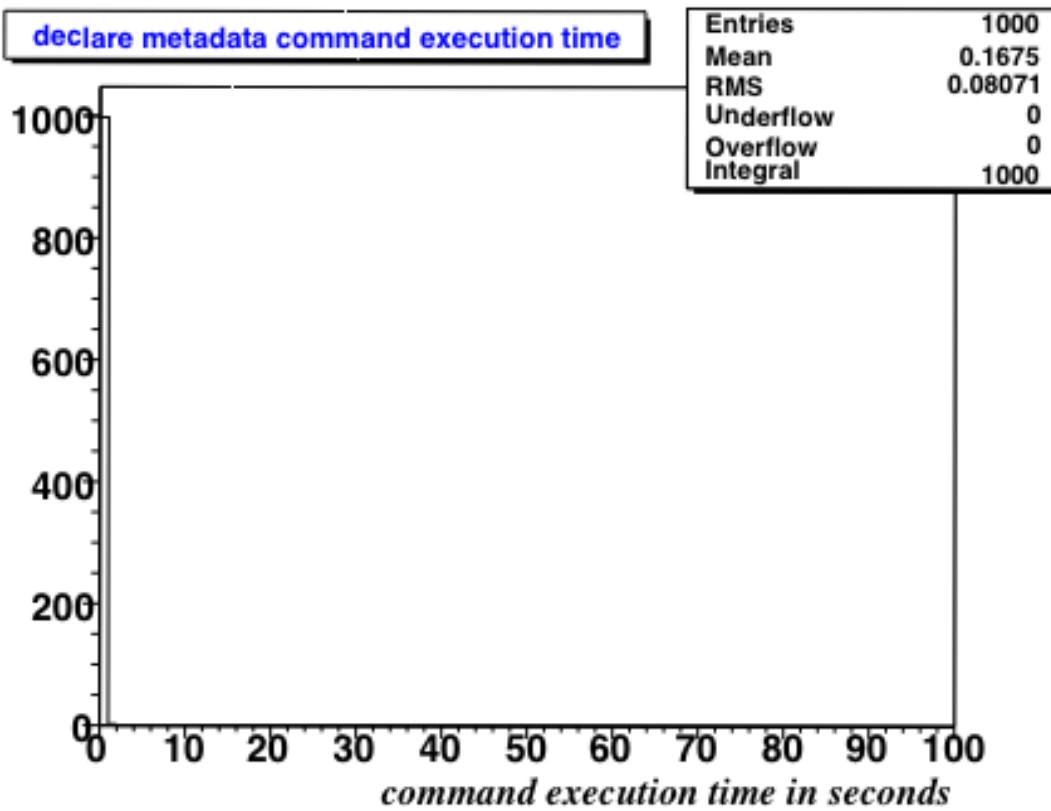


Figure 2 - getMetadata command execution time. Each command took a fraction of a second to complete. The spike in the first bin indicates that the DB server can clearly handle this load. Based on 10 job sections running at once.

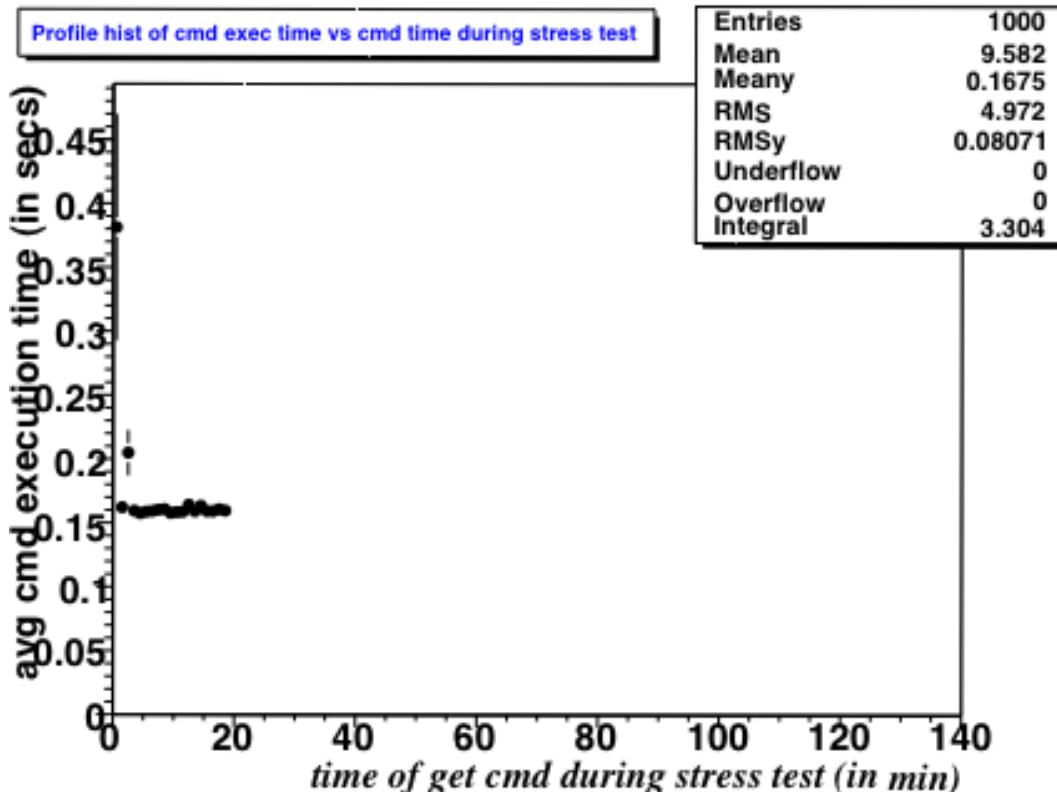


Figure 3 GetMetadata command execution time vs time. Time is measured from the start of the test. Figure 2 is the y axis projection of this figure. 100 jobs were submitted simultaneously. Based on 10 job sections running at once.

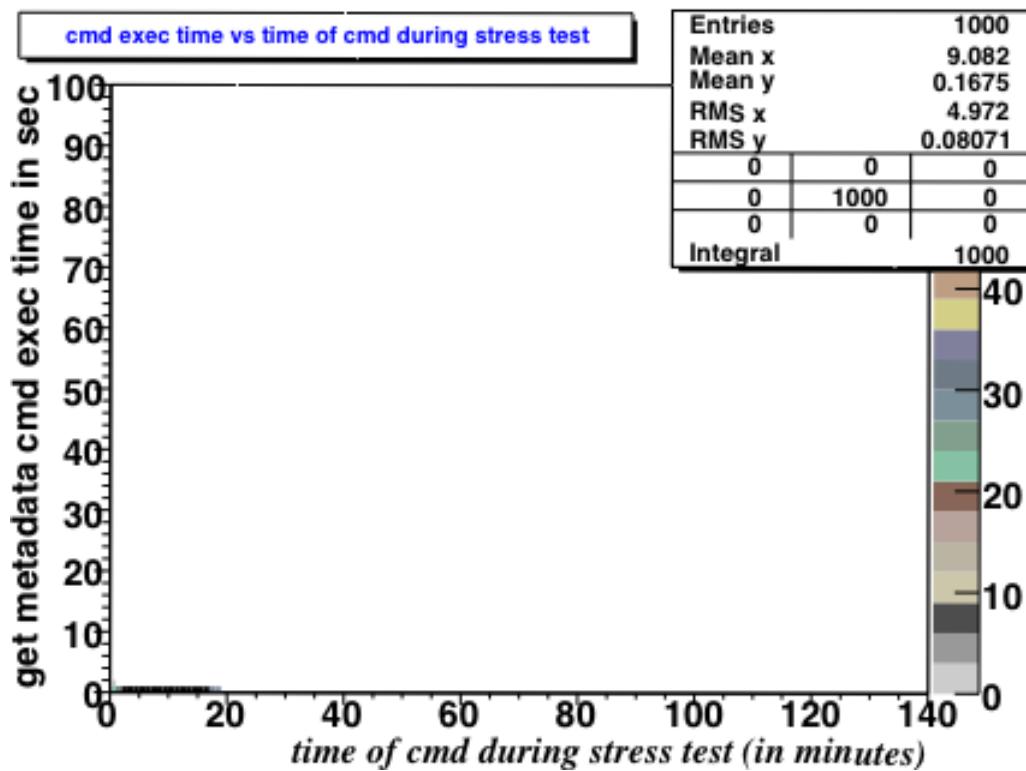


Figure 4 - Two dimensional distribution of getMetadata command execution time vs time during test. Figure 2 is the projection along the y axis. Based on 10 job sections running at once.

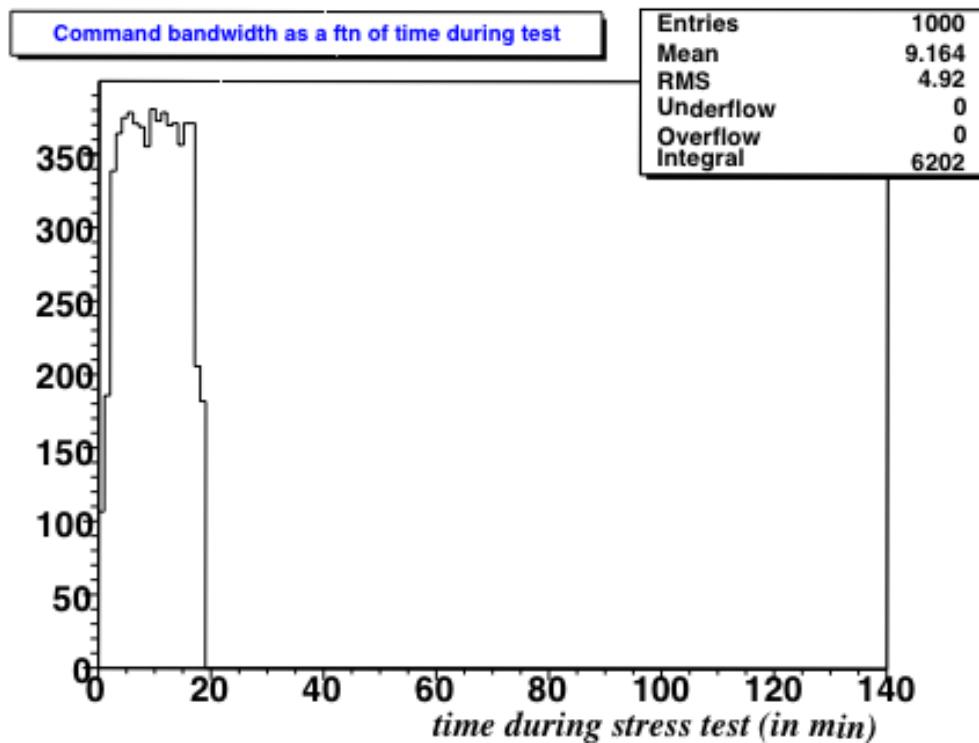


Figure 5 - Estimation of the number of commands possible per minute for the getMetadata command. This is calculated by the reciprocal of the command execution time and number of commands per minute. Based on 10 job sections running at once

Test 2: – 100 CAF job sections running on Phase I farm nodes

This section presents the results from the tests reading the metadata from the integration database using the SAM DB server with 10 job sections running at once. There was a 10 second sleep between getMetadata commands.

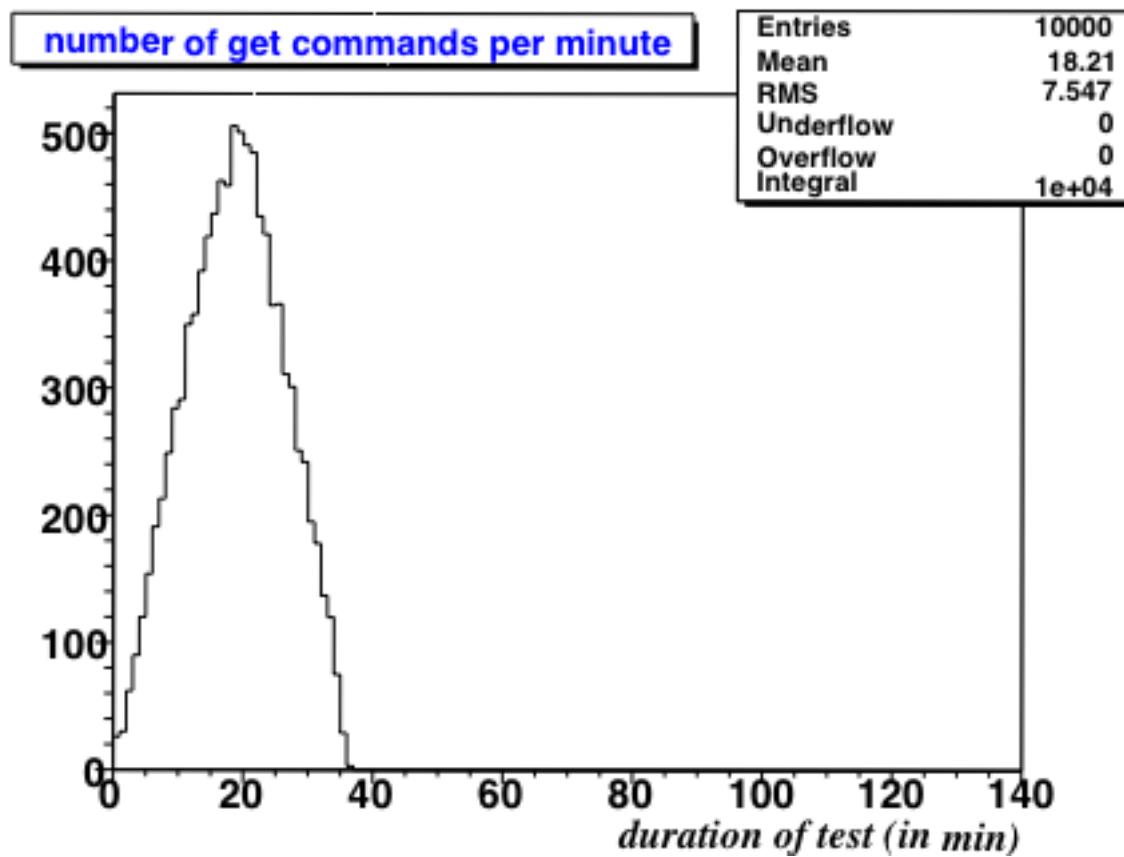


Figure 6 - Number getMetadata commands per minute. The plateau is an artifact of having 10 second sleep between getMetadata commands. The time during the test is along the x axis. There were 100 job sections running at once.

Figure 6 shows the number of getMetadata commands executed per minute. Within a single job there is a 10 second wait between getMetadata commands. With 100 concurrent job sections running the maximum rate can be up to 600 jobs per minute. The maximum possible is not quite reached. The rise time (left edge) of the plot is due to condor job scheduler. The next figure shows that the vast majority of the command occurred within a fraction of a second.

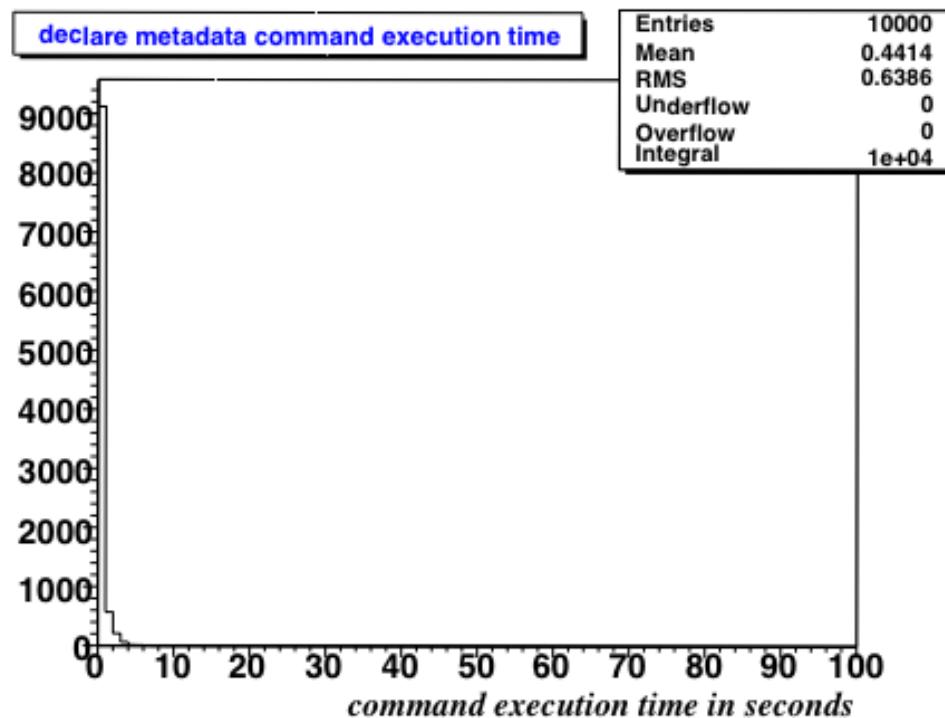


Figure 7 - getMetadata command execution time. Each command took a fraction of a second to complete. The spike in the first bin indicates that the DB server can clearly handle this load. Based on 100 job sections running at once.

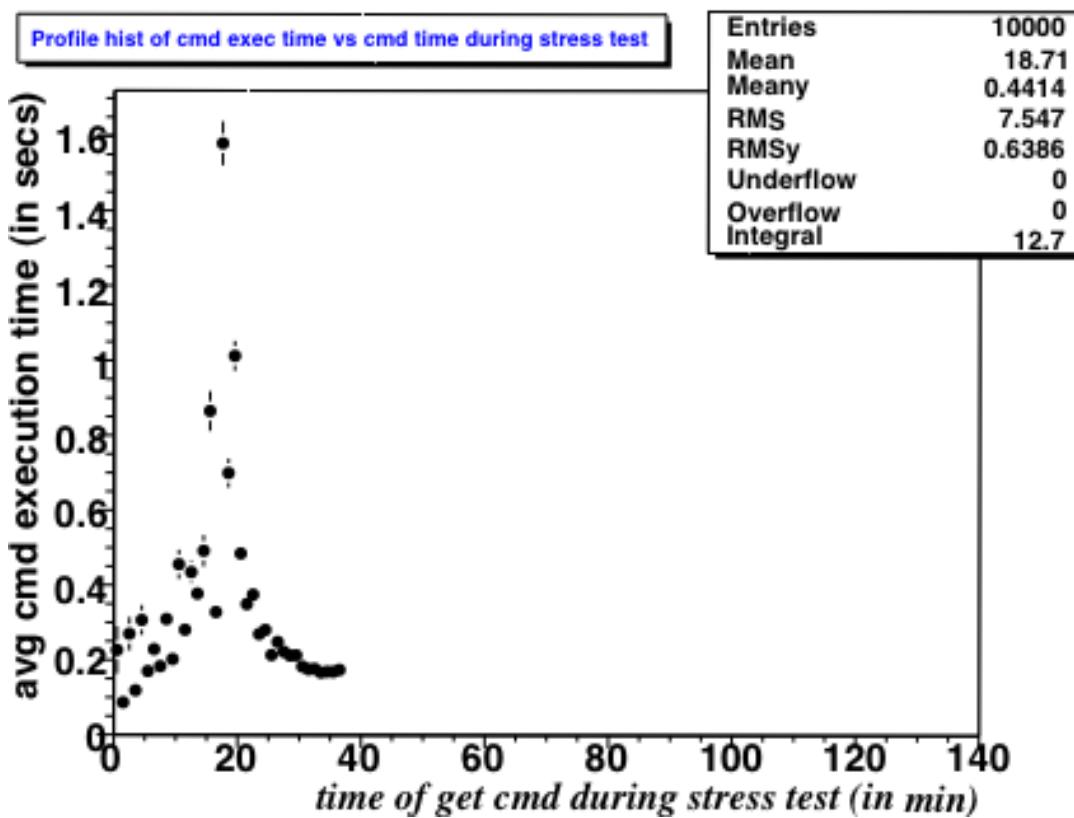


Figure 8 GetMetadata command execution time vs time. Time is measured from the start of the test. Figure 7 is the y axis projection of this figure. 100 jobs were submitted simultaneously. Based on 100 job sections running at once.

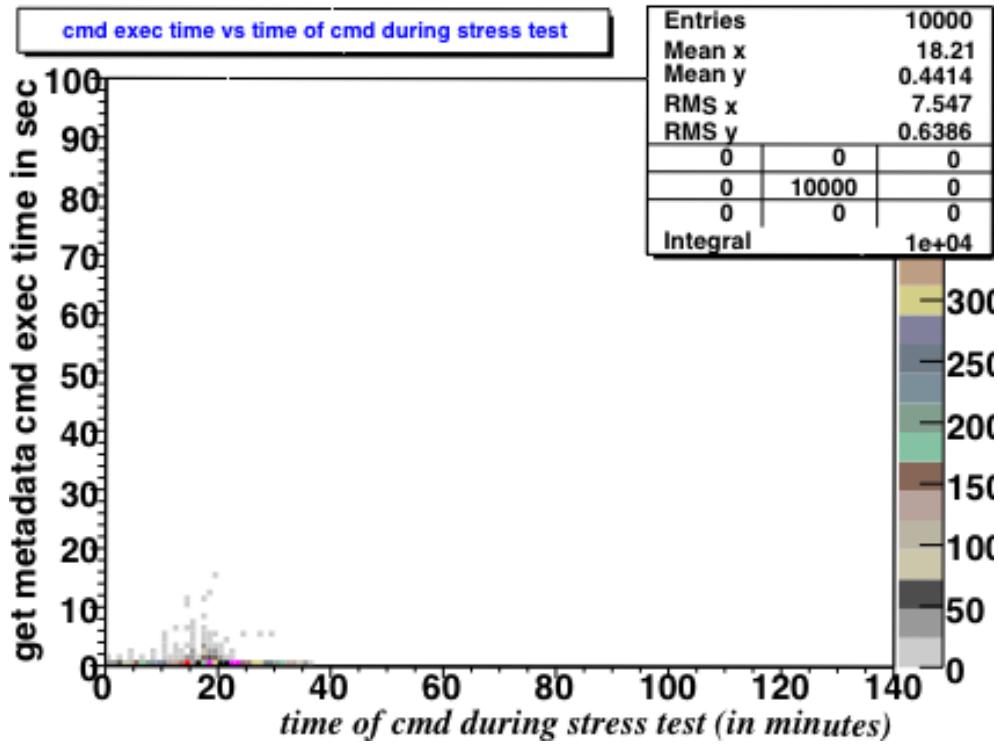


Figure 9 - Two dimensional distribution of getMetadata command execution time vs time during test. Figure 7 is the projection along the y axis. Based on 100 job sections running at once.

In looking at figures 6-9, one can see that once the number of commands per minute exceed ~ 450 getMetadata commands per minute the DB server appears to slow down. This may be due to the fact that during the time of these tests there were only 5 allowed concurrent connections into the ORACLE database. In addition there was a limit to the number of threads to ORACLE the db server had.

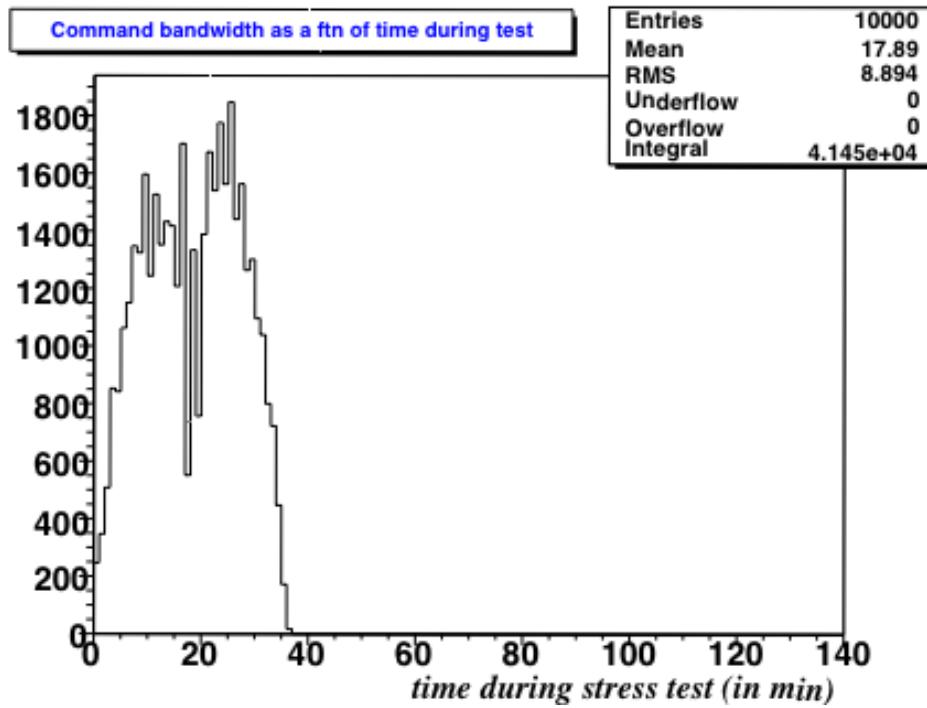


Figure 10 - Estimation of the number of commands possible per minute for the getMetadata command. This is calculated by the reciprocal of the command execution time and number of commands per minute. Based on 100 job sections running at once

Test 3: – 100 CAF job sections running on Phase I farm nodes

This section presents the results from the tests reading the metadata from the integration database using the SAM DB server with 100 job sections running at once. There was a 10 second sleep between getMetadata commands. The DB server and number of connections to the Integration Oracle database were upgraded between test 2 and test 3

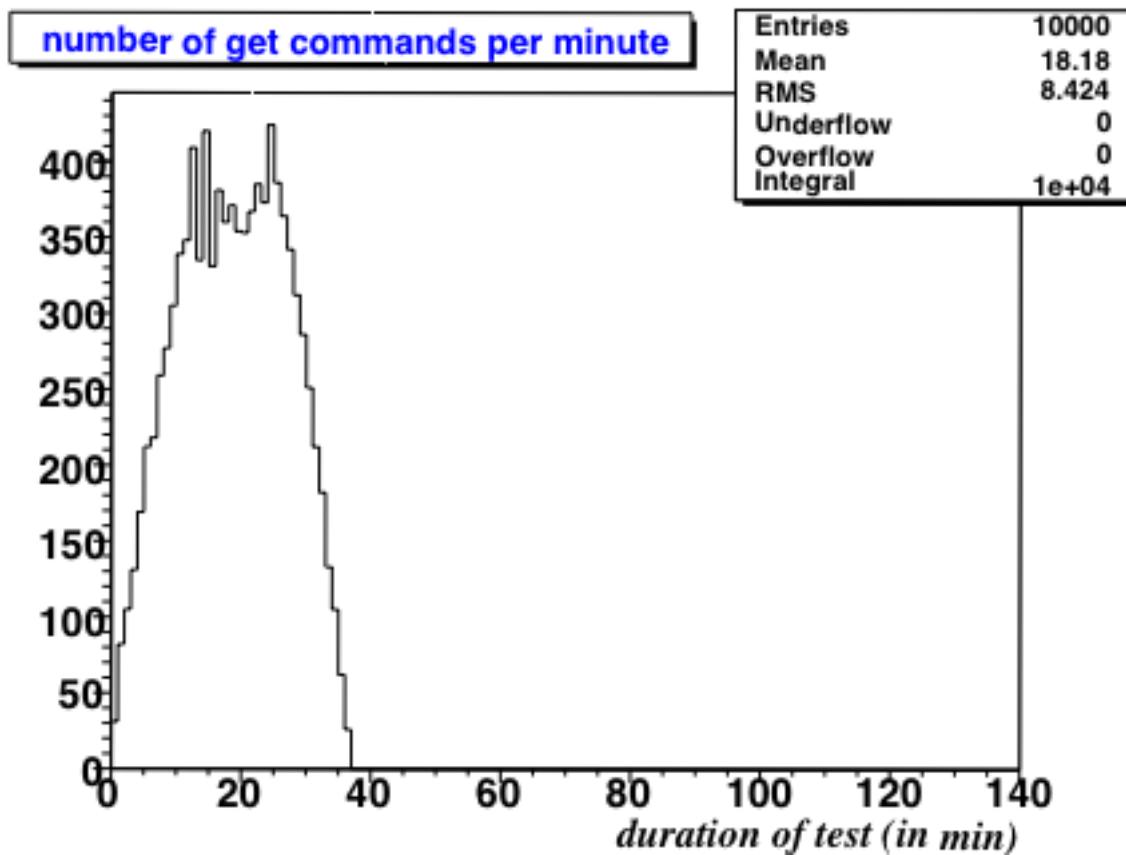


Figure 11 - Number getMetadata commands per minute. The plateau is an artifact of having 10 second sleep between getMetadata commands. The time during the test is along the x axis. There were 100 job sections running at once.

Figure 11 shows the number of getMetadata commands executed per minute. Within a single job there is a 10 second wait between getMetadata commands. With 100 concurrent job sections running the maximum rate can be up to 600 jobs per minute. The maximum possible is not quite reached. The rise time (left edge) of the plot is due to condor job scheduler. The next figure shows that the vast majority of the command occurred within a fraction of a second.

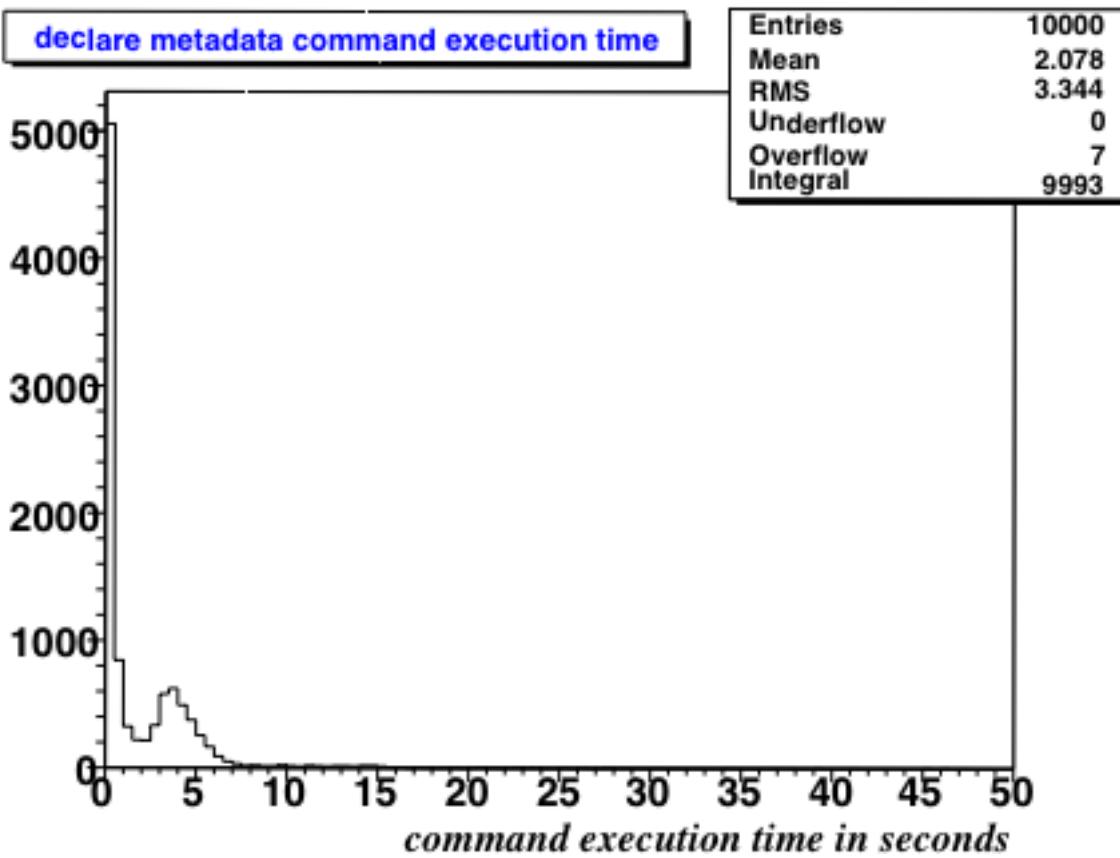


Figure 12 - `getMetadata` command execution time. The overloading of the db server is beginning to become apparent from the appearance of the second peak. Based on 100 job sections running at once. The histogram title is incorrect.

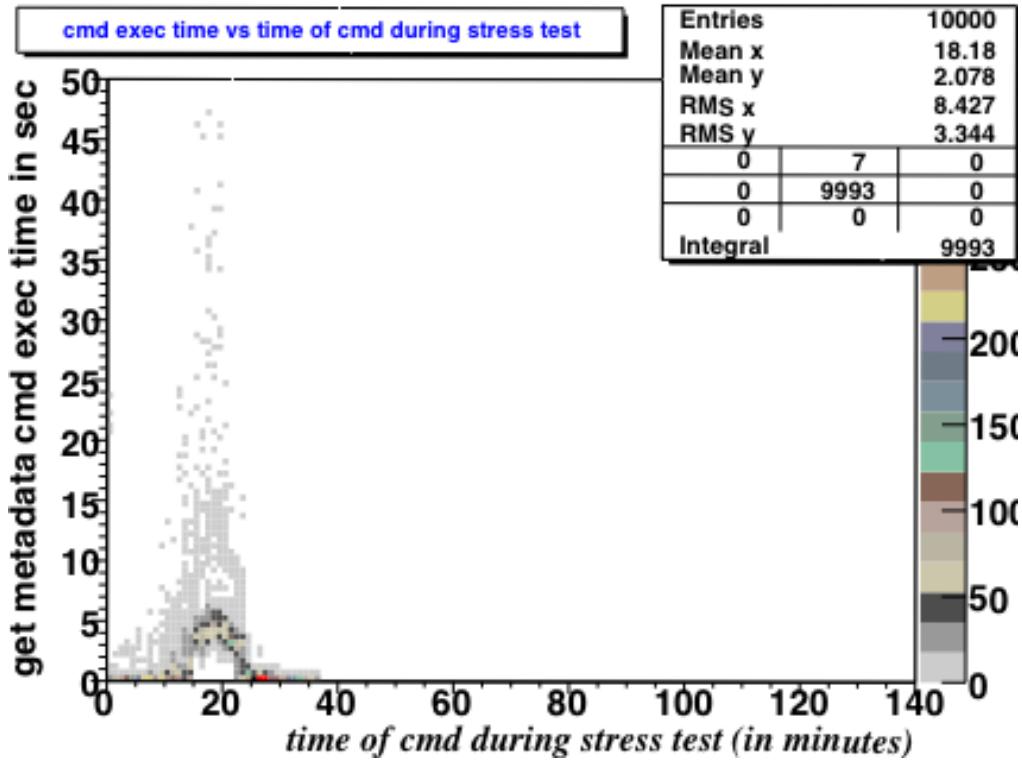


Figure 13 GetMetadata command execution time vs time. Time is measured from the start of the test. Figure 12 is the y axis projection of this figure. 100 jobs were submitted simultaneously. Based on 100 job sections running at once.

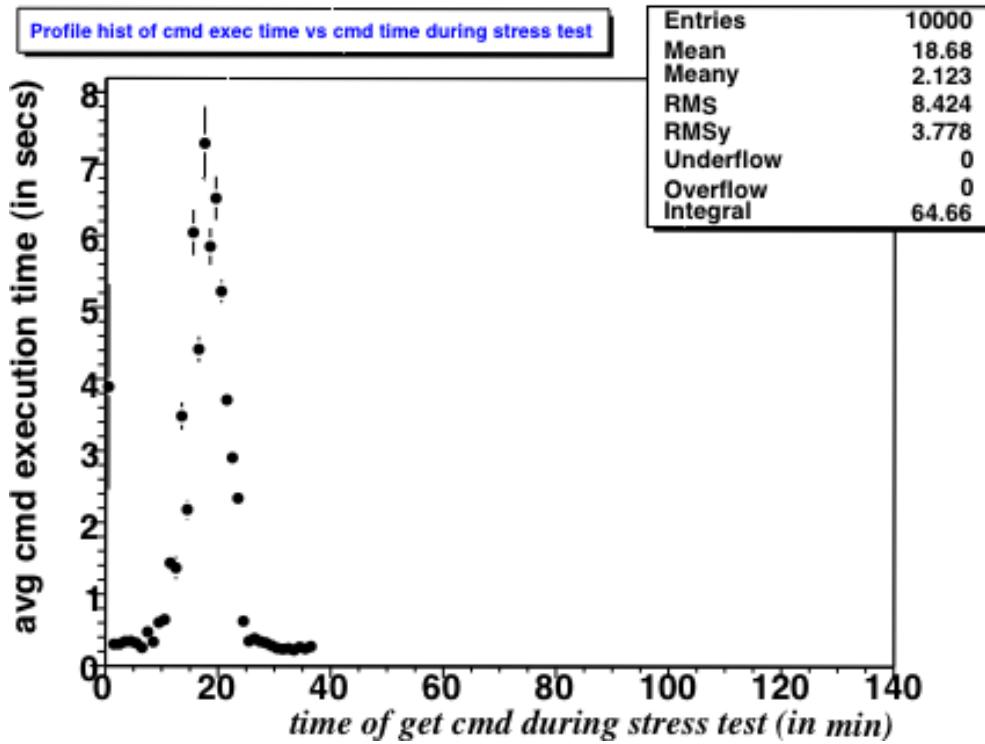


Figure 14 - Profile distribution of getMetadata command execution time vs time during test. Figure 7 is the projection along the y axis. Based on 100 job sections running at once.

In looking at figures 11-14, one can see that once the number of commands per minute exceed ~ 300 getMetadata commands per minute the DB server appears to slow down

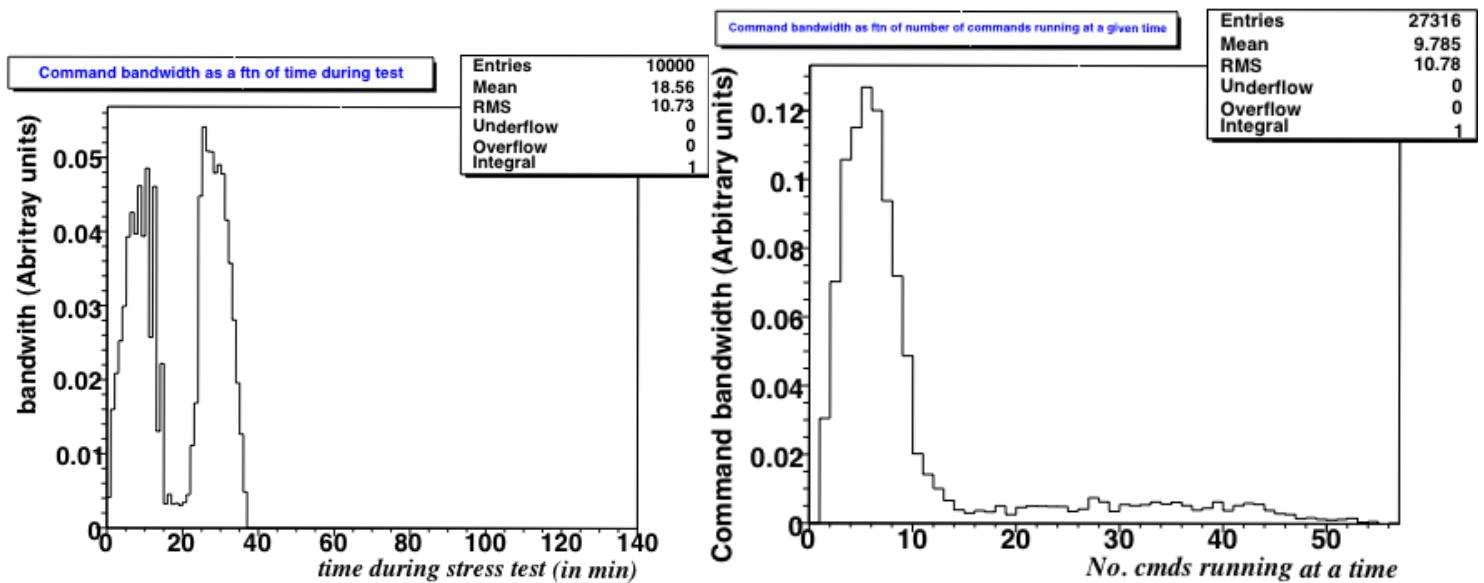


Figure 15 - Estimation of the number of commands possible per minute for the getMetadata command. This is calculated by the reciprocal of the command execution time and number of commands per minute. Based on 100 job sections running at once The left figure shows the command bandwidth as a ftn of time during the test and the right figure shows the bandwidth as a ftn of the number of commands running at once

Figure 15 shows that when the db server is over loaded the amount of commands that it can handle drops very quickly. Once away from the threshold the db server performance increases remarkably.

Test 4: – 399 CAF job sections running on Phase I farm nodes

This section presents the results from the tests reading the metadata from the integration database using the SAM DB server with 399 job sections running at once. There was a 10 second sleep between getMetadata commands.

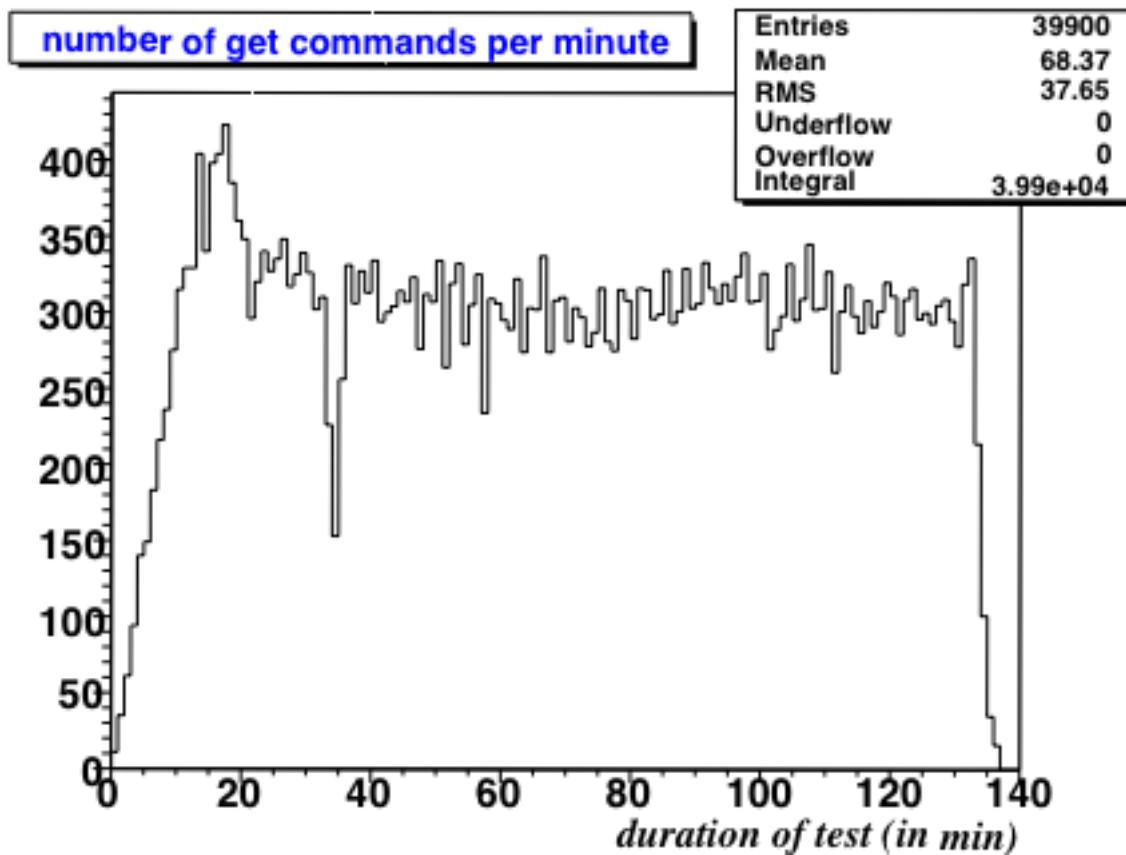


Figure 16 - Number getMetadata commands per minute. The plateau is an artifact of having 10 second sleep between getMetadata commands. The time during the test is along the x axis. There were 100 job sections running at once.

Figure 16 shows the number of getMetadata commands executed per minute. Within a single job there is a 10 second wait between getMetadata commands. With 399 concurrent job sections running the maximum rate can be up to 2394 jobs per minute. The maximum possible is not quite reached. The rise time (left edge) of the plot is due to condor job scheduler. The peak on the left edge of the plot indicates that the db server saturates and then reaches a plateau.

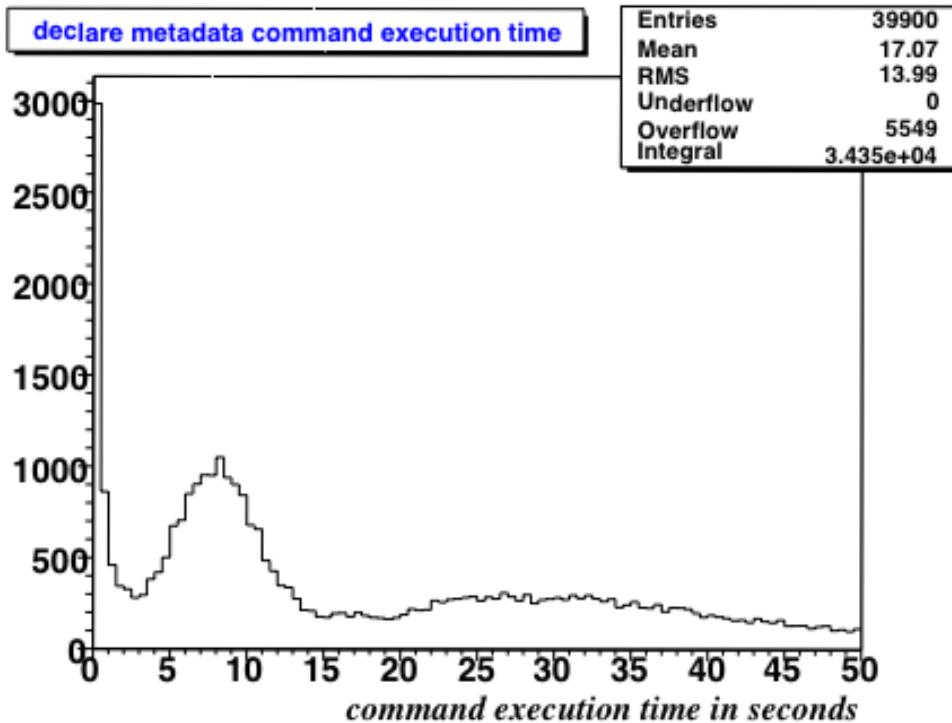


Figure 17 - getMetadata command execution time. Based on 399 job sections running at once. The histogram title is incorrect.

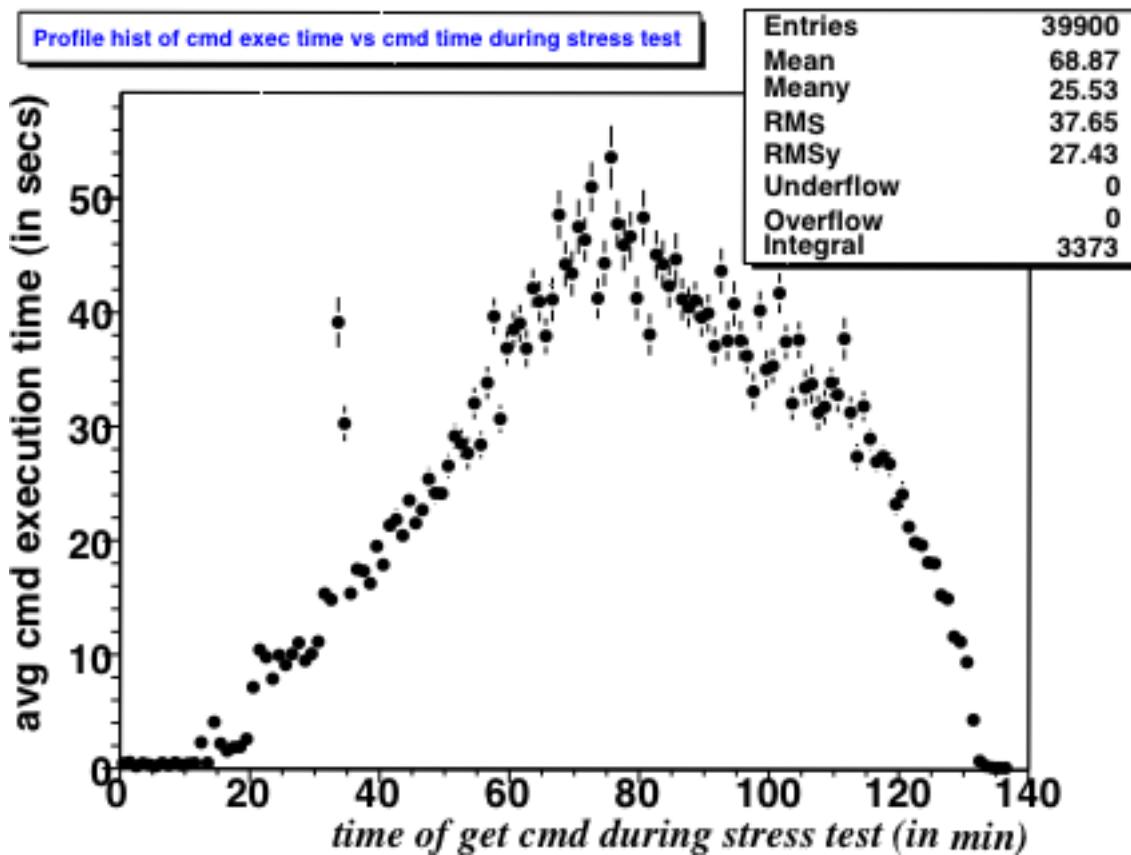
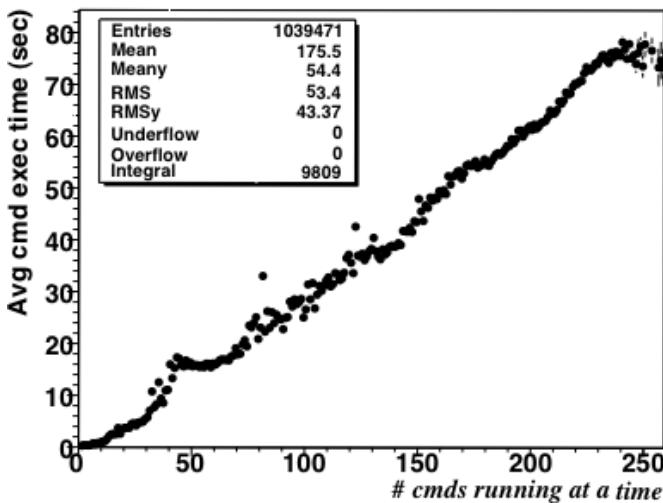
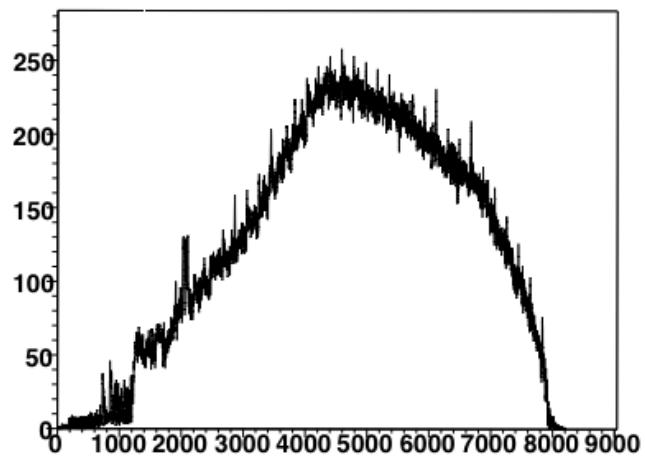


Figure 18 - Profile distribution of getMetadata command execution time vs time during test. Figure 7 is the projection along the y axis. Based on 100 job sections running at once.

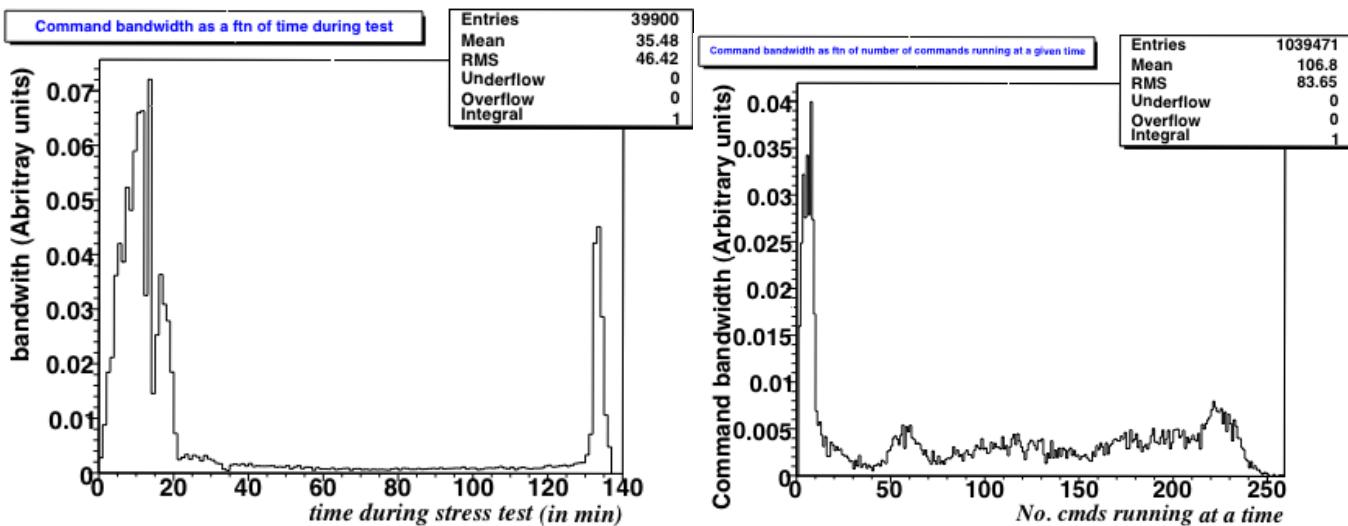
Profile hist of cmd exec time vs num of cmds at once



Graph

**Figure 19** GetMetadata command execution time vs number of commands executed running at once. Based on 399 job sections running at once. The number of commands running at once is shown by the right graph

In looking at figures 16-19, one can see that once the number of commands running at once exceeds ~ 40 commands; the DB server appears to slow down and the average execution time for the commands continues to increase. This can also be seen in the command bandwidth plots:

**Figure 18** - Estimation of the number of commands possible per minute for the getMetadata command. This is calculated by the reciprocal of the command execution time and number of commands per minute. Based on 399 job sections running at once. The left figure shows the command bandwidth as a ftn of time during the test and the right figure shows the bandwidth as a ftn of the number of commands running at once

Test 5: – Concurrent getMetadata commands

During these tests between 10 and ~50 getMetadata commands were run concurrently. The python scripts are located in appendix B. The tests were repeat four times for each (16 runs) to check the variability of the measurements.

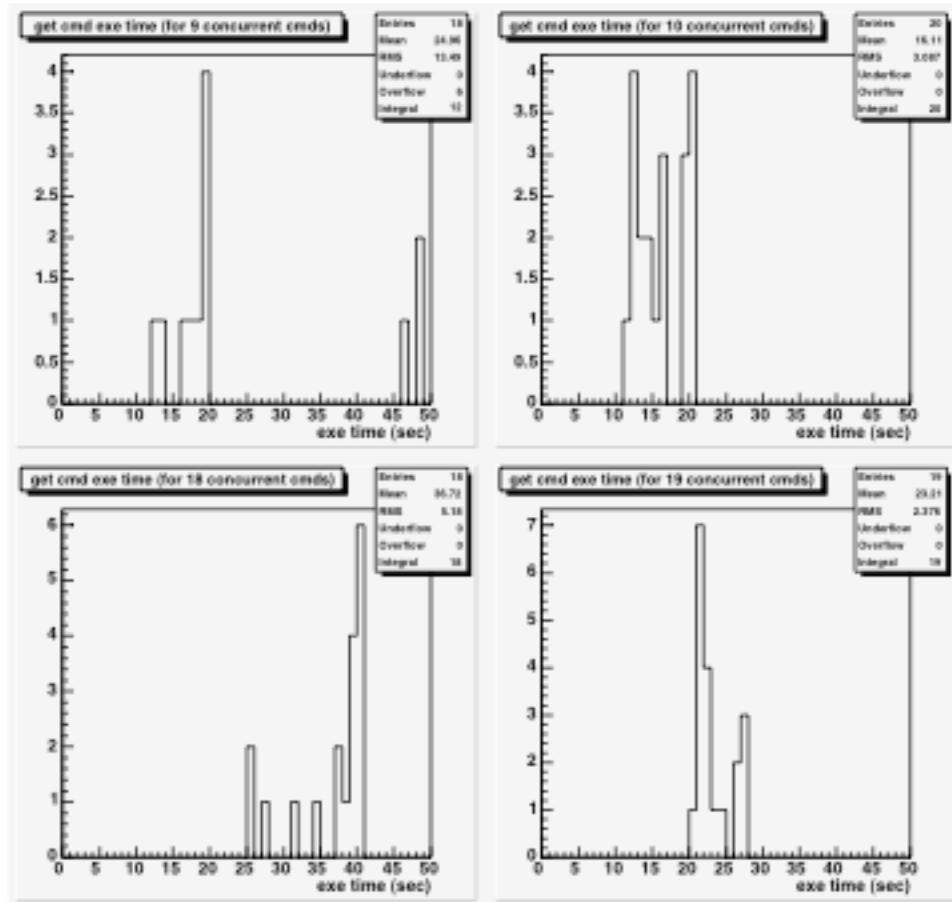


Figure 19 - Execution time for concurrent getMetadata commands

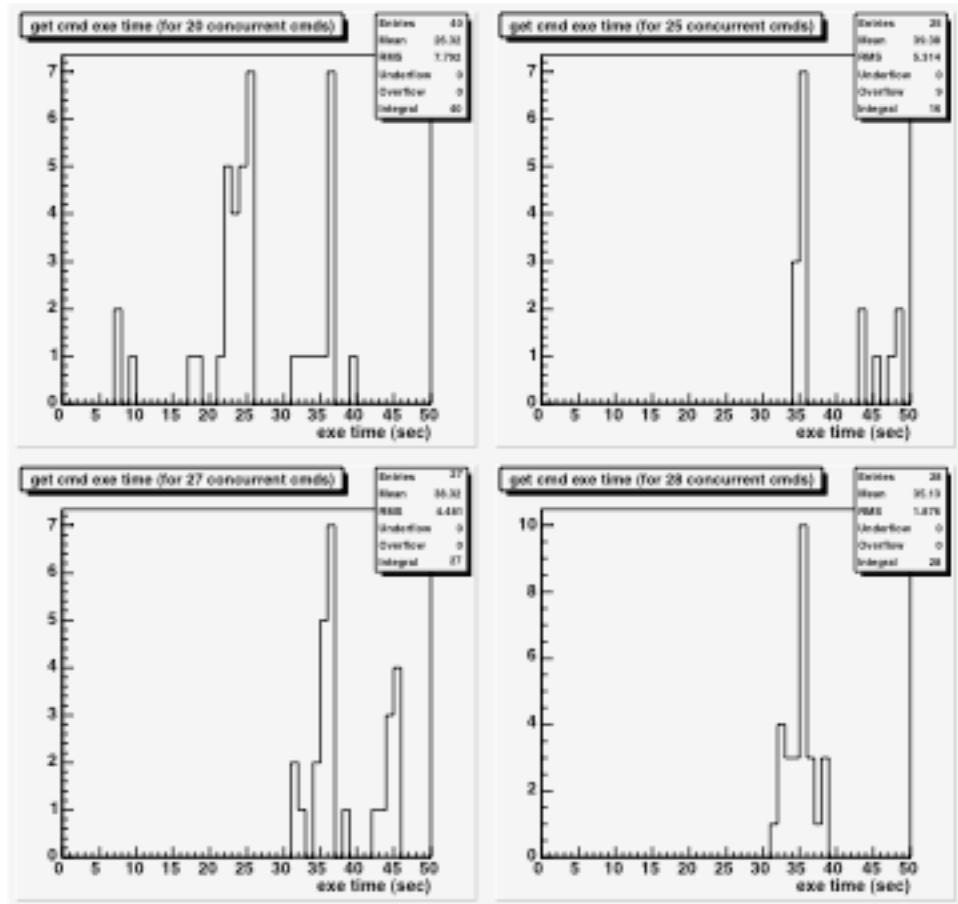


Figure 20 - Execution time for concurrent `getMetadata` commands

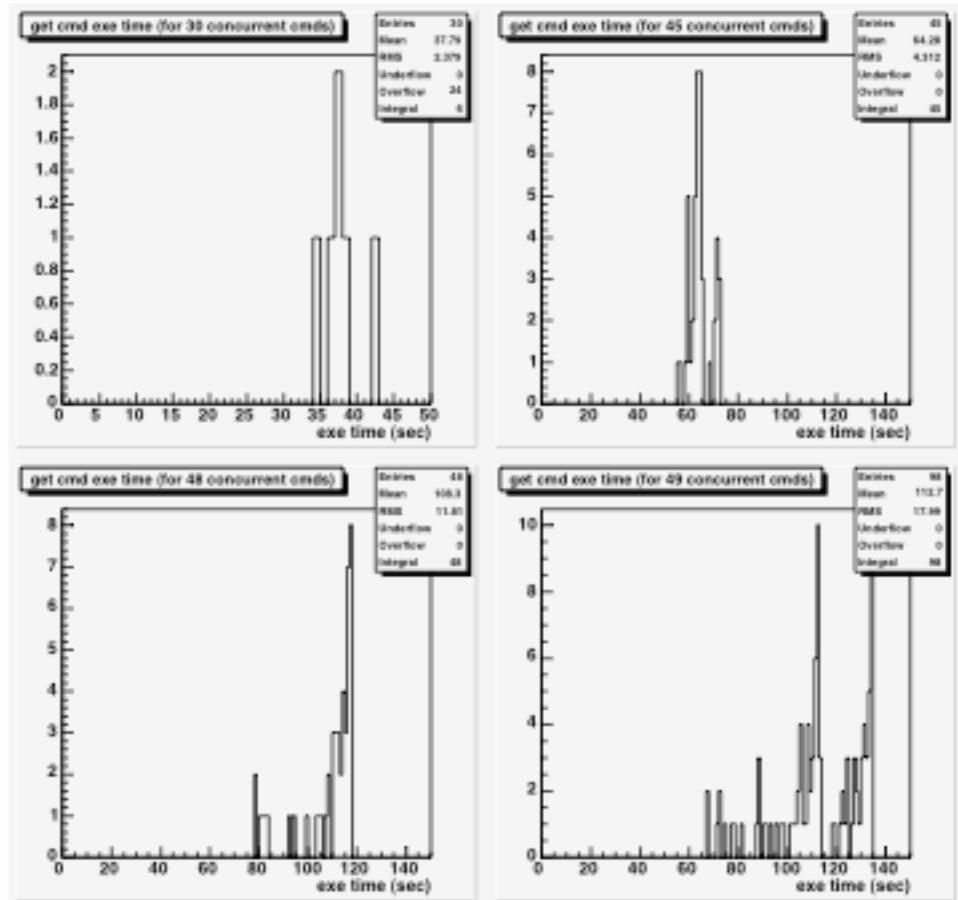


Figure 21 - Execution time for concurrent `getMetadata` commands

Profile hist of cmd exec time vs number of concurrent commands

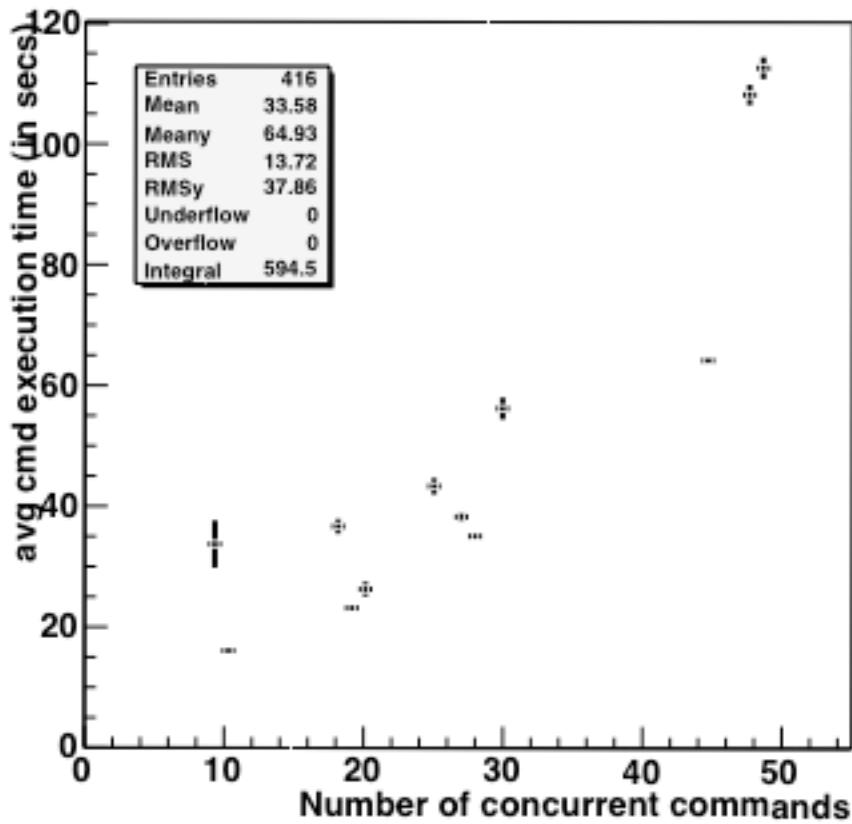
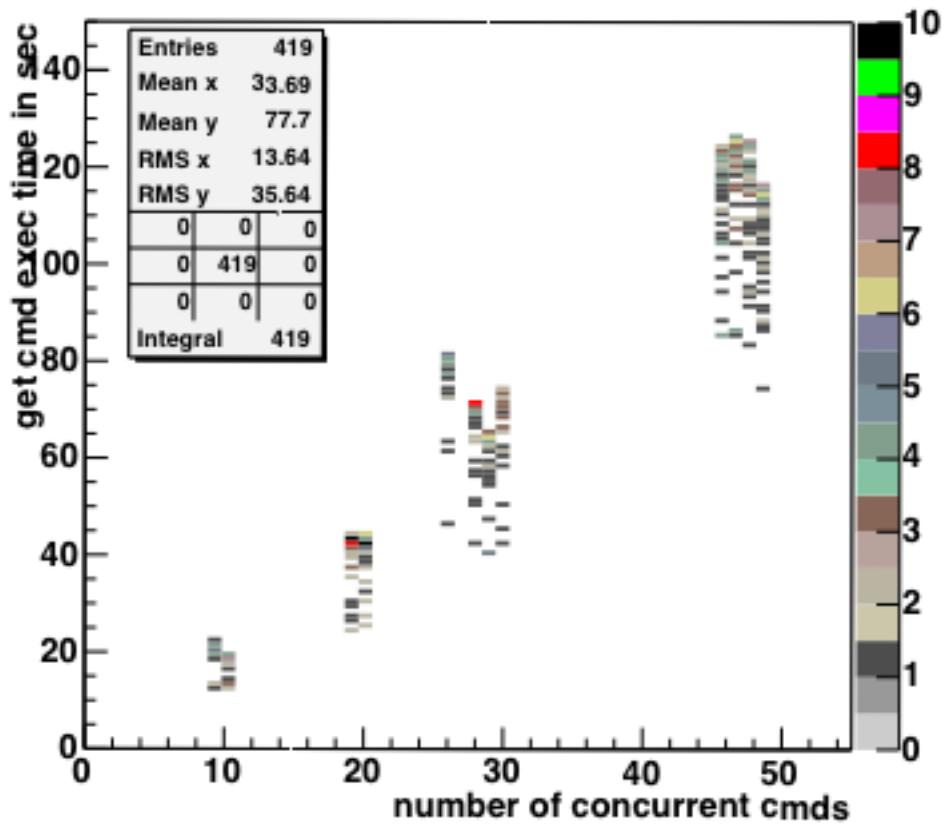


Figure 22 Avg getMetadata command execution vs number of running commands.

cmd exec time(sec) vs number of concurrent cmd's during stress test



The average command excution time for the runs is list here:

9 cmd's mean = 24.958 RMS = 13.491

10 cmd's mean = 16.1136 RMS = 3.08689

18 cmd's mean = 36.716 RMS = 5.17959

19 cmd's mean = 23.21 RMS = 2.37592

20 cmd's mean = 26.325 RMS = 7.79163

25 cmd's mean = 39.3783 RMS = 5.31353

27 cmd's mean = 38.3153 RMS = 4.48081

28 cmd's mean = 35.127 RMS = 1.87646

30 cmd's mean = 37.7937 RMS = 2.3785

45 cmd's mean = 64.2827 RMS = 4.31231

48 cmd's mean = 108.282 RMS = 11.8142

49 cmd's mean = 112.72 RMS = 17.9912

Appendix A

Listing of get-metadata.py

```
#!/usr/bin/env sampy
#
#      declare-metadata.py
#
#      script to do a declare of dummy metadata many times as a test.
#
import string
import sys
import os
import getopt
import commands
import time
import SAM
from Sam import sam
from SamFile.SamDataFile import SamDataFile
from SamException import SamExceptions
from SamStruct.SamBoolean import SamBoolean
from SamStruct.DbServantConnectionInfoList_v2 import
DbServantConnectionInfoList_v2

import stat
from time import gmtime, strftime, localtime

#
#  first check that sam environment has been setup
#
job_start=time.time()
job_starttime=time.clock()
job_start_string = '%s' % strftime("%Y-%m-%d %H:%M:%S", localtime(job_start))

print 'Job get metadata started: %s  % (job_start_string)

try:
    sam2 = os.environ['SETUP_SAM']
except:
    print "Error: sam not set. 'setup sam' before running me"
    sys.exit(1)

dbservername = os.environ['SAM_DB_SERVER_NAME']

job_elapsed_time = 0

#
#sam_station=os.environ['SAM_STATION']
#sam_project=os.environ['SAM_PROJECT']
#sam_dataset=os.environ['SAM_DATASET']

#
#  parse the options
#
file_limit=5000
```

```

try:
    optlist, args = getopt.getopt(sys.argv[1:], 'f', ['file_limit='])
except getopt.GetoptError, e:
    sys.exit(1)

for key, value in optlist:
    if key == '--file_limit':
        file_limit=long(value)

print 'Number of metadata declares = %d' %(file_limit)

#
# Establish create the bulk of the dummy metadata =====
#
# =====
# and here is the loop over the files
#
# =====
ifile = 0
caf_section = int(os.environ['CAF_SECTION'])
new_caf_section = caf_section
if caf_section > 200 :
    new_caf_section = caf_section % 200
    file_limit = 100
    if caf_section > 200 and caf_section <= 400 :
        ifile = 100
    if caf_section > 400 and caf_section <= 600 :
        ifile = 200
    if caf_section > 600 and caf_section <= 800 :
        ifile = 300
    if caf_section > 800 and caf_section <= 1000 :
        ifile = 400

caf_num = 10000*new_caf_section

file_number=1
while file_number <= file_limit:
    #      create the dummy file name
    file_id_no = caf_num + file_number + ifile
    filename = 'test_file_' + str(file_id_no)

    # build metadata
    metadata = {}

        file_number = file_number + 1
task_start=time.time()
starttime = time.time()

```

```
metadata = sam.getMetadata(fileName=filename)
stoptime=time.time()
elapsed_time = stoptime-starttime
starttime_string = '%s' %strptime( "%Y%m%d %H%M%S",localtime(task_start))
print 'getMetadata %s %.3f %d'
%(starttime_string,elapsed_time,len(metadata))

# now sleep for 10 seconds

time.sleep(10)

#
# get the total elapsed time etc.
#

job_stop=time.time()
job_stoptime=time.time()

job_stoptime_string = '%s' %strptime( "%Y-%m-%d %H:%M:%S",localtime(job_stop))
job_elapsed_time = job_stop-job_start

print 'get-metadata finished at %s - it took %.3f secs to run python script
' %(job_stoptime_string,job_elapsed_time)
```

Appendix B

Listing of get-metadata-blast.py

```
#!/usr/bin/env sampy
#
#      declare-metadata.py
#
#      script to do a declare of dummy metadata many times as a test.
#
import string
import sys
import os
import getopt
import commands
import time
import SAM
from Sam import sam
from SamFile.SamDataFile import SamDataFile
from SamException import SamExceptions
from SamStruct.SamBoolean import SamBoolean
from SamStruct.DbServantConnectionInfoList_v2 import
DbServantConnectionInfoList_v2

import stat
from time import gmtime, strftime, localtime

#
#  first check that sam environment has been setup
#
job_start=time.time()
job_starttime=time.clock()
job_start_string = '%s' % strftime("%Y-%m-%d %H:%M:%S", localtime(job_start))

print 'Job get metadata started: %s  % (job_start_string)

try:
    sam2 = os.environ['SETUP_SAM']
except:
    print "Error: sam not set. 'setup sam' before running me"
    sys.exit(1)

dbservername = os.environ['SAM_DB_SERVER_NAME']

job_elapsed_time = 0

#sam_station=os.environ['SAM_STATION']
#sam_project=os.environ['SAM_PROJECT']
#sam_dataset=os.environ['SAM_DATASET']

#
#  parse the options
#
```

```

file_limit=5000

try:
    optlist, args = getopt.getopt(sys.argv[1:], 'f', ['file_limit='])
except getopt.GetoptError, e:
    sys.exit(1)

for key, value in optlist:
    if key == '--file_limit':
        file_limit=long(value)

print 'Number of metadata declares = %d' %(file_limit)

#
# Establish create the bulk of the dummy metadata =====
#
# and here is the loop over the files
#
=====

now = time.time()
min = int(strftime("%M",localtime(now)))
sec = int(strftime("%S",localtime(now)))
if min < 50:
    sleep_sec = (60-sec) + 60*(49-min)
else:
    sleep_sec = (60-sec) + 60*(79-min)

#sleep_sec = (60-sec) + 60*(79-min)

print ' now = %s sleep_sec = %d'
%(strftime("%H:%M:%S",localtime(now)),sleep_sec)

time.sleep(sleep_sec)

ifile = 0
caf_section = int(os.environ['CAF_SECTION'])
new_caf_section = caf_section
if caf_section > 200 :
    new_caf_section = caf_section % 200
    file_limit = 100
    if   caf_section > 200 and  caf_section <= 400 :
        ifile = 100
    if   caf_section > 400 and caf_section <= 600 :
        ifile = 200
    if   caf_section > 600 and caf_section <= 800 :
        ifile = 300

```

```

if  caf_section > 800 and caf_section <= 1000 :
    ifile = 400

caf_num = 10000*new_caf_section

file_number=1
while file_number <= file_limit:
    #      create the dummy file name
    file_id_no = caf_num + file_number + ifile
    filename = 'test_file_' + str(file_id_no)

    # build metadata
    metadata = {}

        file_number = file_number + 1
    task_start=time.time()
    starttime = time.time()
    metadata = sam.getMetadata(fileName=filename)
    stoptime=time.time()
    elapsed_time = stoptime-starttime
    starttime_string = '%s' %strptime("%Y%m%d %H%M%S",localtime(task_start))
    print 'getMetadata %s %.3f %d'
%(starttime_string,elapsed_time,len(metadata))

#     now sleep for 10 seconds

        time.sleep(10)

#
#  get the total elapsed time etc.
#

job_stop=time.time()
job_stoptime=time.time()

job_stoptime_string = '%s' %strptime("%Y-%m-%d %H:%M:%S",localtime(job_stop))
job_elapsed_time = job_stop-job_start

print 'get-metadata finished at %s - it took %.3f secs to run python script
' %(job_stoptime_string,job_elapsed_time)

```