File system support for fast virus checking

Contents –

REDUCER : A TOOL To Reduce Redundant Disk I/O

Abstract

Most of the people use virus scanners and security applications such as anti spywares ,rootkit hunters to secure their PC.Majority of population also use backup programs to backup their data and prevent data loss.One concern regarding these applications is that they all need to scan the system as part of their functioning. Scanning process demands lots of disk reads,which implies these applications are **disk I/O intensive** and tend to put a lot of load on the disk.Moreover,when these applications are installed on a single system for different purposes they are meant for,system is read multiple times ,one time corresponding to each application,thus inducing **redundant Disk I/O .**

In this paper,we create a pipeline to reduce redundant Disk I/O activity and make this process more efficient.We tried to achieve co-ordination between different scanning applications and tried to schedule them using round robin scheme.

Trick is to make use of buffer cache to reduce Disk I/O.Accessing buffered file blocks is much cheaper100-1000 times cheaper than accessing the same block from the disk.When a scanning application does some disk reads, file blocks read will be cached for some time,this fact can be exploited and used to reduce disk I/O ,if another scanning application which also need to read these file blocks as part of its functioning, access blocks from buffer cache (cached due to an earlier scanning activity),instead of reading these same blocks again from the disk and ,thus reducing disk I/O.We achieve this ,by scheduling the scans according to a scheduling algorithm namely round robin scheme and have been able to observe significant disk I/O reduction ,about 50% in some cases and have also,consequently, seen improvement in combined scanning times ,which drops almost to half in one case.

1. Introduction

Anti-virus software,anti spywares ,backup programs have become an integral part of one's life.
These are the essential utilities which one must have on his system ,to keep his system safe and secure and avoid system failures and downtime.One concern regarding these applications ,is that they need to scan the system as part of their functioning.Consequently,these tasks are disk I/O intensive.
When installed on a system they induce redundant Disk I/O activity as shown in figure 1.
We aim to reduce redundant Disk I/O activity as shown in figure 2

Redundant Disk I/O –

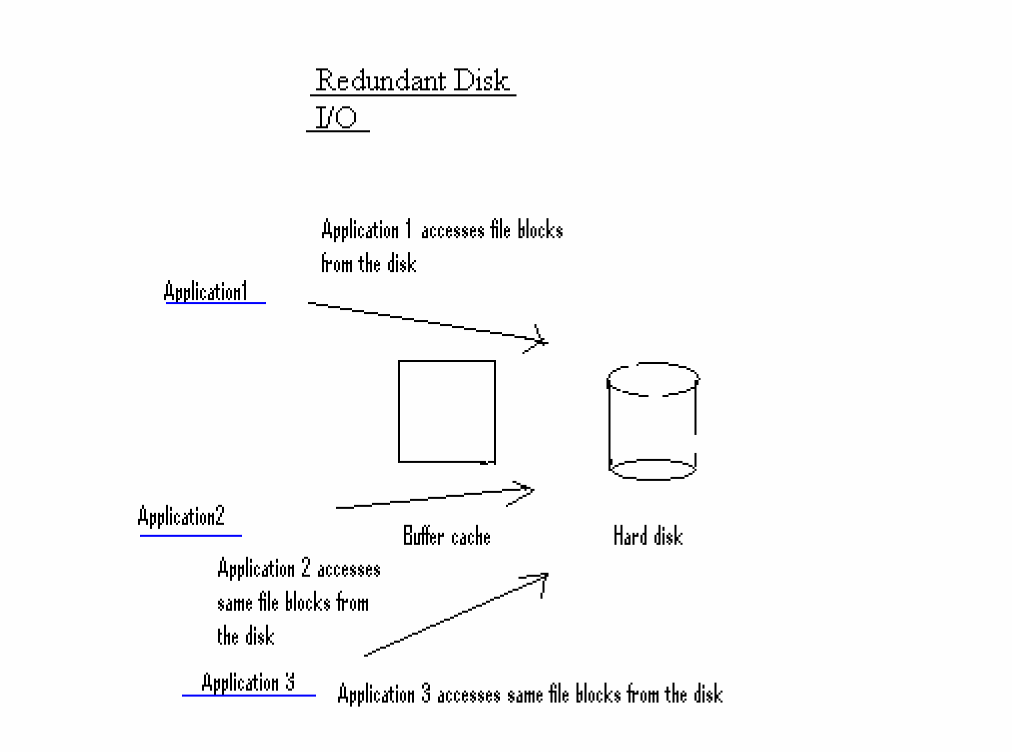
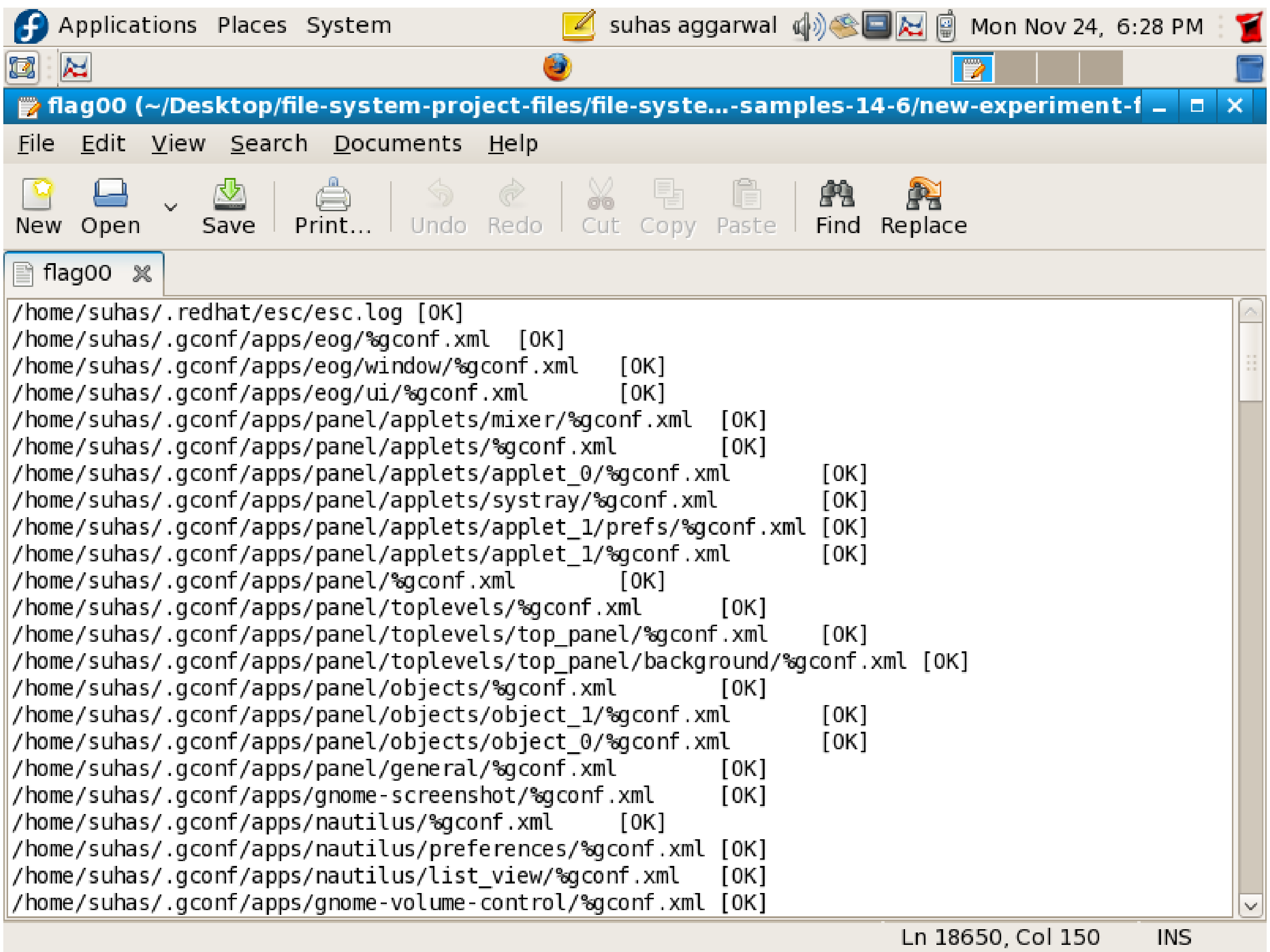


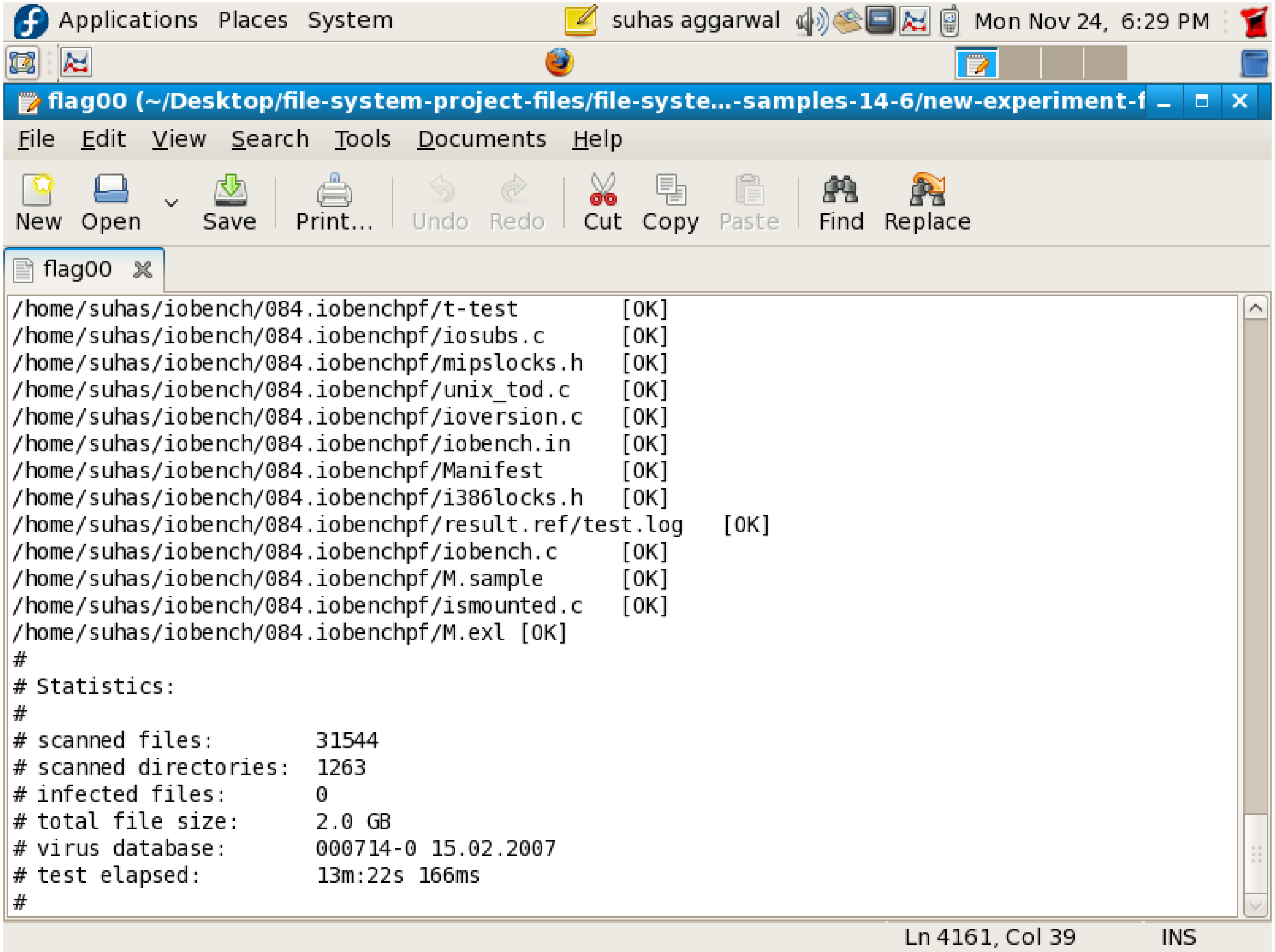
Figure 1

There was almost no related work in this area and we had to start from scratch.
We divided our work in different phases.
We conducted some experiments to study the behaviour of malware checkers involves trapping file system calls ,monitoring I/O activity ,doing scan analysis like time to do a full system scan, types of files being read by these scanners.
Some sample scans we obtained on fedora system by Avast and ClamAV antivirus software.

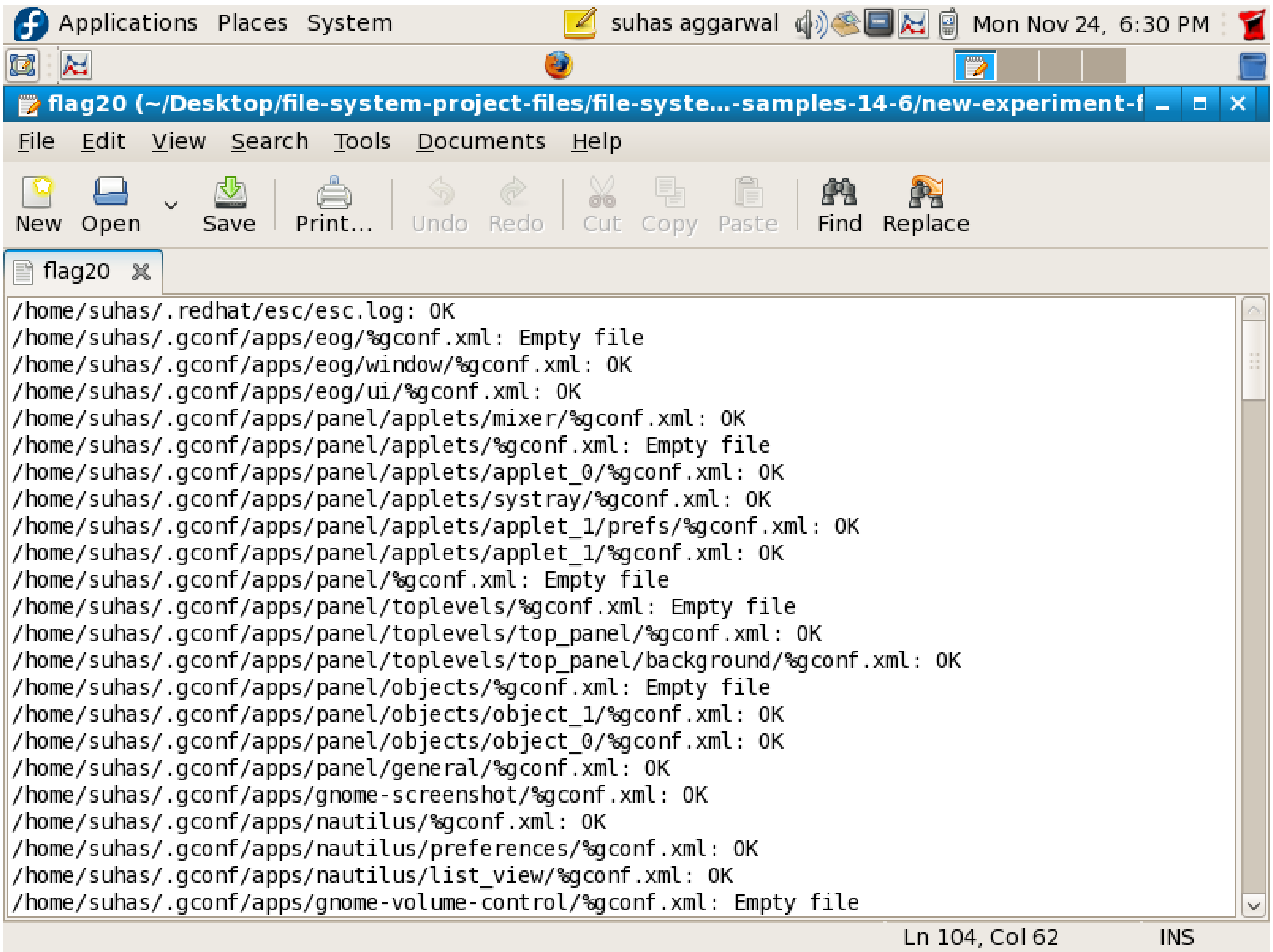
Scans -

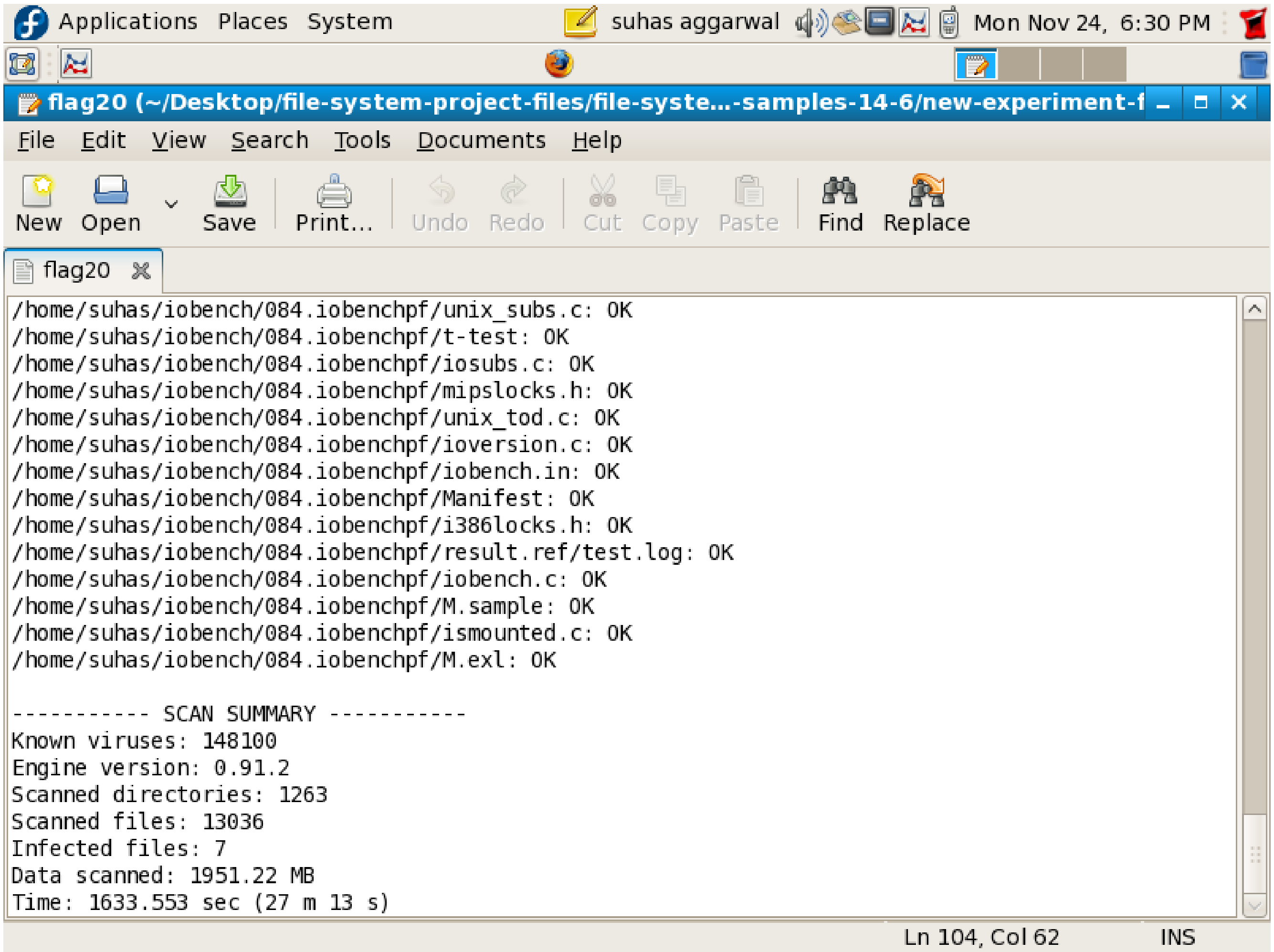
Avast Antivirus




Applications Places System
suhas aggarwal
Mon Nov 24, 6:29 PM
flag00 (~/Desktop/file-system-project-files/file-syste...-samples-14-6/new-experiment-f
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
flag00
/home/suhas/iobench/084.iobenchpf/t-test [OK]
/home/suhas/iobench/084.iobenchpf/iosubs.c [OK]
/home/suhas/iobench/084.iobenchpf/mipslocks.h [OK]
/home/suhas/iobench/084.iobenchpf/unix_tod.c [OK]
/home/suhas/iobench/084.iobenchpf/ioversion.c [OK]
/home/suhas/iobench/084.iobenchpf/iobench.in [OK]
/home/suhas/iobench/084.iobenchpf/Manifest [OK]
/home/suhas/iobench/084.iobenchpf/i386locks.h [OK]
/home/suhas/iobench/084.iobenchpf/result.ref/test.log [OK]
/home/suhas/iobench/084.iobenchpf/iobench.c [OK]
/home/suhas/iobench/084.iobenchpf/M.sample [OK]
/home/suhas/iobench/084.iobenchpf/ismounted.c [OK]
/home/suhas/iobench/084.iobenchpf/M.exl [OK]
#
Statistics:
#
scanned files: 31544
scanned directories: 1263
infected files: 0
total file size: 2.0 GB
virus database: 000714-0 15.02.2007
test elapsed: 13m:22s 166ms
#
Ln 4161, Col 39
INS

ClamAV antivirus





After that we designed and implemented a scheduling service which will work to co-ordinate different scanning applications.We were able to implement a friendly API to which scanners can be interfaced with no scanner code editing requirements.

Finally,we conducted experiment to demonstrate reduced disk I/O .

2. Main Principle -

Main concept behind our implementaton is to make use of file system caching. Disk accesses are very expensive.When a file is read its blocks are buffered in the cache for a while. Accessing a buffered block is much cheaper about(100-1000 times cheaper) than accessing the same block from the disk,we exploited this difference.
When a set of files is read, we give every tool an opportunity to scan it, if it wishes to.
This is the cheapest time to do the scan because disk blocks are likely to be in the cache.
We implemented a prototype which is a schedular service which runs to co-ordinate the scans and reduce redundant Disk I/O activity.

Scheduling the scans with a round robin scheme -

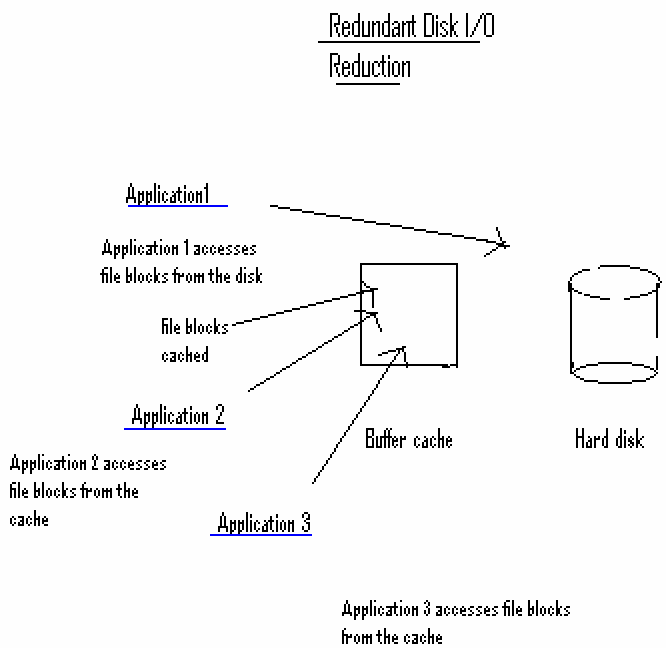
Scan scheduling criteria.

Queue - tool1--->tool2--->tool3

Each tool in the queue does the scan for a fixed time quanta and is able to complete its scan in certain no. of rounds. At the end of a time quanta, tool doing the scan saves its state ,next tool in the queue is given the opportunity to scan.In the next round,tool starts from the state it left and cycle continues till it has scanned the entire file.

File blocks read by tool1 in first round are cached,tool 2 tends to read the same file blocks when it starts ,but it fetches these block from the buffer cache instead from the disk ,thus saving expensive disk I/O operations.Same procedure is followed by tool3.

Redundant Disk I/O Reduction -



Round robin scheduling -

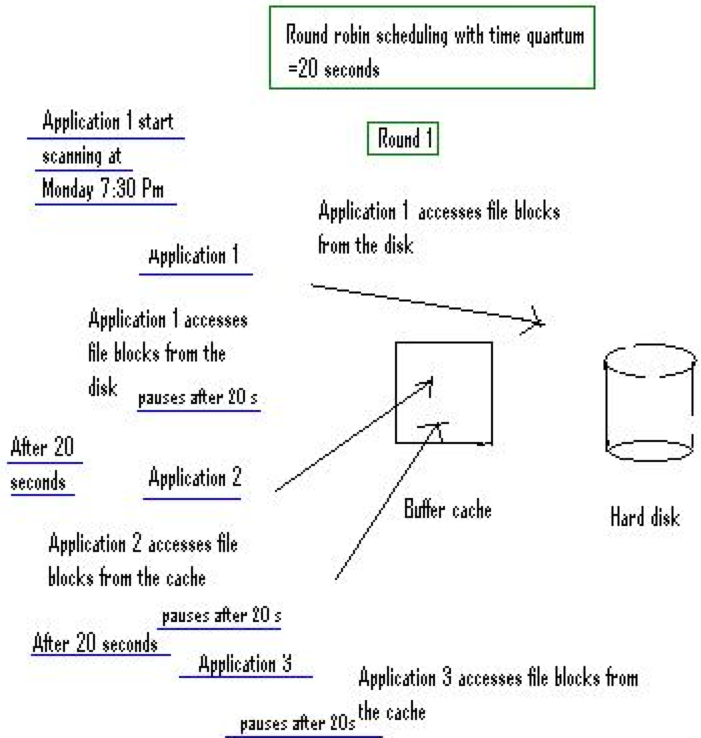


Fig1

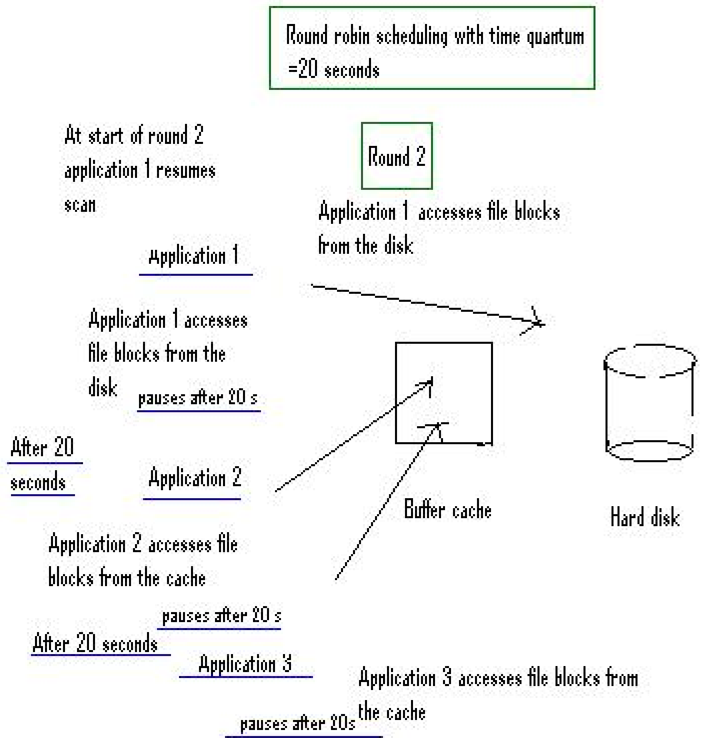


Fig 2

We implemented the schedular service and studied the impact of quantum size as the first step.

3.Reducer

3.1 Reducer Design

Architecture Description -

```
                                      ----------------
                                     Message queue1  ------->App1 Thread
                                      ----------------

                                     -----------------
                                     Message queue2  ------->App2 Thread
                                     -----------------
    Schedular Thread  ------>
                                     ------------------
                                     MessageQueue3 -------->App3 Thread
                                     ------------------

                                              |
                                              |
                                              |
                                     -------------------
                                     Message QueueN ------->AppN Thread
                                     -------------------
```

There are N application threads,each corresponding to a scanning application.
Thread poll their respective message queues continuously,looking for messages.
Schedular thread writes messages in message queue.
Schedular thread writes the appropriate function code in the message queue, application thread read it from the message queue and perform respective function.
There are 3 function codes -
100->start the scan
101->pause the scan
110->resume the paused scan

A sample run

There are 3 scanning applications.Time quanta = 20 seconds.
At the start of the scan ,1st round of round robin -
Schedular thread writes 100 in message queue 1,application 1 thread reads this message and scan by first application starts.
After 20 seconds (end of first time quantum),schedular thread writes 101 in message queue 1 and 100 in message queue 2,application threads read their respective message queues,application 1 pauses and application 2 starts the scan.
At the end of 20 seconds,(end of second time quantum),schedular thread writes 101 in message queue 2 and 100 in message queue 3,application 2 pauses its scan,application 3 starts its scan.
After 20 seconds ,first round completes.
Now 1st round of round robin has completed.
At the start of 2nd round,schedular thread writes 110 in message queue 1 and 101 in message queue 3,meaning resume the paused application 1 and and pause application 3 .After 20 seconds(when first time quantum of second round expires)schedular thread writes 101 in message queue 1 ,and 110 in message queue 2.Application 1 pauses its scan and application 2 resumes its scan.Similarly,at the end of time quantum, application 2 pauses its scan and application 3 resumes its scan.

3.2 Reducer Code -

```
/****************************************Preliminary Prototype of Scan Schedular*******************************************/
                /******************(c)Suhas Aggarwal(suhas@iitg.ernet.in)*****************/

#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<sys/ioctl.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<time.h>
#include<string.h>
#define MAX 3

/************************************************ Application buffer*********************************************************/

typedef struct {

      int App_id;
      char *path;

}App_info;


typedef struct {

      App_info *front,*tail;
      App_info *container[MAX];

}Application_queue;


Application_queue *aqt,*aqt0;


/***************************************************************************************************************************/

void set_app_queue()
{


int i=0;
char max_path[100];
```

```
aqt = (Application_queue *) malloc (sizeof(Application_queue));

while(i<MAX)

      {

      printf("Please enter the path of application program\n");
      scanf("%s",max_path);
      aqt->container[i]= (App_info *) malloc (sizeof(App_info));
      aqt->container[i]->path = (char *) malloc (strlen(max_path));
      strcpy(aqt->container[i]->path,max_path);
      aqt->container[i]->App_id=i;
      i++;

      }

     aqt->front=(App_info *) malloc (sizeof(App_info));
     aqt->front=aqt->container[0];
     aqt->tail=(App_info *) malloc (sizeof(App_info));
     aqt->tail=aqt->container[MAX-1];


}


/*****************************************************************************************************************/

void reset_app_queue()
{

int i;
App_info *temp;
temp=(App_info *) malloc (sizeof(App_info));
temp=aqt->front;
for (i=0;i<MAX-1;i++)
aqt->container[i]=aqt->container[i+1];
aqt->container[2]=temp;
aqt->front=aqt->container[0];
aqt->tail=temp;

}

/******************************************************************************************************************/


/* void storestate()
 {

 } */
```

```
/****************************************Application-schedular interaction interface****************************************/

#define PATHNAME_FTOK   "/etc/services"
#define PERMISSION          0666

typedef struct Message{

            long mtype;
            int data; // message_code

}Message;

/*********************************************Message Queue API**************************************************************/

int CreateMessageQueue(int a)
{
            int messageQ;
            key_t key;

            if((key = frock(PATHNAME_FTOK,a)) == -1)
            {
                        perform("FTOK() failed ! - exiting \n");
                        exit(-1);
            }




            if((messageQ = msgget(key, PERMISSION | IPC_CREAT)) == -1)
            {
                        perror("msgget() failed - Exiting\n");
                        exit(-1);
            }


      return messageQ;
}

/**************************************************************************************************************************/

SendMessage (int messageQ, Message buf)
{
            if((msgsnd(messageQ, &buf, sizeof(Message), 0)) == -1)
            {
```

```
                        perror("Message send failed\n");
            }
}

/******************************************************************************************************************/

Message *ReceiveMessage (int messageQ)
{
            Message *buf = (Message *)malloc(sizeof (Message));


      if((msgrcv(messageQ, buf, sizeof(Message), 0, 0)) == -1)
            {
                        perror("Receive failed");
                        free(buf);
                        buf = NULL;
            }

            return buf;
}

/******************************************************************************************************************/


DestroyMessageQueue(int messageQ)
{

            if((msgctl(messageQ, IPC_RMID, NULL)) == -1)
            {
               perror("Could not destroy Message Queue\n");
            }

}

/******************************************************************************************************************/


int messageQueue1,messageQueue2,messageQueue3,messageQueue4;


/******************************************************************************************************************/


int round = 0;


/*********************************************Application threads*****************************************************/

void *application1_thread(void *arg)
```

```
{
        Message *m;

    while(1)
        {

    if((m = ReceiveMessage(messageQueue1)) != NULL  )
        {

    if(m->data == 100)
    system("/home/suhas/Desktop/Anti_Virus/avast4workstation-1.0.8/bin/./avast /home/suhas/ >>flag10");


    if(m->data == 101)
    system("pkill -SIGSTOP avast");


    if(m->data == 110)
    system ("pkill -SIGCONT avast");

    free(m);

    }

    }

        return;

}

/*****************************************************************************************************************************/



void *application2_thread(void *arg)
{

        Message *m;

        while(1)
        {

    if((m = ReceiveMessage(messageQueue2)) != NULL )
    {
```

```
if(m->data == 100)
system("/home/suhas/Desktop/Anti_Virus/vascan-1.3.4-4.3.23-Linux-i386/./vascan /home/suhas/ >>flag00");


if(m->data == 101)
system("pkill -SIGSTOP vascan");


if(m->data == 110)
system("pkill -SIGCONT vascan");



free(m);

}

}

return;


}

/*******************************************************************************************************************************/


void *application3_thread(void *arg)
{

        Message *m;

        while(1)
        {

    if((m = ReceiveMessage(messageQueue3)) != NULL )
        {

    if(m->data == 100)
    system("clamscan -r /home/suhas/ >>flag20");


    if(m->data == 101)
    system("pkill -SIGSTOP clamscan");


    If(m->data == 110)
```

```
    system("pkill -SIGCONT clamscan");


    free(m);

    }

        }
        return;

}

/*************************************************************************************************************************


void *application4_thread(void *arg)
{

        Message *m;

    while(1)
        {

    if((m = ReceiveMessage(messageQueue4)) != NULL )
        printf("Application4 %d\n",m->data);
    free(m);

    }

    return;

}


/*************************************************Schedular Thread*******************************************************/


  void *schedular_thread(void *arg)

 {

/*************************************************Round Robin Scheduling**********************************************/

    int a,b,choice,time_quanta=20000000,counter=0; //time quanta is specified in microseconds
    Message m;

    m.mtype = 1;

    printf("***************************** Select a scheduling algorithm ***********************************\n\n");
```

```
printf("1)FIFO\n");
printf("2)Round Robin\n\n");



scanf("%d",&choice);

if(choice == 1)
{

usleep(60000000);

m.data = 100;
SendMessage(messageQueue1,m);
usleep(600000000);

m.data = 101;
SendMessage(messageQueue1,m);
usleep(60000000);

m.data = 100;
SendMessage(messageQueue2,m);
usleep(600000000);

m.data = 101;
SendMessage(messageQueue2,m);
usleep(60000000);

m.data = 100;
SendMessage(messageQueue3,m);
usleep(600000000);

m.data = 101;
SendMessage(messageQueue3,m);
usleep(60000000);

m.data = 100;
SendMessage(messageQueue4,m);
usleep(600000000);

m.data = 101;
SendMessage(messageQueue4,m);
usleep(60000000);

}


if(choice == 2)
{
```

```
while(1)
{

usleep(time_quanta);            // time quanta of an application expires

printf("Round - %d\n",round);


if(round == 0)
{

if(counter == 0)
{
m.data = 100;
SendMessage(messageQueue1, m);
usleep(1000000);

}

if(counter == 1)
{
m.data = 101;
SendMessage(messageQueue1, m);
usleep(1000000);
m.data = 100;
SendMessage(messageQueue2, m);
}

if(counter == 2)
{
m.data = 101;
SendMessage(messageQueue2, m);
usleep(1000000);
m.data = 100;
SendMessage(messageQueue3, m);
}




}


else
{
```

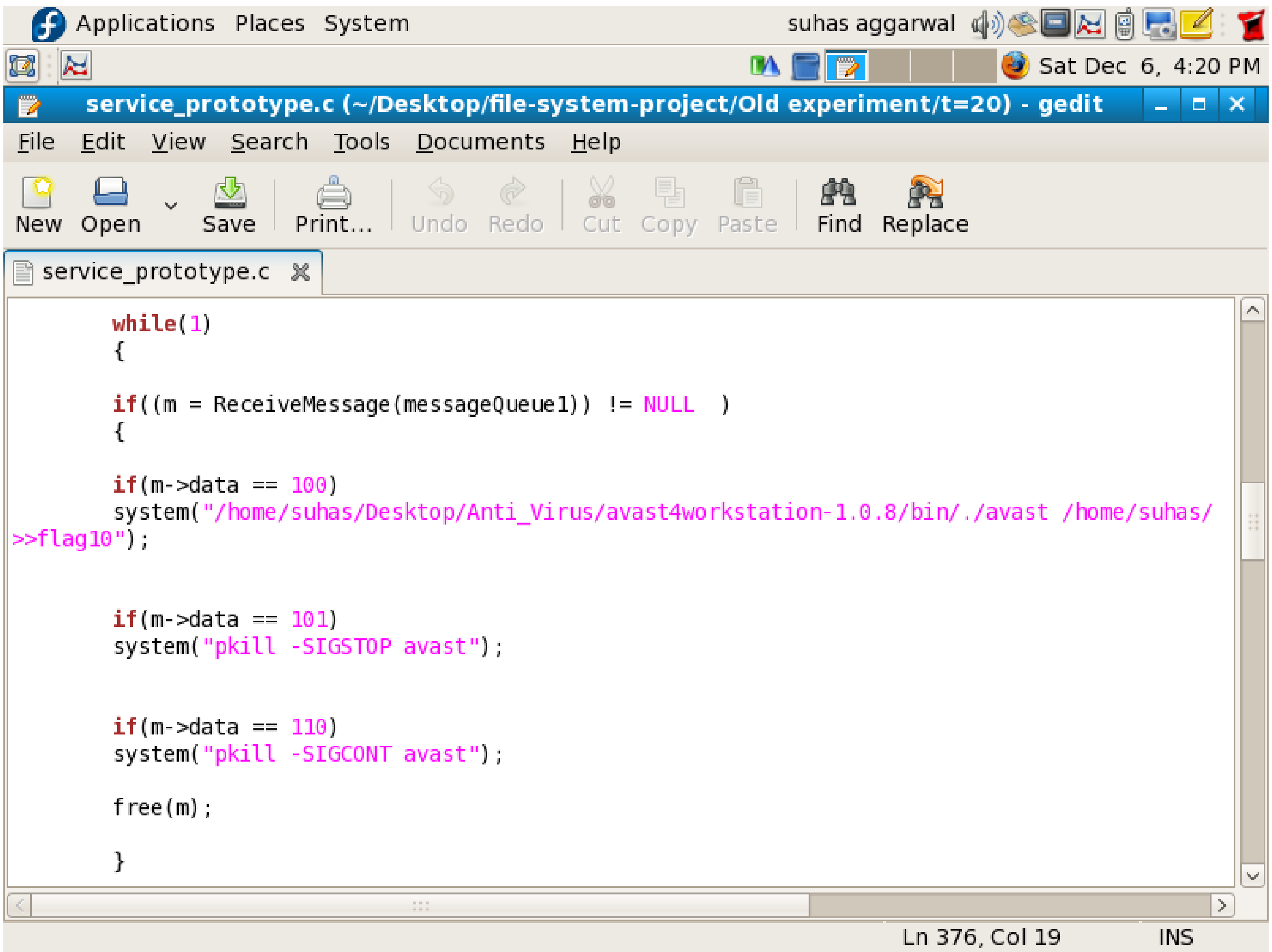
```
a=aqt->tail->App_id;
m.data=101;



if(a==0)
SendMessage(messageQueue1, m);


else if(a==1)
SendMessage(messageQueue2, m);



else if(a==2)
SendMessage(messageQueue3, m);






b=aqt->front->App_id;
m.data=110;

usleep(1000000);

if(b==0)
SendMessage(messageQueue1, m);


else if(b==1)
SendMessage(messageQueue2, m);


else if(b==2)
SendMessage(messageQueue3, m);




}

if(counter == MAX-1)
 {
 round++;                    // Increment to next round
 counter=-1;
 }
```

```
     counter++;
     reset_app_queue();

    }

    }

/************************************************************************************************************************/

    return;
}



/****************************************************** main **************************************************************/


main(int argc, char **argv)
{


/****************************************************** The Thread Ids****************************************************/


      pthread_t application_1;
     pthread_t application_2;
     pthread_t application_3;
     pthread_t schedular;


/************************************************* Initialise application buffer *******************************************/


     set_app_queue();


/************************************************* Store buffer's state at t=0 ******************************************/


          aqt0 = (Application_queue *) malloc (sizeof(Application_queue));


/************************************************************************************************************************


    storestate();
```

```
*********************************************** Initialise message buffers ************************************************/

    messageQueue1 = CreateMessageQueue(1);
    messageQueue2 = CreateMessageQueue(2);
    messageQueue3 = CreateMessageQueue(3);
   messageQueue4 = CreateMessageQueue(4);

/************************************************ Start the Application threads ********************************************/

        if((pthread_create(&application_1, NULL, application1_thread, NULL )) != 0)
        {
                printf("Error creating application_1 thread - Exiting\n");
                exit(-1);
        }

        if((pthread_create(&application_2, NULL, application2_thread, NULL )) != 0)
        {
                printf("Error creating application_2 thread - Exiting\n");
                exit(-1);
        }

        if((pthread_create(&application_3, NULL, application3_thread, NULL )) != 0)
        {
                printf("Error creating application_3 thread - Exiting\n");
                exit(-1);
        }

    if((pthread_create(&schedular, NULL, schedular_thread, NULL )) != 0)
        {
                printf("Error creating schedular thread - Exiting\n");
                exit(-1);
        }

/****************************************************** Wait for the threads to exit*****************************************/

     pthread_join(application_1);
     pthread_join(application_2);
    pthread_join(application_3);
    pthread_join(schedular);

/********************************************************* Cleanup code ***************************************************/

     DestroyMessageQueue(messageQueue1);
```

```
    DestroyMessageQueue (messageQueue2);
    DestroyMessageQueue(messageQueue3);



    return 0;

}
/*****************************************************************************************************************************/
```

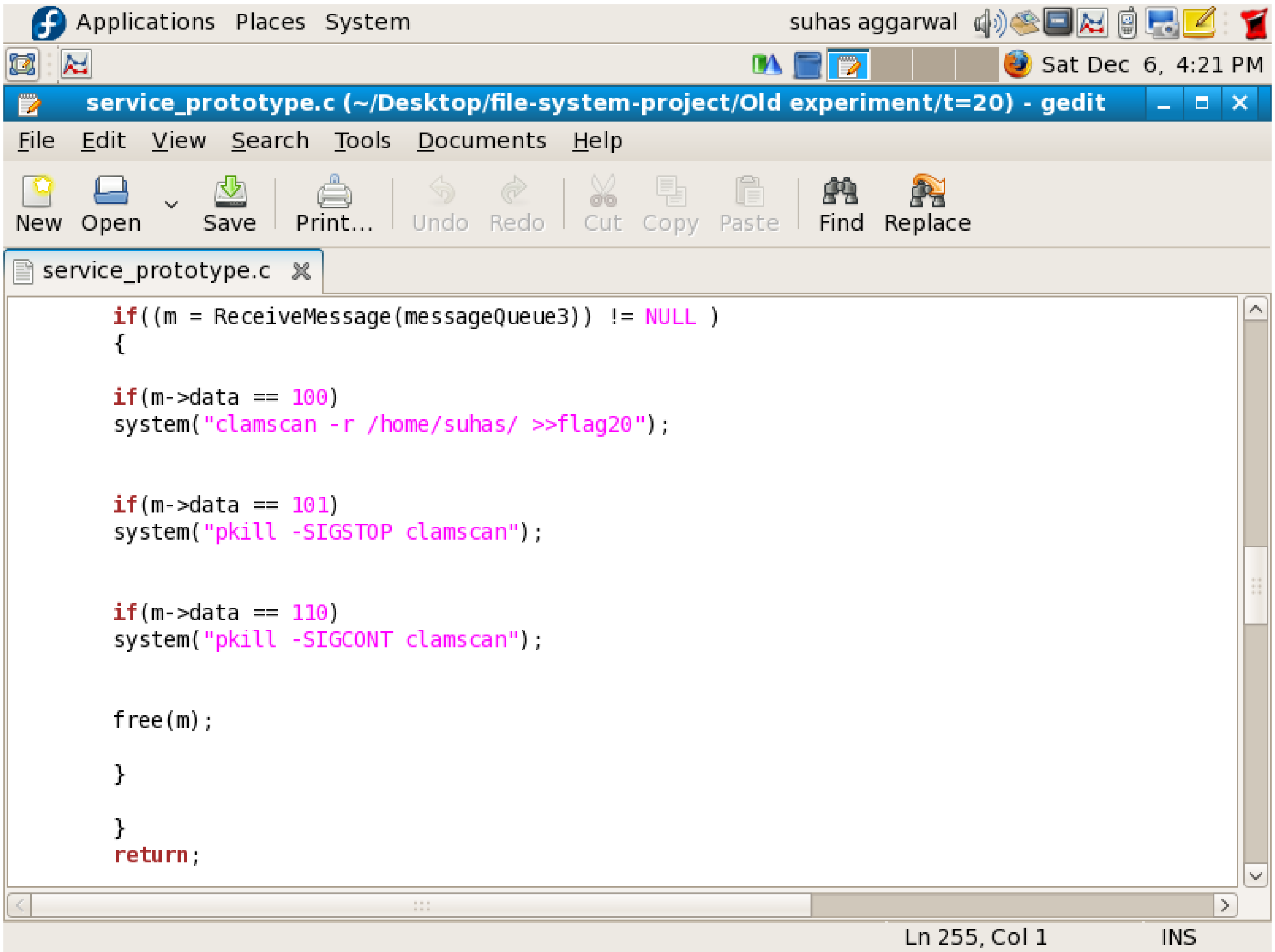
3.3 Interfacing Applications with Reducer:



Avast virus scanner interfacing

```c
if((m = ReceiveMessage(messageQueue2)) != NULL )
{


if(m->data == 100)
system("/home/suhas/Desktop/Anti_Virus/vascan-1.3.4-4.3.23-Linux-i386/./vascan /home/
suhas/ >>flag00");


if(m->data == 101)
system("pkill -SIGSTOP vascan");


if(m->data == 110)
system("pkill -SIGCONT vascan");



free(m);

}
```

Vascan virus scanner interfacing



CLAMAV Virus scanner Interfacing

Above figures ,illustrate interfacing applications with our schedular code.
In the above figures, three Disk I/O intensive applications / 3 virus scanners, avast,vascan and clamAV have been interfaced with our schedular.
Schedular has a very simple ,user friendly, API , NO APPLICATON/SCANNER CODE EDITING is required to interface it with our schedular.
System command is used to give command to the applications from within the code to initiate.
pkill along with appropriate code SIGSTOP or SIGCONT is used to pause or resume respective scanning applications.

4. SCHEDULAR Experiment

4.1   Studying Disk I/O Activity when Disk I/O intensive applications work individually (as when during a normal periodic scan)

1)
Aim-
To study the disk I/O activity of Disk I/O intensive applications (virus scanners,anti spywares, backup programs) when they work individually (as when during a normal periodic scan)

For simplicity, we are studying the disk I/O caused by scanning applications of the same type – virus scanners at first.We consider three virus scanners and study Disk I/O activity when each of them work individually (as they do during normal periodic scan).
We will also study the disk I/O redundancy three scanners induce ,and reduction in disk I/O redundancy when three scanners are scheduled using a scheduling algorithm,particularly round robin scheme,which we have chosen.

2)
System set up –
Operating system – Fedora core 6

Filesystem - ext3
RAM- 512 MB
Processor-Pentium 4 Processor ,3 GHz
Disk Space occupied – 2.25 GB

3)
Scanners used -
1)Avast – avast4workstation-1.0.8

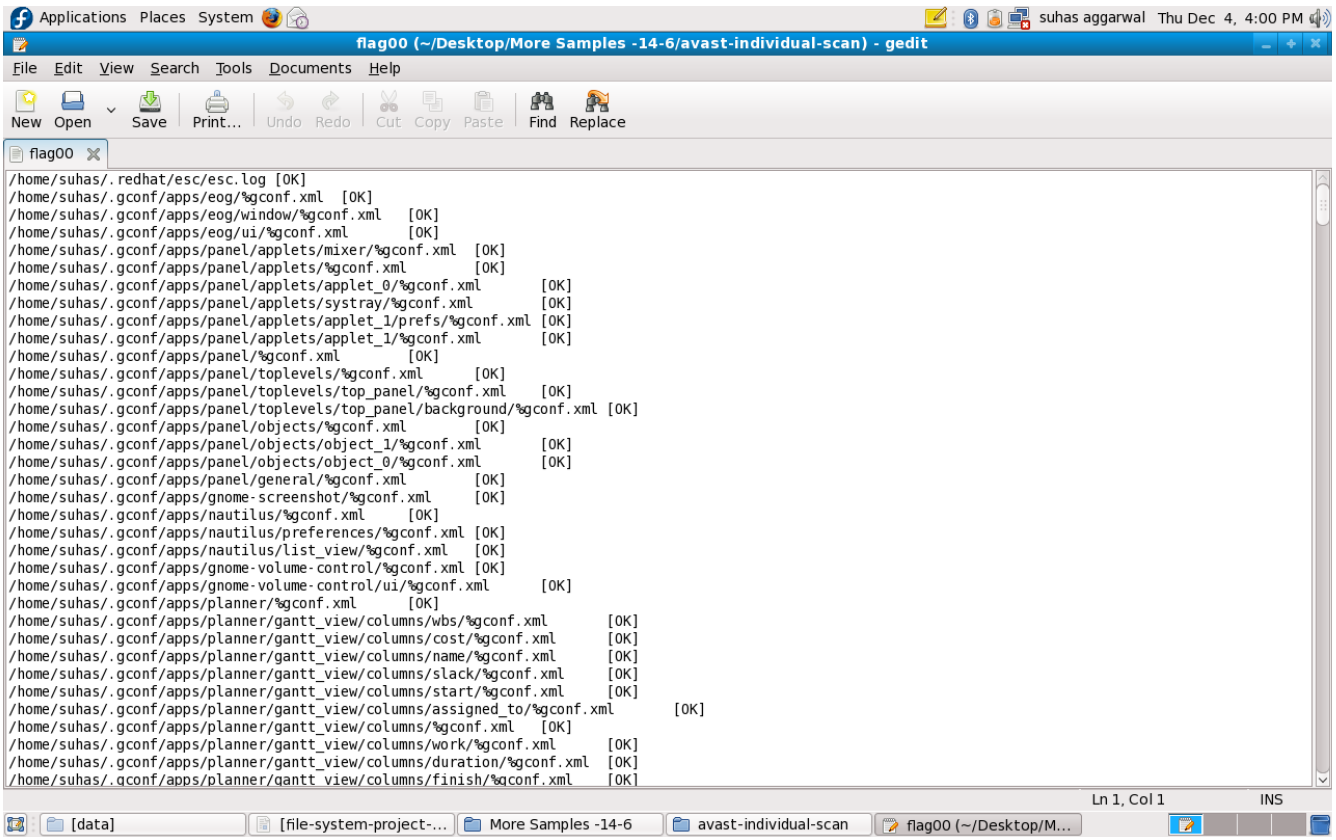
2)Vascan – vascan-1.3.4-4.3.23-Linux-i386

3)ClamAV- Engine version-0.91.2

Tool used for monitoring Disk I/O activity -
iostat
Command used – iostat –x 2 >>filename
Individual virus scans are executed and iostat is used to monitor disk i/o activity during scans.

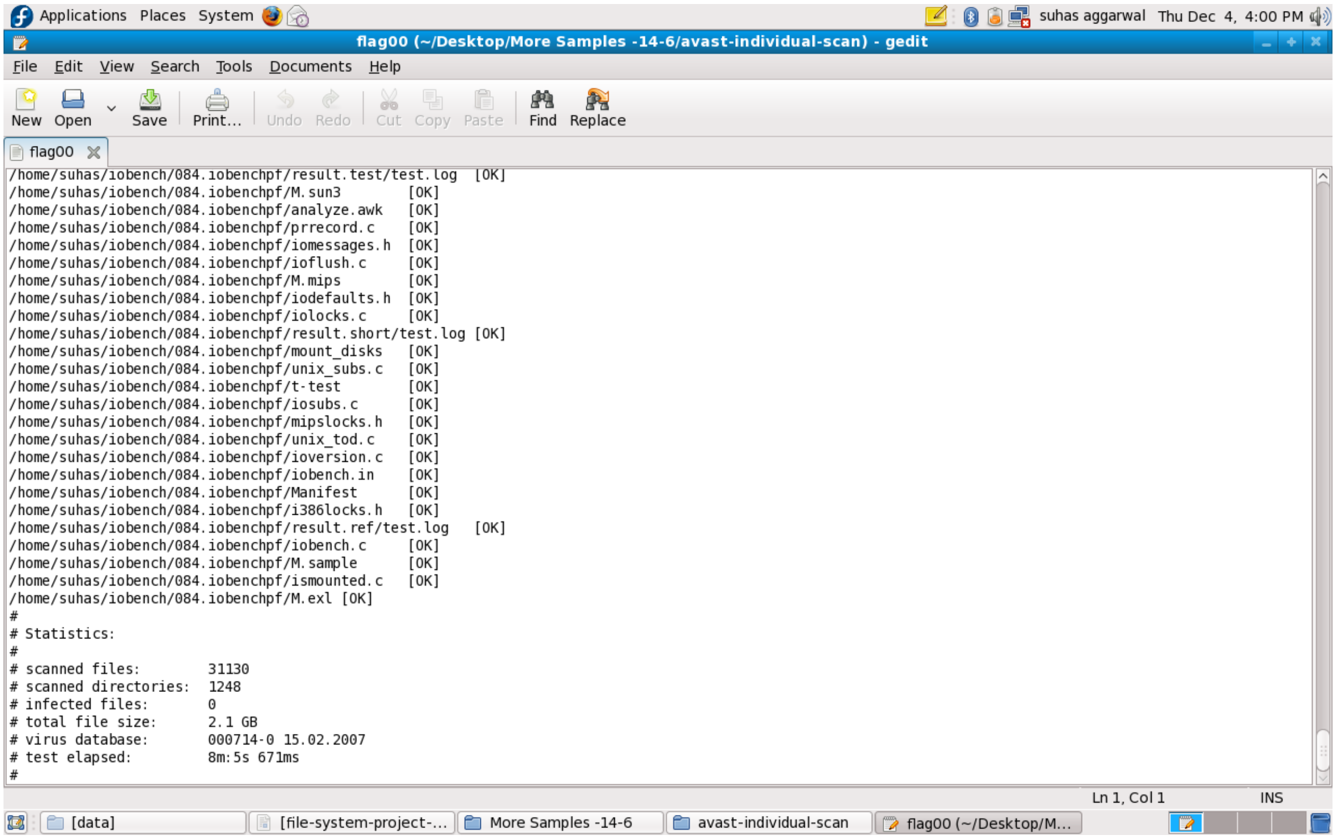
4.1.1. File scanning logs of individual virus scans -

Avast virus scanner

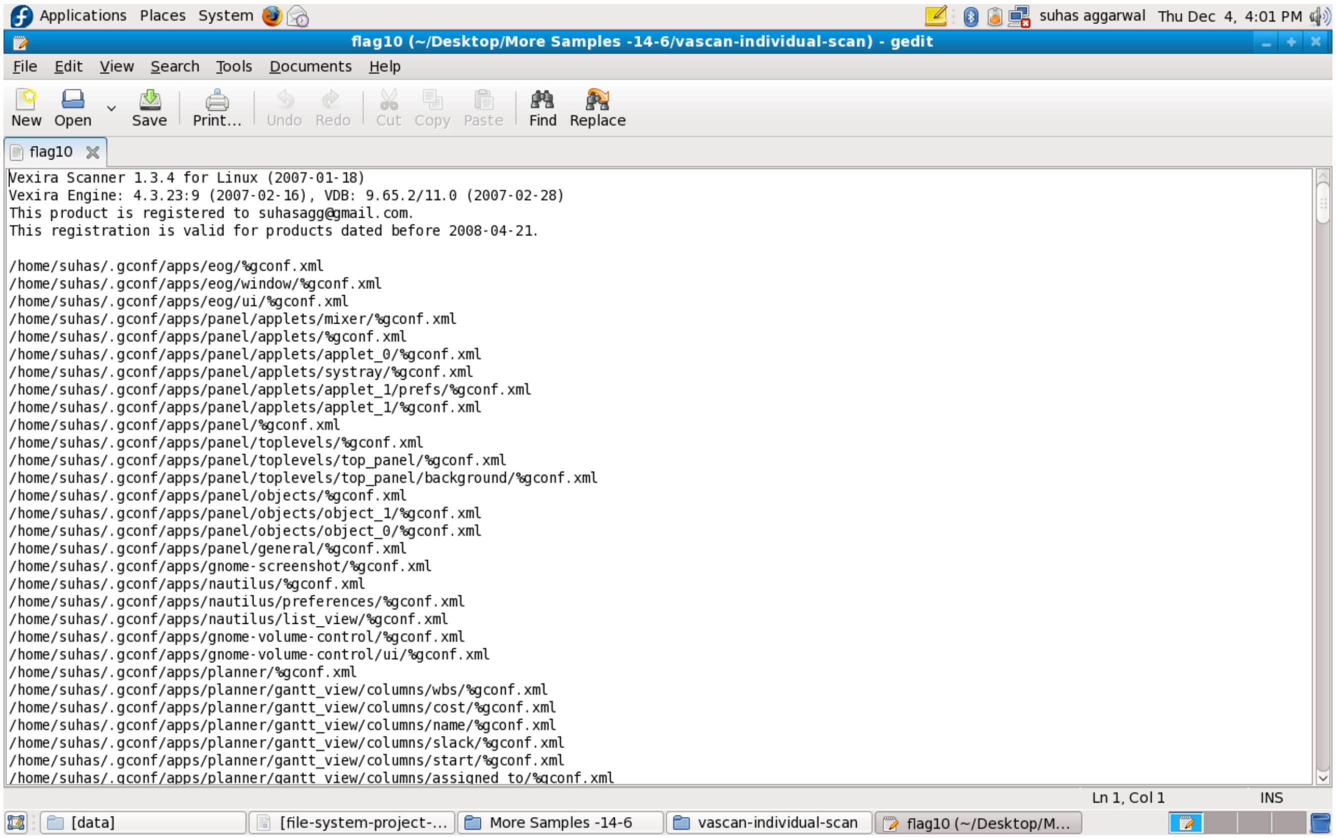


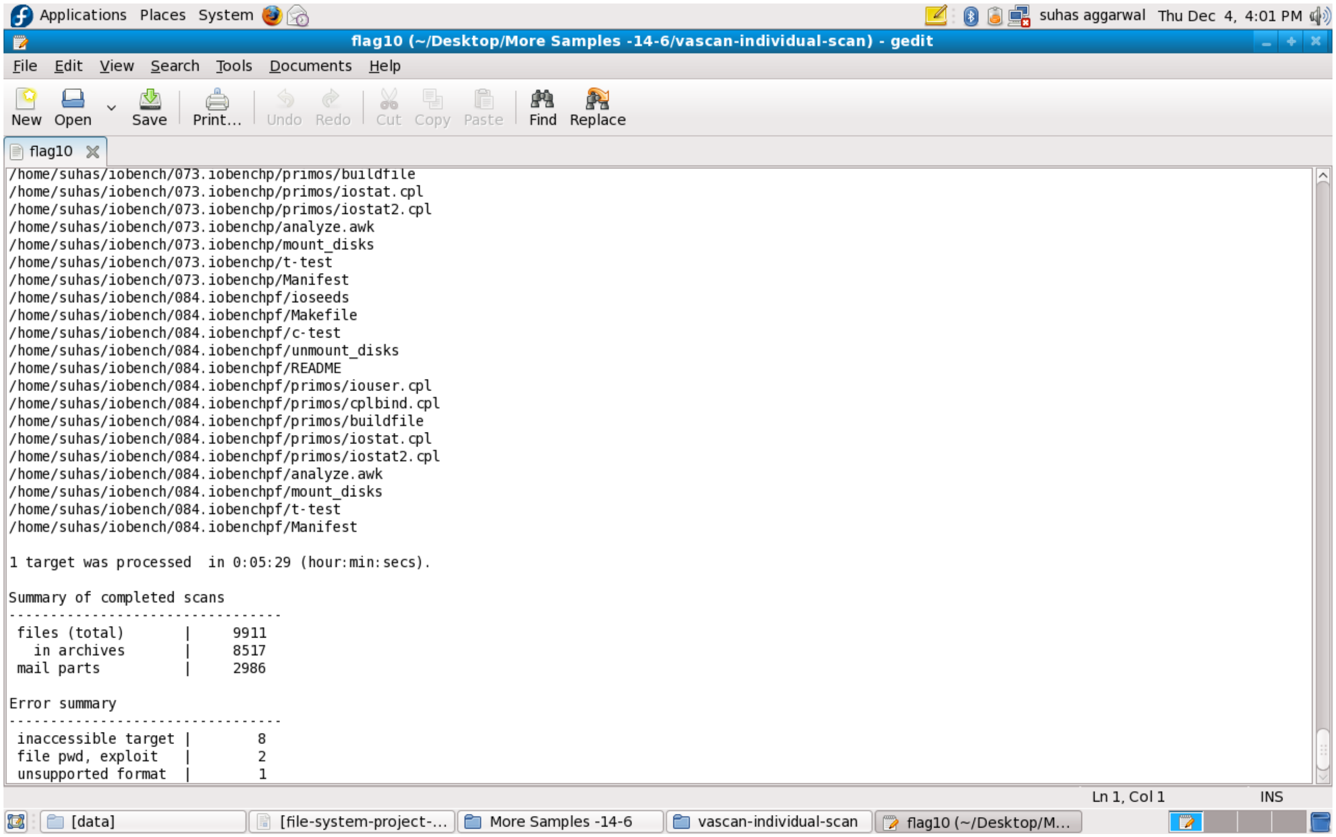
```
/home/suhas/.redhat/esc/esc.log [OK]
/home/suhas/.gconf/apps/eog/%gconf.xml  [OK]
/home/suhas/.gconf/apps/eog/window/%gconf.xml   [OK]
/home/suhas/.gconf/apps/eog/ui/%gconf.xml       [OK]
/home/suhas/.gconf/apps/panel/applets/mixer/%gconf.xml  [OK]
/home/suhas/.gconf/apps/panel/applets/%gconf.xml        [OK]
/home/suhas/.gconf/apps/panel/applets/applet_0/%gconf.xml       [OK]
/home/suhas/.gconf/apps/panel/applets/systray/%gconf.xml        [OK]
/home/suhas/.gconf/apps/panel/applets/applet_1/prefs/%gconf.xml [OK]
/home/suhas/.gconf/apps/panel/applets/applet_1/%gconf.xml       [OK]
/home/suhas/.gconf/apps/panel/%gconf.xml        [OK]
/home/suhas/.gconf/apps/panel/toplevels/%gconf.xml      [OK]
/home/suhas/.gconf/apps/panel/toplevels/top_panel/%gconf.xml    [OK]
/home/suhas/.gconf/apps/panel/toplevels/top_panel/background/%gconf.xml [OK]
/home/suhas/.gconf/apps/panel/objects/%gconf.xml        [OK]
/home/suhas/.gconf/apps/panel/objects/object_1/%gconf.xml       [OK]
/home/suhas/.gconf/apps/panel/objects/object_0/%gconf.xml       [OK]
/home/suhas/.gconf/apps/panel/general/%gconf.xml        [OK]
/home/suhas/.gconf/apps/gnome-screenshot/%gconf.xml     [OK]
/home/suhas/.gconf/apps/nautilus/%gconf.xml     [OK]
/home/suhas/.gconf/apps/nautilus/preferences/%gconf.xml [OK]
/home/suhas/.gconf/apps/nautilus/list_view/%gconf.xml   [OK]
/home/suhas/.gconf/apps/gnome-volume-control/%gconf.xml [OK]
/home/suhas/.gconf/apps/gnome-volume-control/ui/%gconf.xml      [OK]
/home/suhas/.gconf/apps/planner/%gconf.xml      [OK]
/home/suhas/.gconf/apps/planner/gantt_view/columns/wbs/%gconf.xml       [OK]
/home/suhas/.gconf/apps/planner/gantt_view/columns/cost/%gconf.xml      [OK]
/home/suhas/.gconf/apps/planner/gantt_view/columns/name/%gconf.xml      [OK]
/home/suhas/.gconf/apps/planner/gantt_view/columns/slack/%gconf.xml     [OK]
/home/suhas/.gconf/apps/planner/gantt_view/columns/start/%gconf.xml     [OK]
/home/suhas/.gconf/apps/planner/gantt_view/columns/assigned_to/%gconf.xml       [OK]
/home/suhas/.gconf/apps/planner/gantt_view/columns/%gconf.xml   [OK]
/home/suhas/.gconf/apps/planner/gantt_view/columns/work/%gconf.xml      [OK]
/home/suhas/.gconf/apps/planner/gantt_view/columns/duration/%gconf.xml  [OK]
/home/suhas/.gconf/apps/planner/gantt_view/columns/finish/%gconf.xml    [OK]
```

Note: For complete log, please refer to data samples .

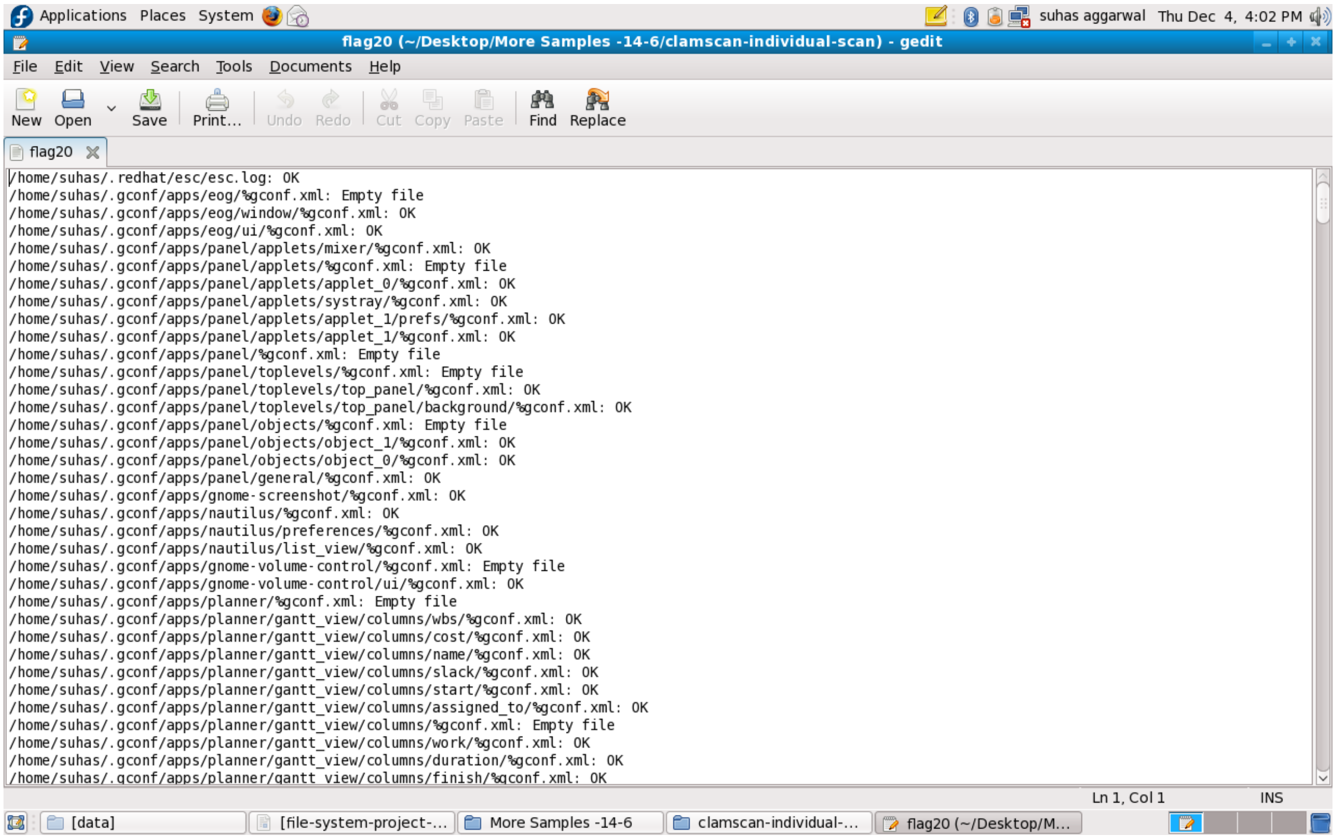

Applications Places System
suhas aggarwal Thu Dec 4, 4:00 PM
flag00 (~/Desktop/More Samples -14-6/avast-individual-scan) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
flag00
/home/suhas/iobench/084.iobenchpf/result.test/test.log [OK]
/home/suhas/iobench/084.iobenchpf/M.sun3 [OK]
/home/suhas/iobench/084.iobenchpf/analyze.awk [OK]
/home/suhas/iobench/084.iobenchpf/prrecord.c [OK]
/home/suhas/iobench/084.iobenchpf/iomessages.h [OK]
/home/suhas/iobench/084.iobenchpf/ioflush.c [OK]
/home/suhas/iobench/084.iobenchpf/M.mips [OK]
/home/suhas/iobench/084.iobenchpf/iodefaults.h [OK]
/home/suhas/iobench/084.iobenchpf/iolocks.c [OK]
/home/suhas/iobench/084.iobenchpf/result.short/test.log [OK]
/home/suhas/iobench/084.iobenchpf/mount_disks [OK]
/home/suhas/iobench/084.iobenchpf/unix_subs.c [OK]
/home/suhas/iobench/084.iobenchpf/t-test [OK]
/home/suhas/iobench/084.iobenchpf/iosubs.c [OK]
/home/suhas/iobench/084.iobenchpf/mipslocks.h [OK]
/home/suhas/iobench/084.iobenchpf/unix_tod.c [OK]
/home/suhas/iobench/084.iobenchpf/ioversion.c [OK]
/home/suhas/iobench/084.iobenchpf/iobench.in [OK]
/home/suhas/iobench/084.iobenchpf/Manifest [OK]
/home/suhas/iobench/084.iobenchpf/i386locks.h [OK]
/home/suhas/iobench/084.iobenchpf/result.ref/test.log [OK]
/home/suhas/iobench/084.iobenchpf/iobench.c [OK]
/home/suhas/iobench/084.iobenchpf/M.sample [OK]
/home/suhas/iobench/084.iobenchpf/ismounted.c [OK]
/home/suhas/iobench/084.iobenchpf/M.exl [OK]
#
Statistics:
#
scanned files: 31130
scanned directories: 1248
infected files: 0
total file size: 2.1 GB
virus database: 000714-0 15.02.2007
test elapsed: 8m:5s 671ms
#
Ln 1, Col 1 INS
[data] [file-system-project-... More Samples -14-6 avast-individual-scan flag00 (~/Desktop/M...

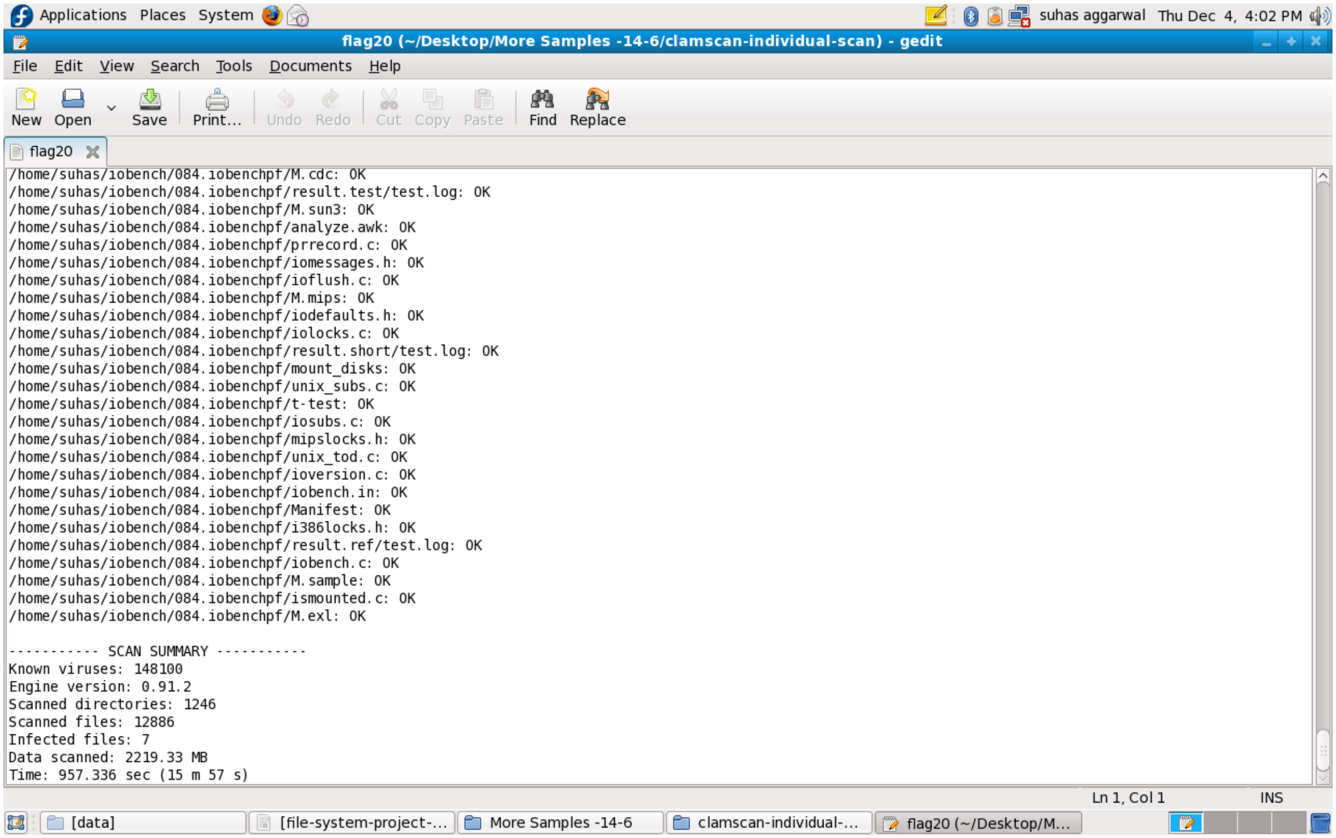
Vascan virus scanner




Applications Places System
suhas aggarwal Thu Dec 4, 4:01 PM
flag10 (~/Desktop/More Samples -14-6/vascan-individual-scan) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
flag10
/home/suhas/iobench/073.iobenchp/primos/buildfile
/home/suhas/iobench/073.iobenchp/primos/iostat.cpl
/home/suhas/iobench/073.iobenchp/primos/iostat2.cpl
/home/suhas/iobench/073.iobenchp/analyze.awk
/home/suhas/iobench/073.iobenchp/mount_disks
/home/suhas/iobench/073.iobenchp/t-test
/home/suhas/iobench/073.iobenchp/Manifest
/home/suhas/iobench/084.iobenchpf/ioseeds
/home/suhas/iobench/084.iobenchpf/Makefile
/home/suhas/iobench/084.iobenchpf/c-test
/home/suhas/iobench/084.iobenchpf/unmount_disks
/home/suhas/iobench/084.iobenchpf/README
/home/suhas/iobench/084.iobenchpf/primos/iouser.cpl
/home/suhas/iobench/084.iobenchpf/primos/cplbind.cpl
/home/suhas/iobench/084.iobenchpf/primos/buildfile
/home/suhas/iobench/084.iobenchpf/primos/iostat.cpl
/home/suhas/iobench/084.iobenchpf/primos/iostat2.cpl
/home/suhas/iobench/084.iobenchpf/analyze.awk
/home/suhas/iobench/084.iobenchpf/mount_disks
/home/suhas/iobench/084.iobenchpf/t-test
/home/suhas/iobench/084.iobenchpf/Manifest
1 target was processed in 0:05:29 (hour:min:secs).
Summary of completed scans
----
files (total) | 9911
in archives | 8517
mail parts | 2986
Error summary
----
inaccessible target | 8
file pwd, exploit | 2
unsupported format | 1
Ln 1, Col 1 INS
[data] [file-system-project-... More Samples -14-6 vascan-individual-scan flag10 (~/Desktop/M...

CLAMAV virus scanner



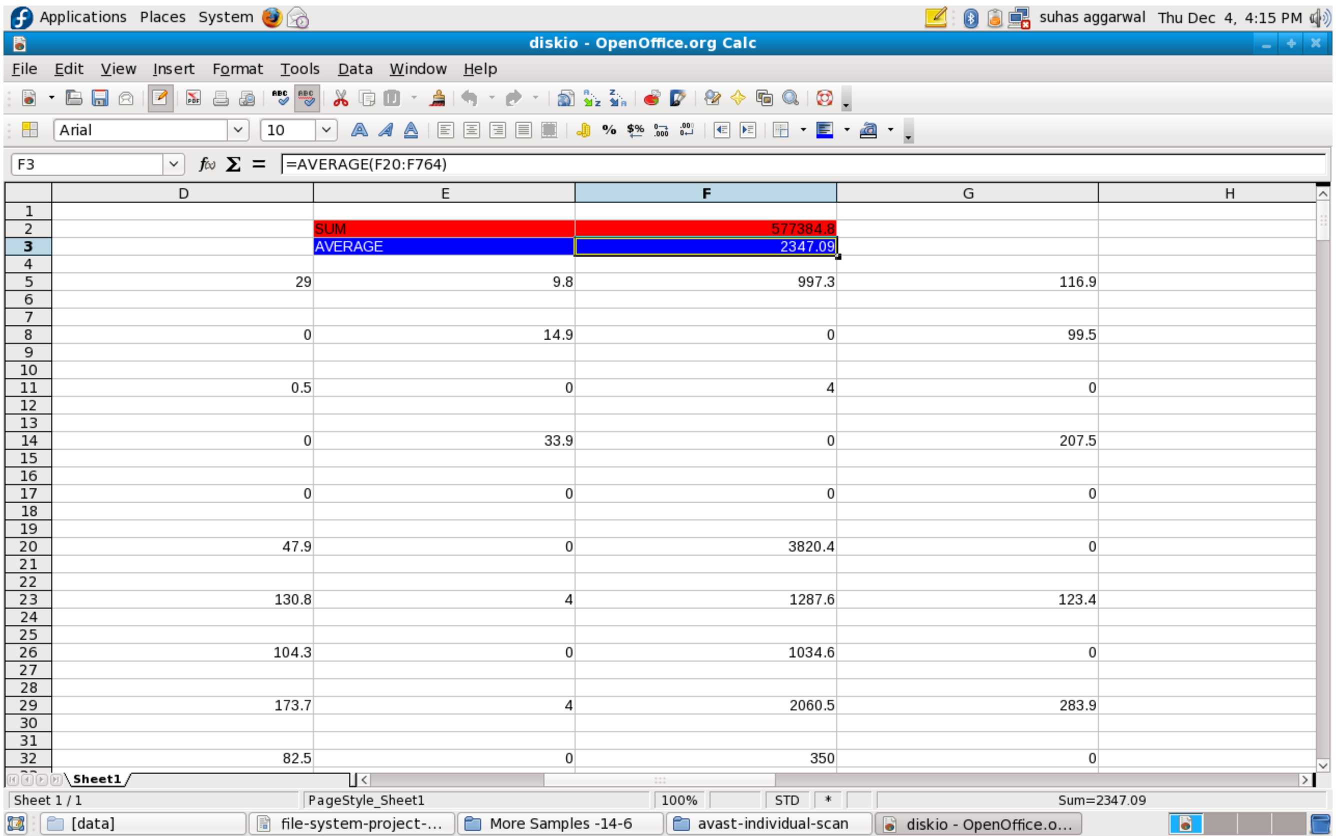


On observing file scanning log of three virus scanner we derive that ,3 virus scanners employ **same file scanning algorithm.**
2.25 GB system is scanned 3 times by 3 virus scanners.

4.1.2 AVAST virus scan analysis

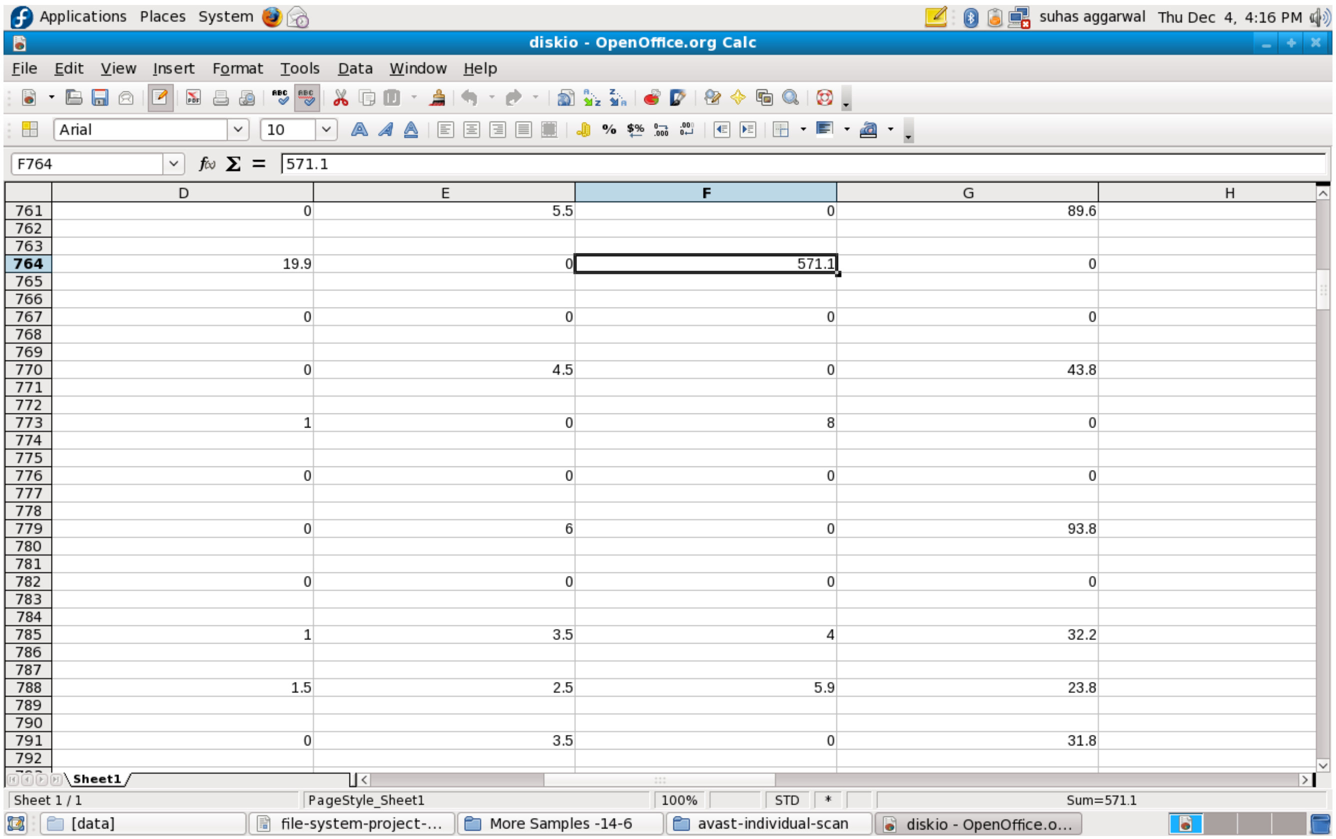
Disk I/O report generated by iostat for Avast virus scanner -

Avast virus scanner



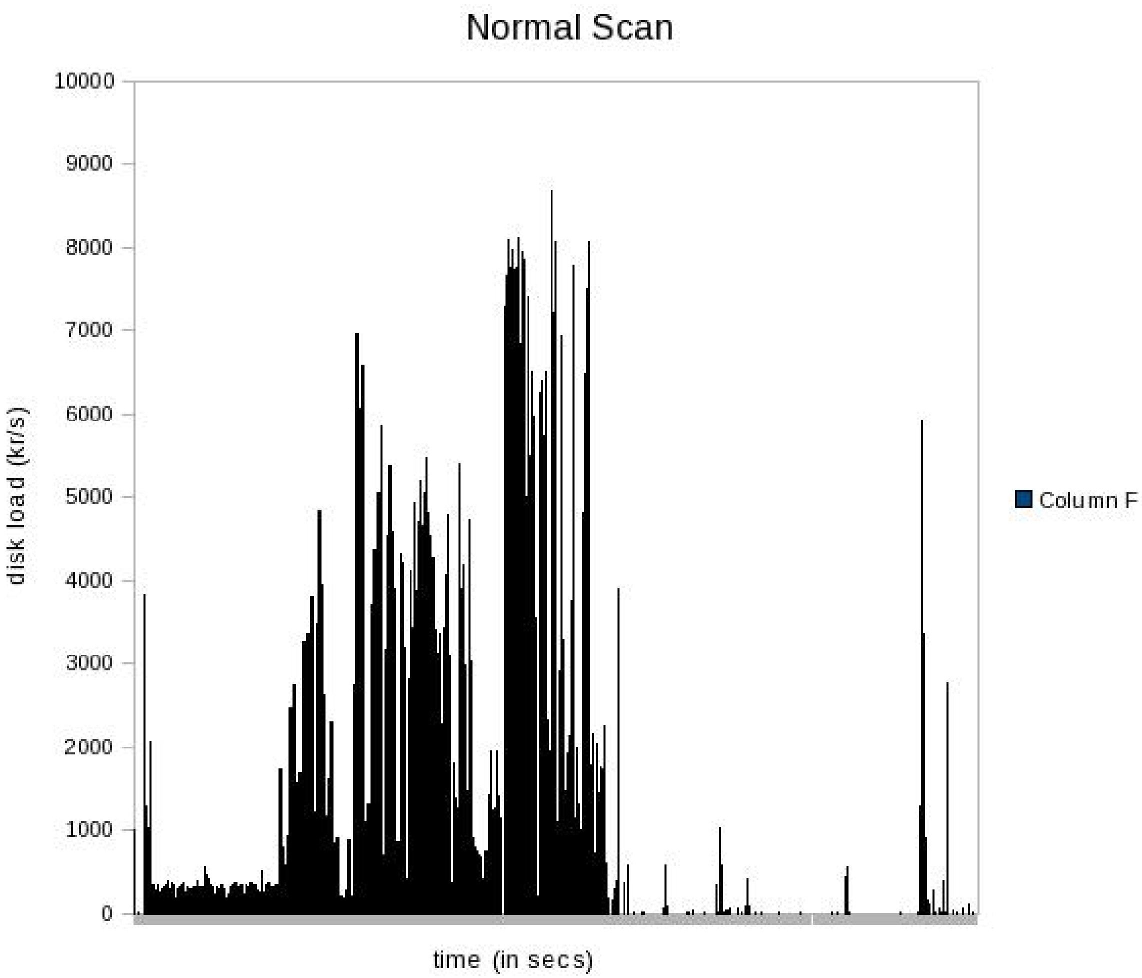
	D	E	F	G	H
2		SUM	577384.8		
3		AVERAGE	2347.09		
5	29	9.8	997.3	116.9	
8	0	14.9	0	99.5	
11	0.5	0	4	0	
14	0	33.9	0	207.5	
17	0	0	0	0	
20	47.9	0	3820.4	0	
23	130.8	4	1287.6	123.4	
26	104.3	0	1034.6	0	
29	173.7	4	2060.5	283.9	
32	82.5	0	350	0	

Note: For complete scan sheets please refer to Data samples.



Net Disk I/O caused by avast virus scanner during the scan -
577384.8*2 =1154769.6KB

Scan plot -

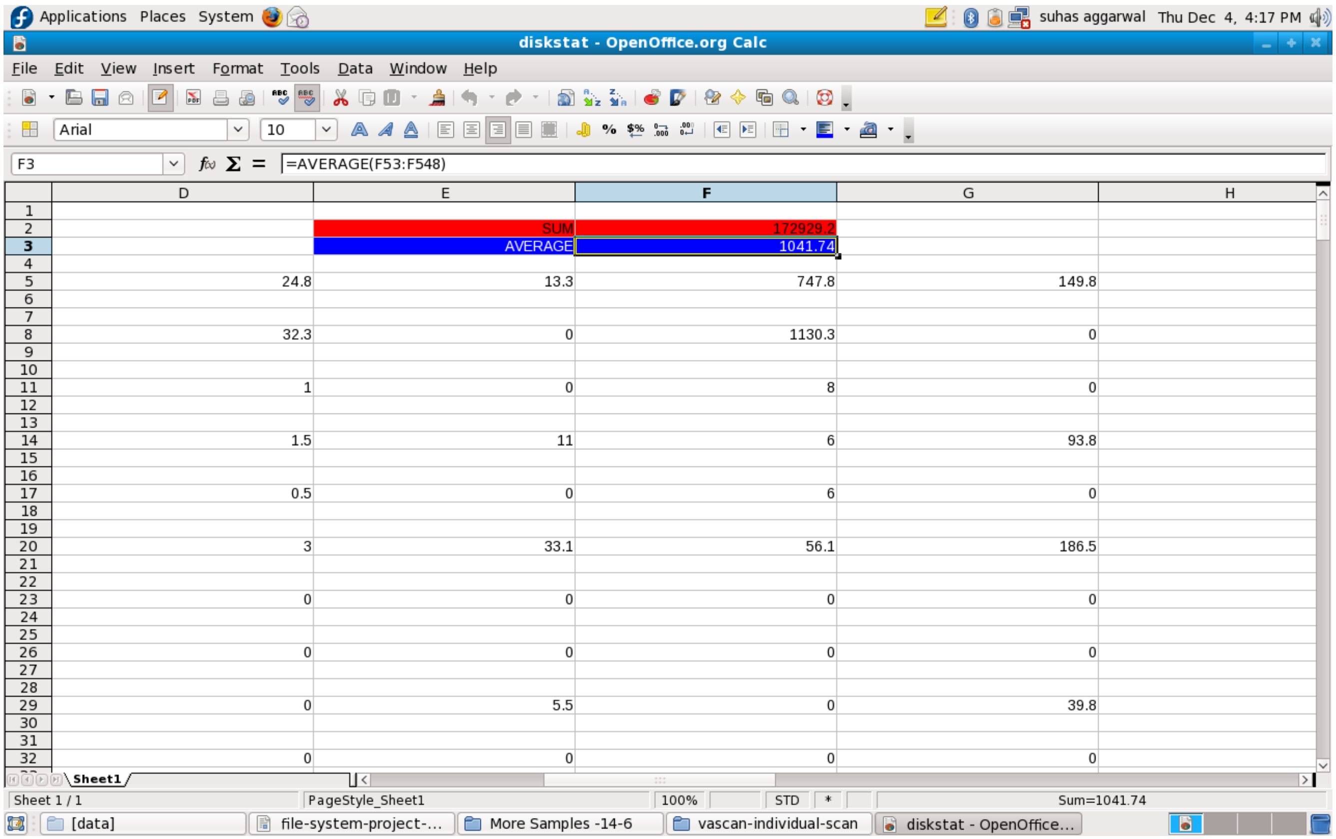


This scan plot depicts disk read done by avast virus scanner during the scan (plot of kr/s field in iostat output)

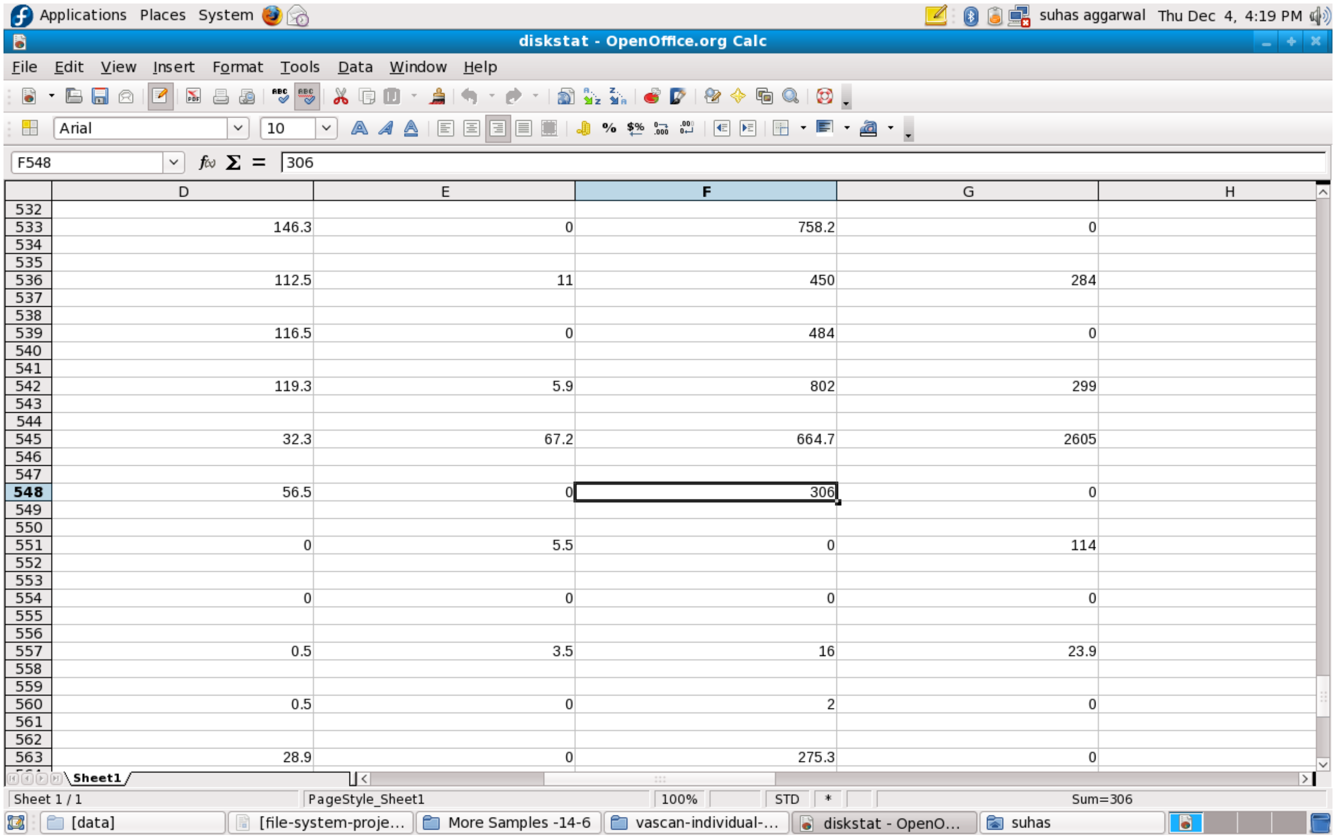
4.1.3. VASCAN Virus scan analysis -

Disk I/O report generated by iostat for vascan virus scanner-

Vascan virus scanner



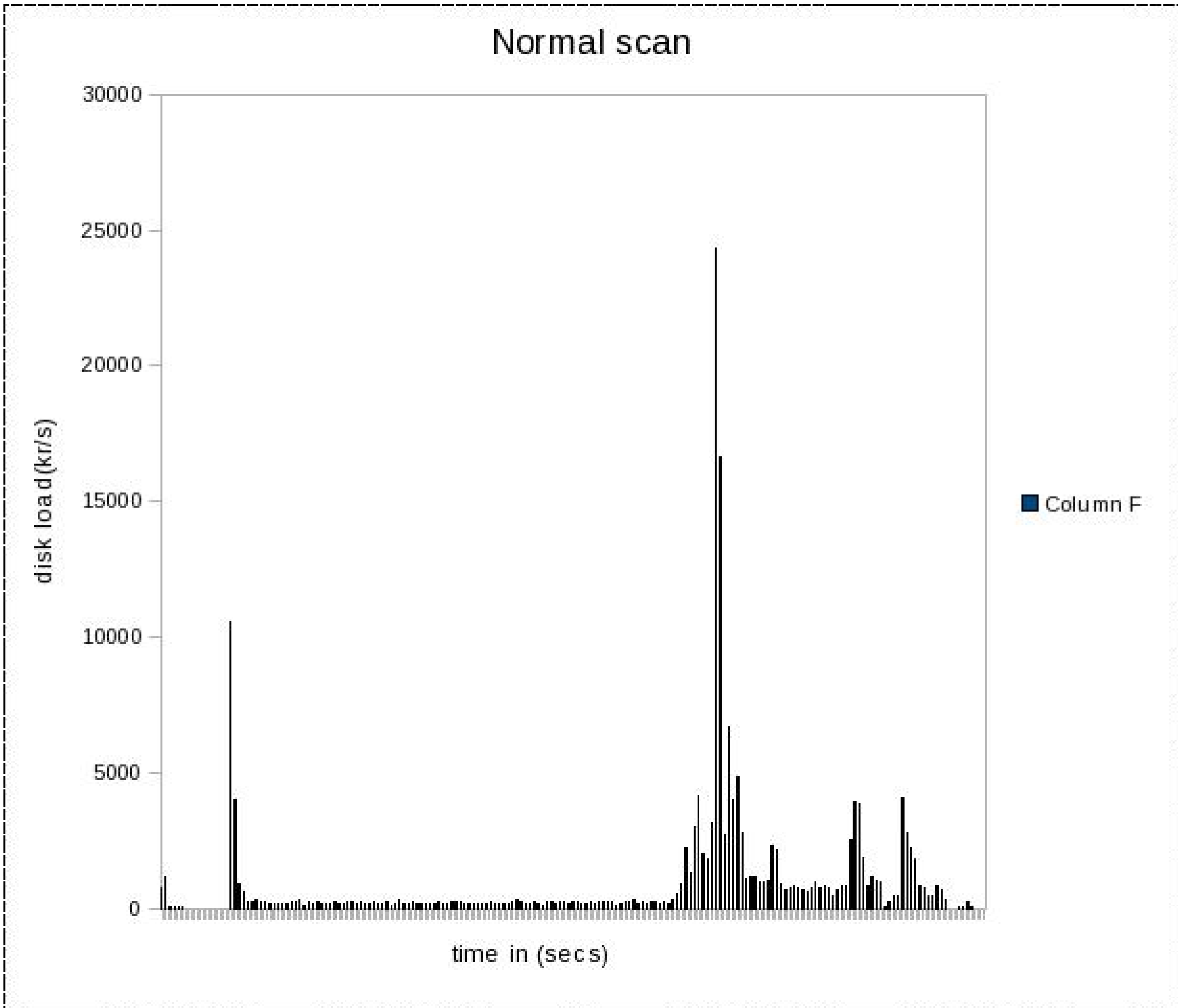
	D	E	F	G	H
1					
2		SUM	172929.2		
3		AVERAGE	1041.74		
4					
5	24.8	13.3	747.8	149.8	
6					
7					
8	32.3	0	1130.3	0	
9					
10					
11	1	0	8	0	
12					
13					
14	1.5	11	6	93.8	
15					
16					
17	0.5	0	6	0	
18					
19					
20	3	33.1	56.1	186.5	
21					
22					
23	0	0	0	0	
24					
25					
26	0	0	0	0	
27					
28					
29	0	5.5	0	39.8	
30					
31					
32	0	0	0	0	



	D	E	F	G	H
532					
533	146.3	0	758.2	0	
534					
535					
536	112.5	11	450	284	
537					
538					
539	116.5	0	484	0	
540					
541					
542	119.3	5.9	802	299	
543					
544					
545	32.3	67.2	664.7	2605	
546					
547					
548	56.5	0	306	0	
549					
550					
551	0	5.5	0	114	
552					
553					
554	0	0	0	0	
555					
556					
557	0.5	3.5	16	23.9	
558					
559					
560	0.5	0	2	0	
561					
562					
563	28.9	0	275.3	0	

Net disk I/O caused by vascan virus scanner during the scan - 172929.2*2=345858.4 KB

Scan plot-

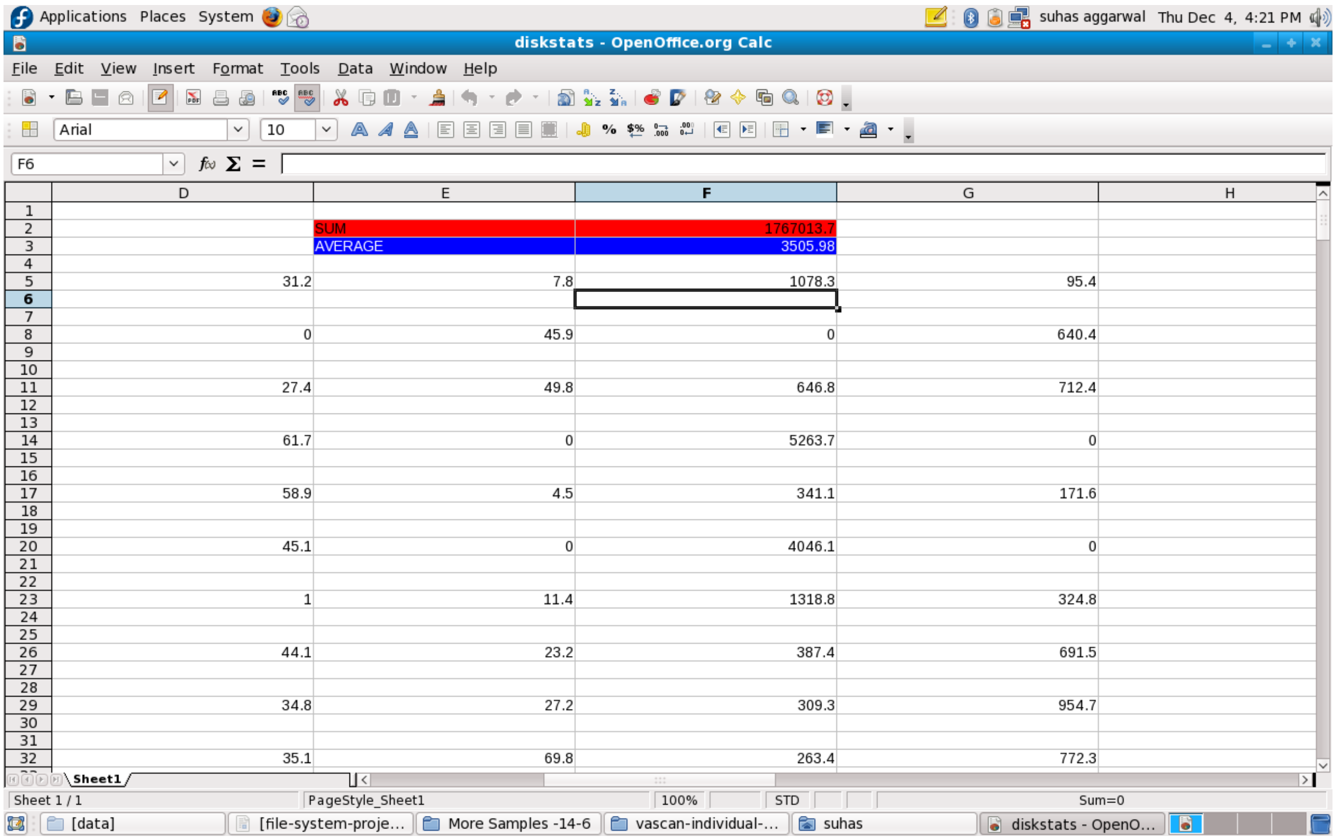


This plot depicts disk reads done by vascan virus scanner during the scan.(plot of kr/s field in iostat output)

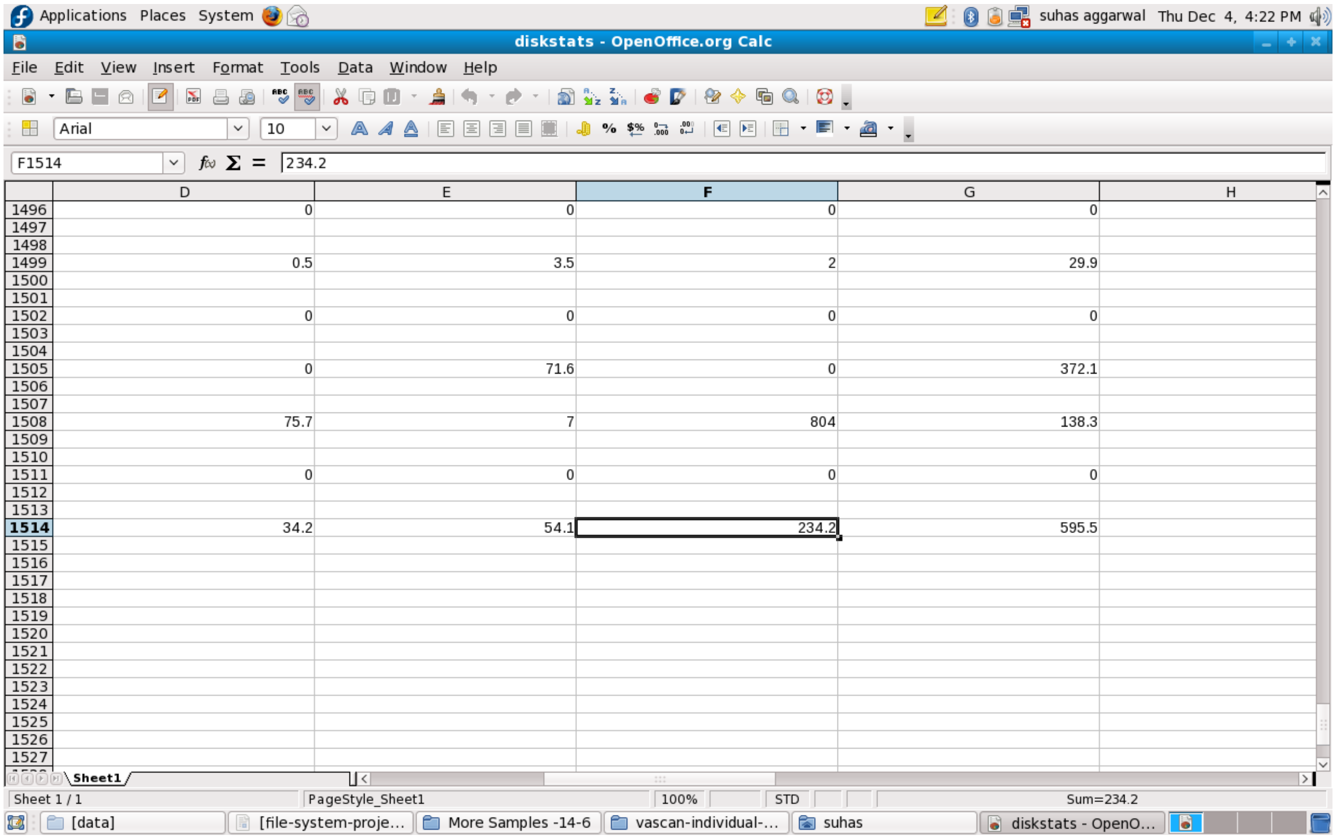
4.1.4 CLAMAV Virus scan Anaylsis –

Disk I/O report generated by iostat for CLAMAV virus scanner –

CLAMAV Virus scanner



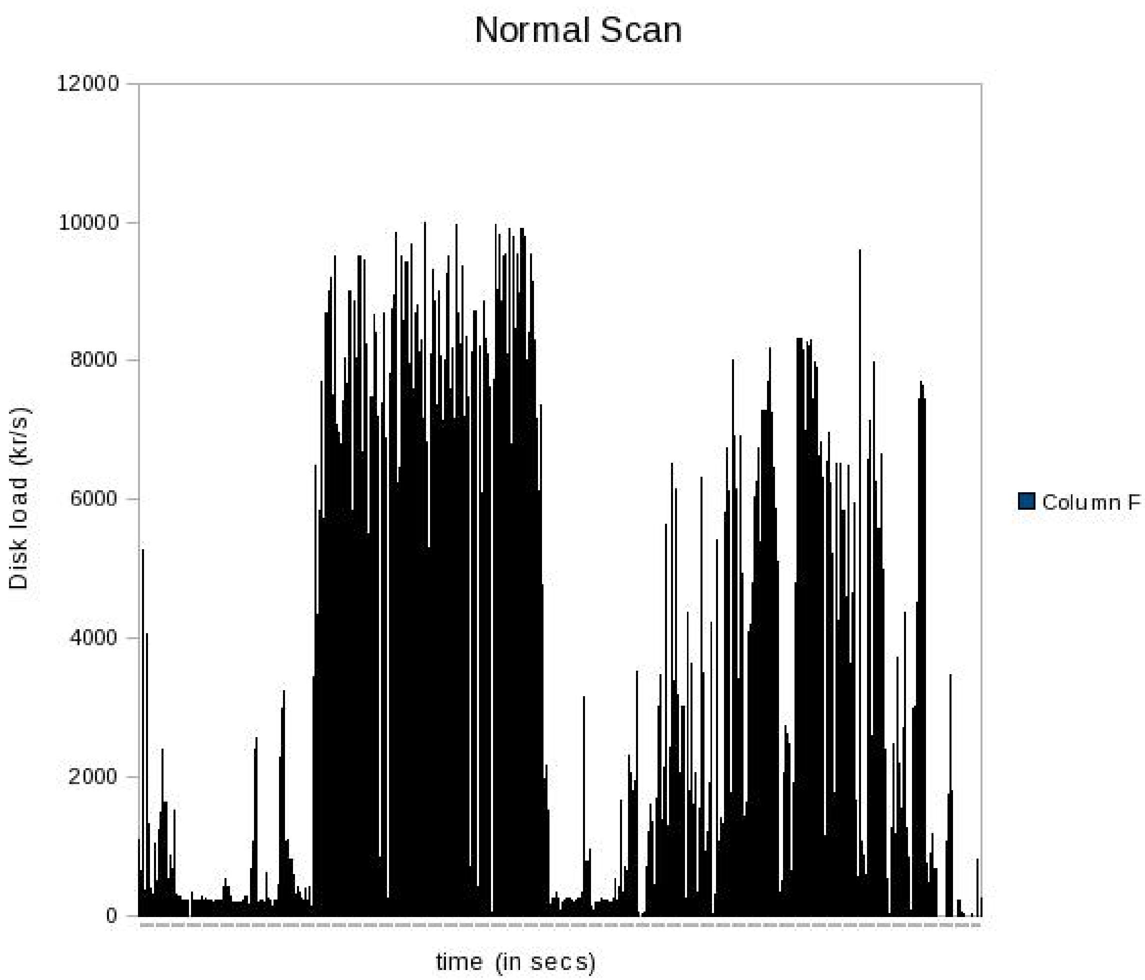
	D	E	F	G	H
1					
2		SUM	1767013.7		
3		AVERAGE	3505.98		
4					
5	31.2	7.8	1078.3	95.4	
6					
7					
8	0	45.9	0	640.4	
9					
10					
11	27.4	49.8	646.8	712.4	
12					
13					
14	61.7	0	5263.7	0	
15					
16					
17	58.9	4.5	341.1	171.6	
18					
19					
20	45.1	0	4046.1	0	
21					
22					
23	1	11.4	1318.8	324.8	
24					
25					
26	44.1	23.2	387.4	691.5	
27					
28					
29	34.8	27.2	309.3	954.7	
30					
31					
32	35.1	69.8	263.4	772.3	



	D	E	F	G	H
1496	0	0	0	0	
1497					
1498					
1499	0.5	3.5	2	29.9	
1500					
1501					
1502	0	0	0	0	
1503					
1504					
1505	0	71.6	0	372.1	
1506					
1507					
1508	75.7	7	804	138.3	
1509					
1510					
1511	0	0	0	0	
1512					
1513					
1514	34.2	54.1	234.2	595.5	
1515					
1516					
1517					
1518					
1519					
1520					
1521					
1522					
1523					
1524					
1525					
1526					
1527					

Net Disk I/O caused by CLAM AV during the scan -
1767013.7*2=3534027.4KB

Scan plot-



This plot shows disk reads done by CLAMAV virus scanner during the scan.(plot of kr/s field in iostat output)

Net Disk I/O caused by three virus scanners (Avast + Vascan + ClamAV)
1154769.6KB + 345858.4KB + 3534027.4KB = 5034655.4KB

4.2. Scheduling the 3 virus scanners with a scheduling algorithm namely round robin scheme using Reducer -

1)
AIM: To study Disk I/O activity when 3 virus scanners are scheduled with a round robin scheme with different time quanta settings.

2)
System set up -
Operating system – Fedora core 6
Filesystem - ext3
RAM- 512 MB
Processor-Pentium 4 Processor ,3 GHz
Disk Space occupied – 2.25 GB

3)
Scanners used -
1)Avast – avast4workstation-1.0.8

2)Vascan – vascan-1.3.4-4.3.23-Linux-i386

3)ClamAV- Engine version-0.91.2

4)Tool used to monitor Disk I/O activity –iostat

Command used -
iostat –x 2 >>filename

Workload -
It is to be ensured that no other process is running (especially ,one demanding high Disk I/O)when schedular code is running

Steps -
1)
iostat –x 2>>filename
Above command is initiated to log Disk I/O activity.

2)
Scan schedular is started to schedule the scans
Schedular schedules the scans according to round robin scheduling algorithm and FIFO depending on choice selected.

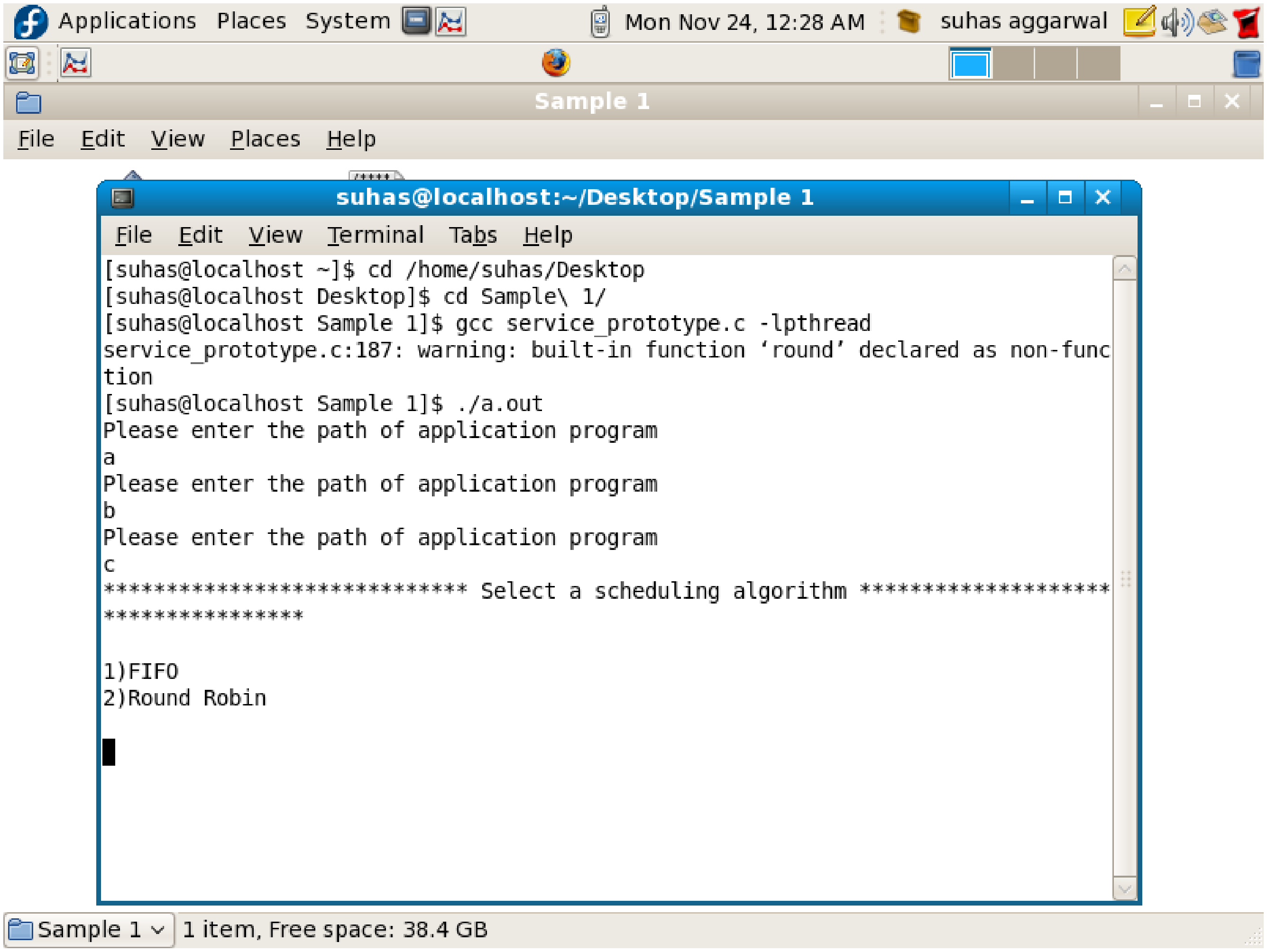
3)
Schedular also creates File scanning logs of 3 virus scanners in 3 separate files.

4)
Kilobytes of data read per second (kr/s field of iostat output) from the disk is analysed in scan sheets (obtained via iostat)

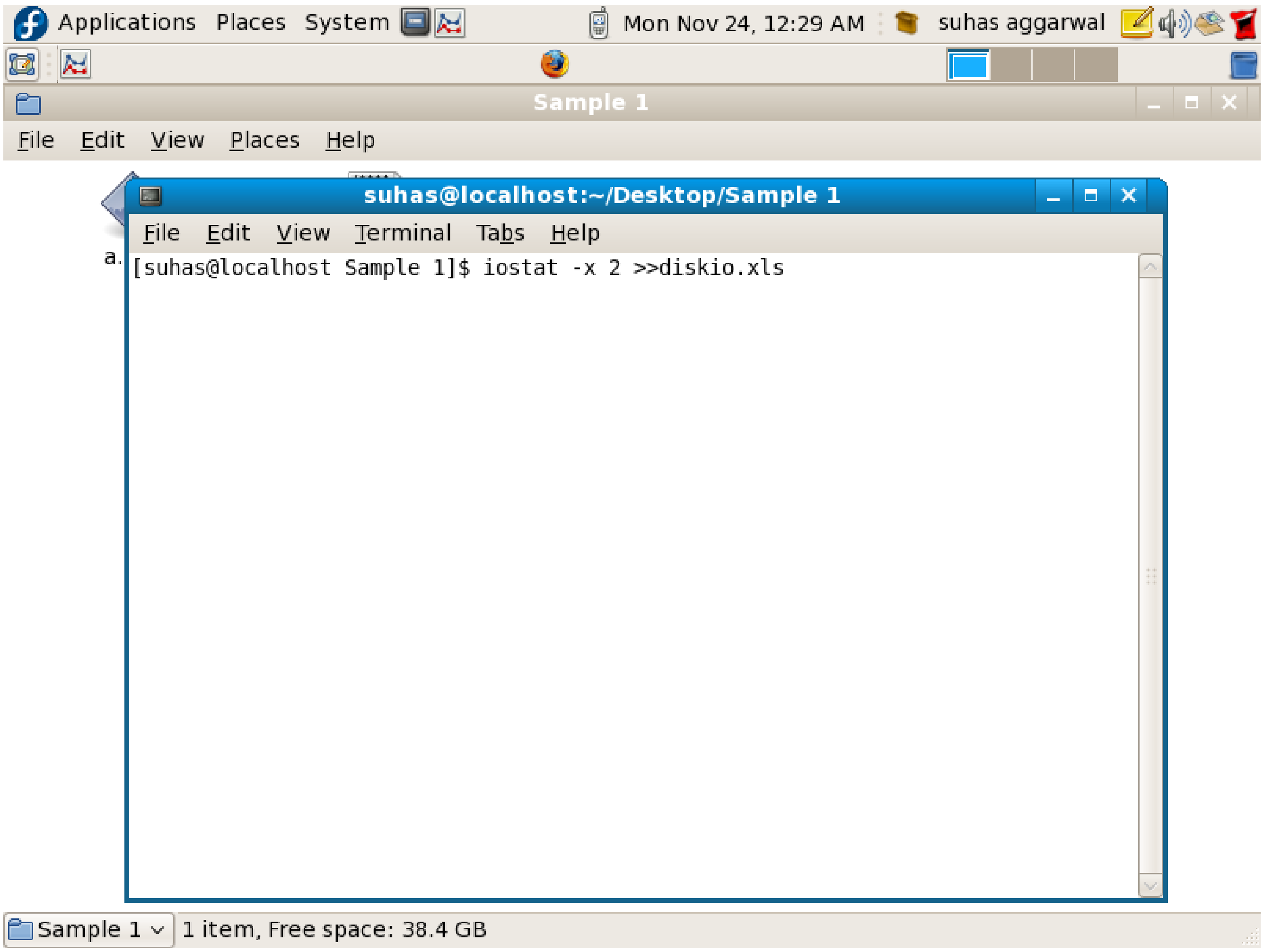
5)
Sum and **average** of Disk I/O caused by scanners is computed

A sample experiment run-

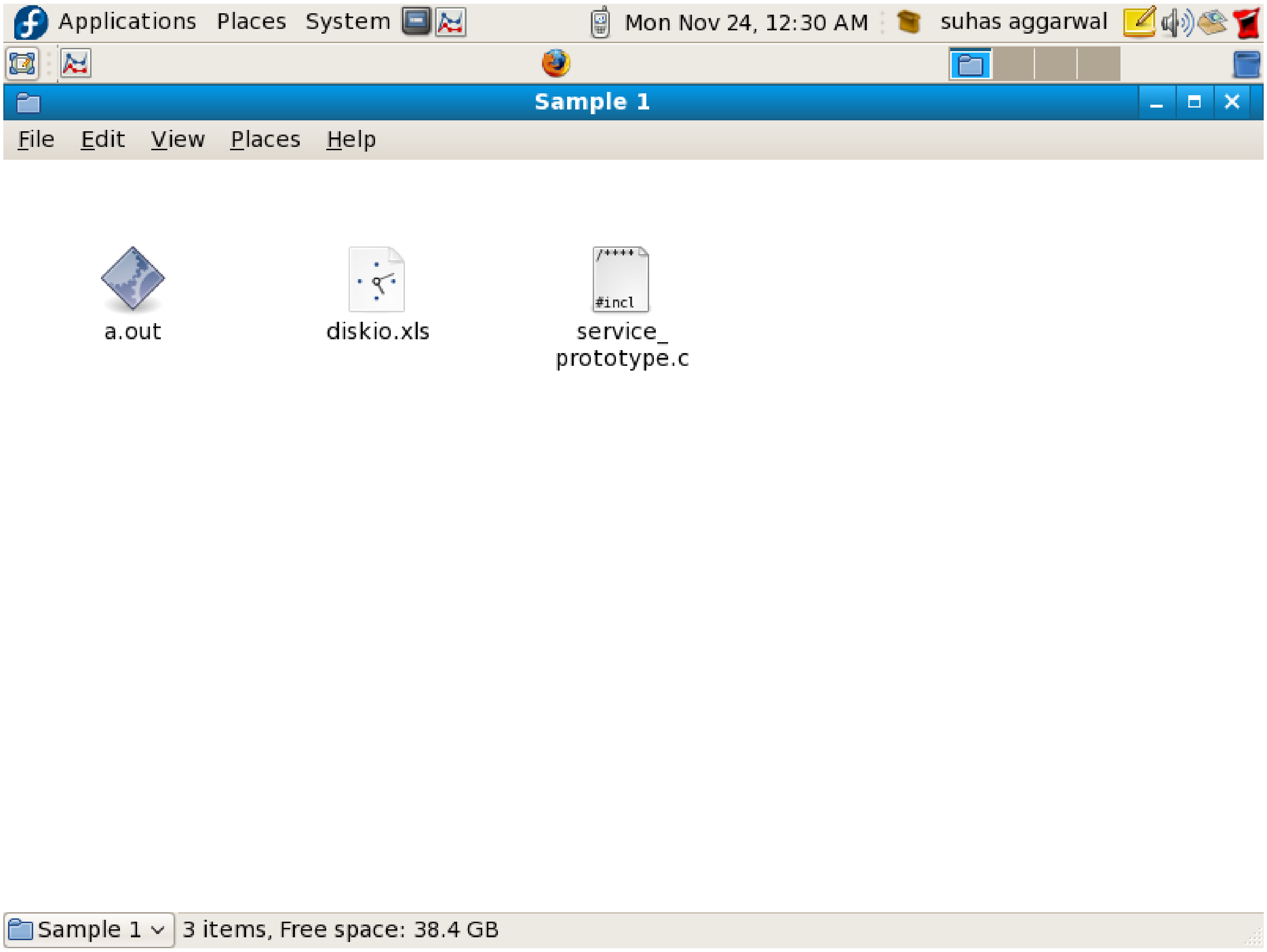
1)Start the scan schedular



1. Start iostat tool to monitor Disk I/O activity.
   Command used -
   iostat -x 2 >>diskio.xls
   (Take reading every 2 second and save the log in a file)

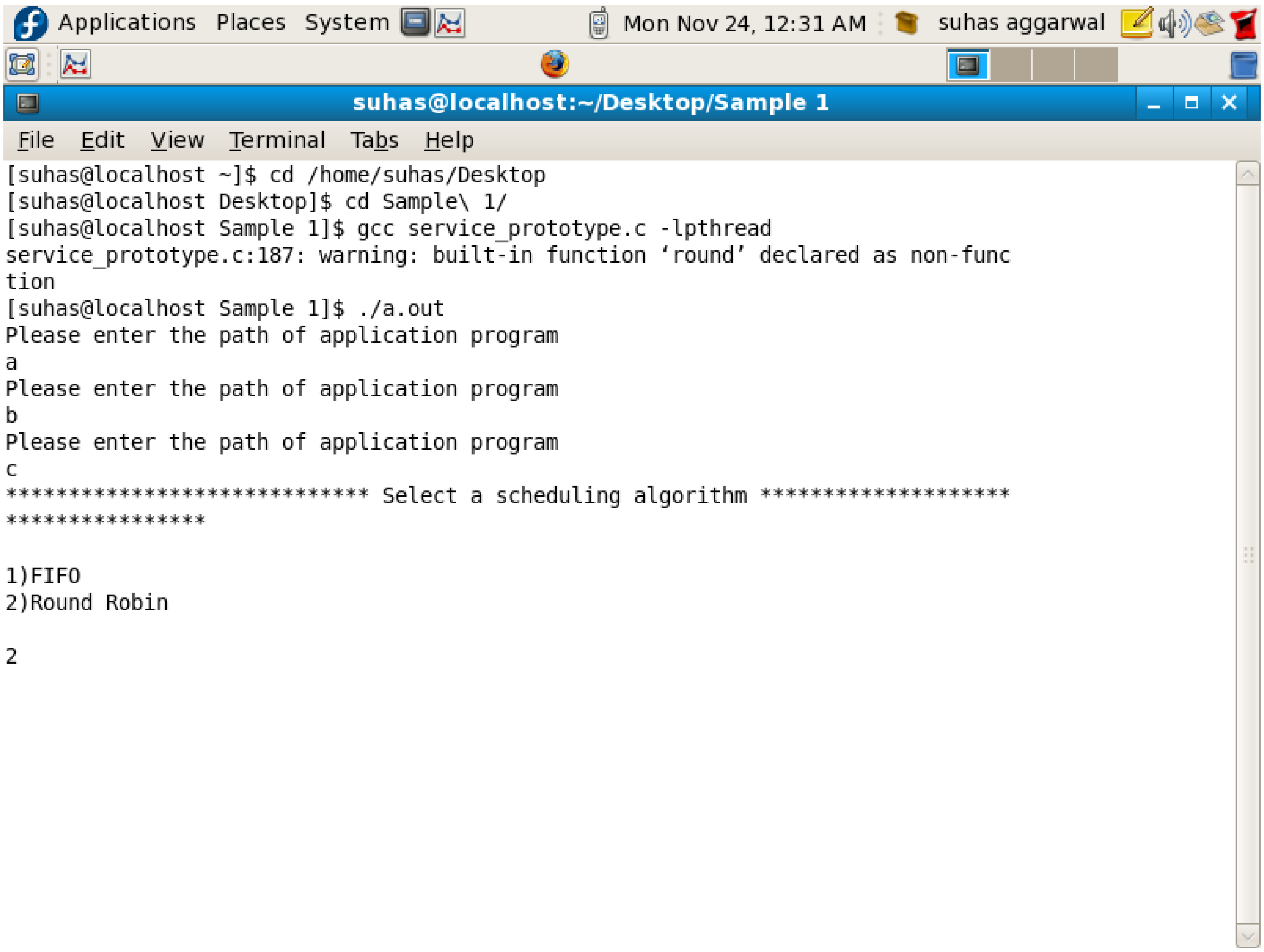


One can observe that log file is created in the experiment directory.

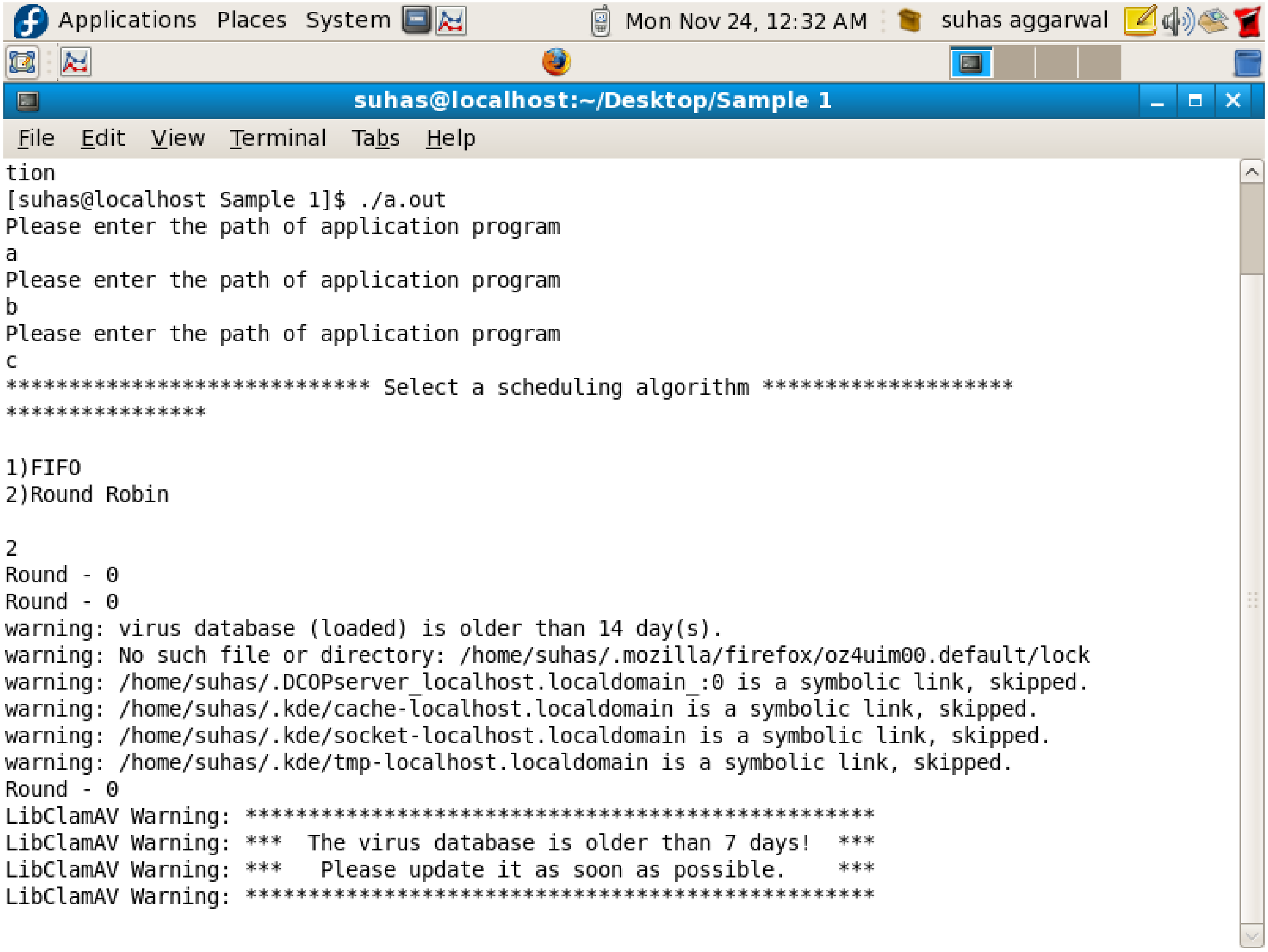


3)Select the scheduling algorithm - (2nd choice for Round robin algorithm)
For setting time quanta (time_quanta field in the schedular code has to be edited)

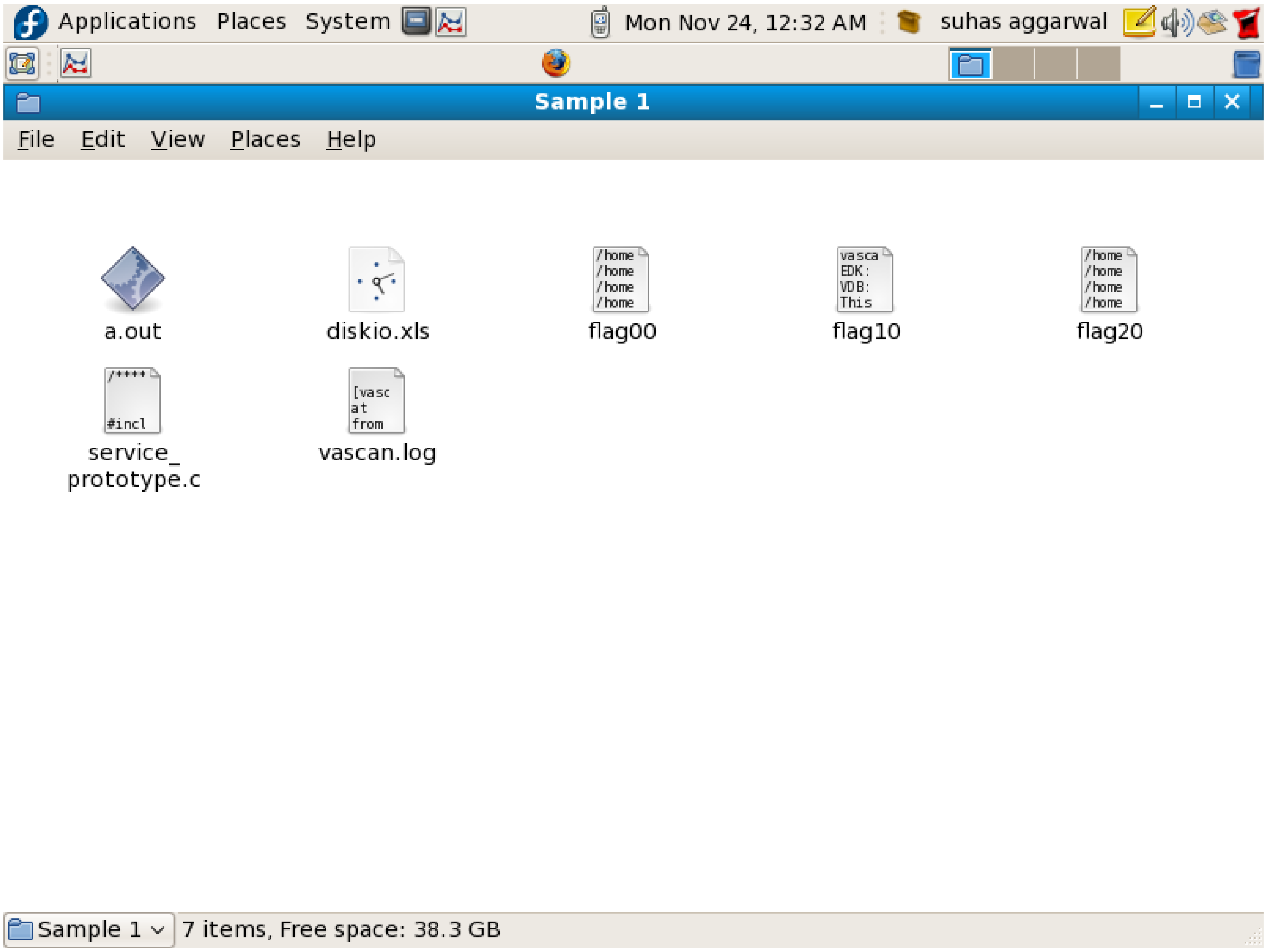
Note:FIFO is same as round robin with time quanta t = 600 in this scenario)



One can observe that 1st round of round robin algorithm has started.

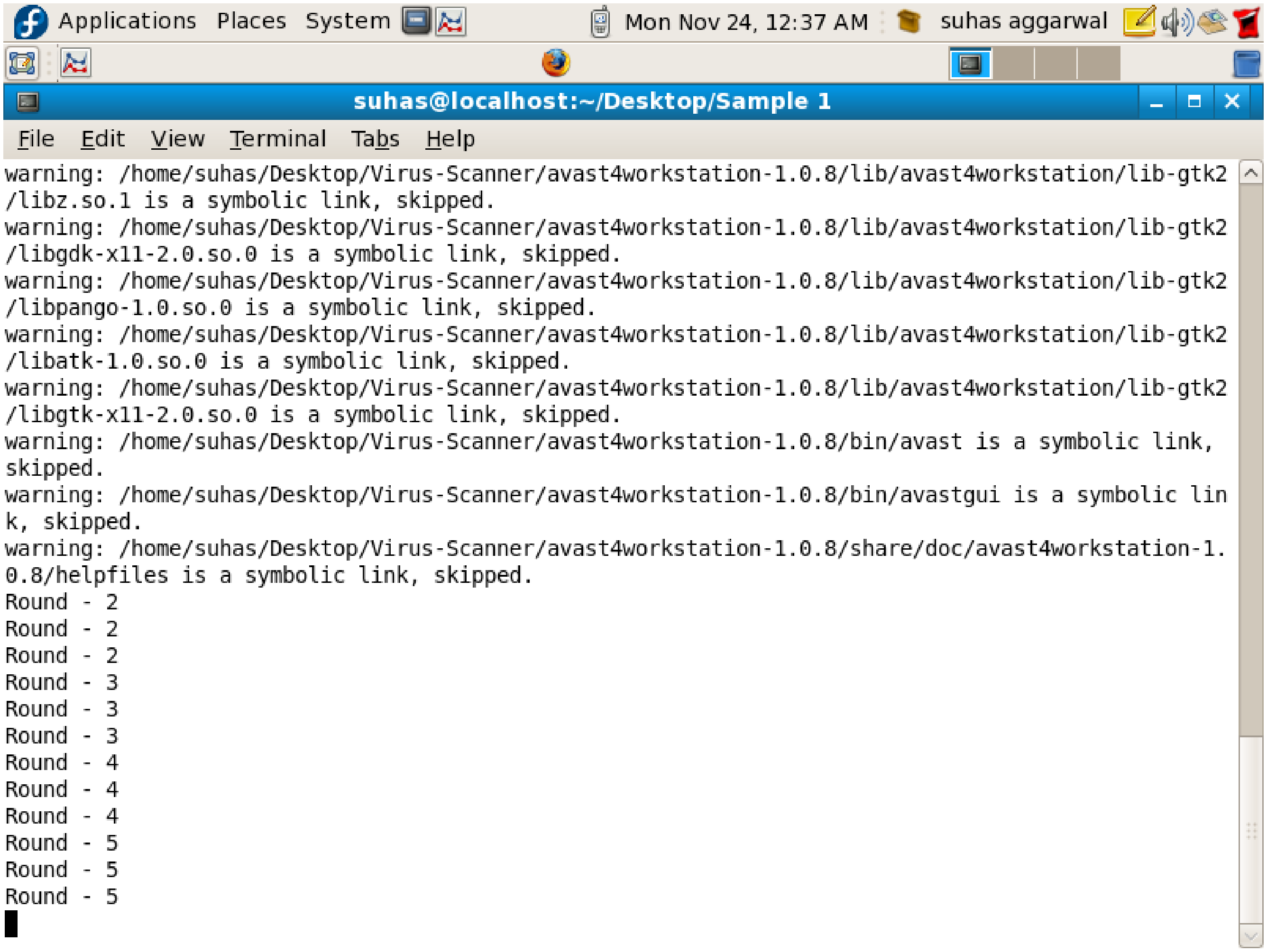


One can observe that scan report of three virus scanners are stored in three separate log files (flag00,flag10,flag20) by the the schedular (it directs the output of three scanners to these files as can

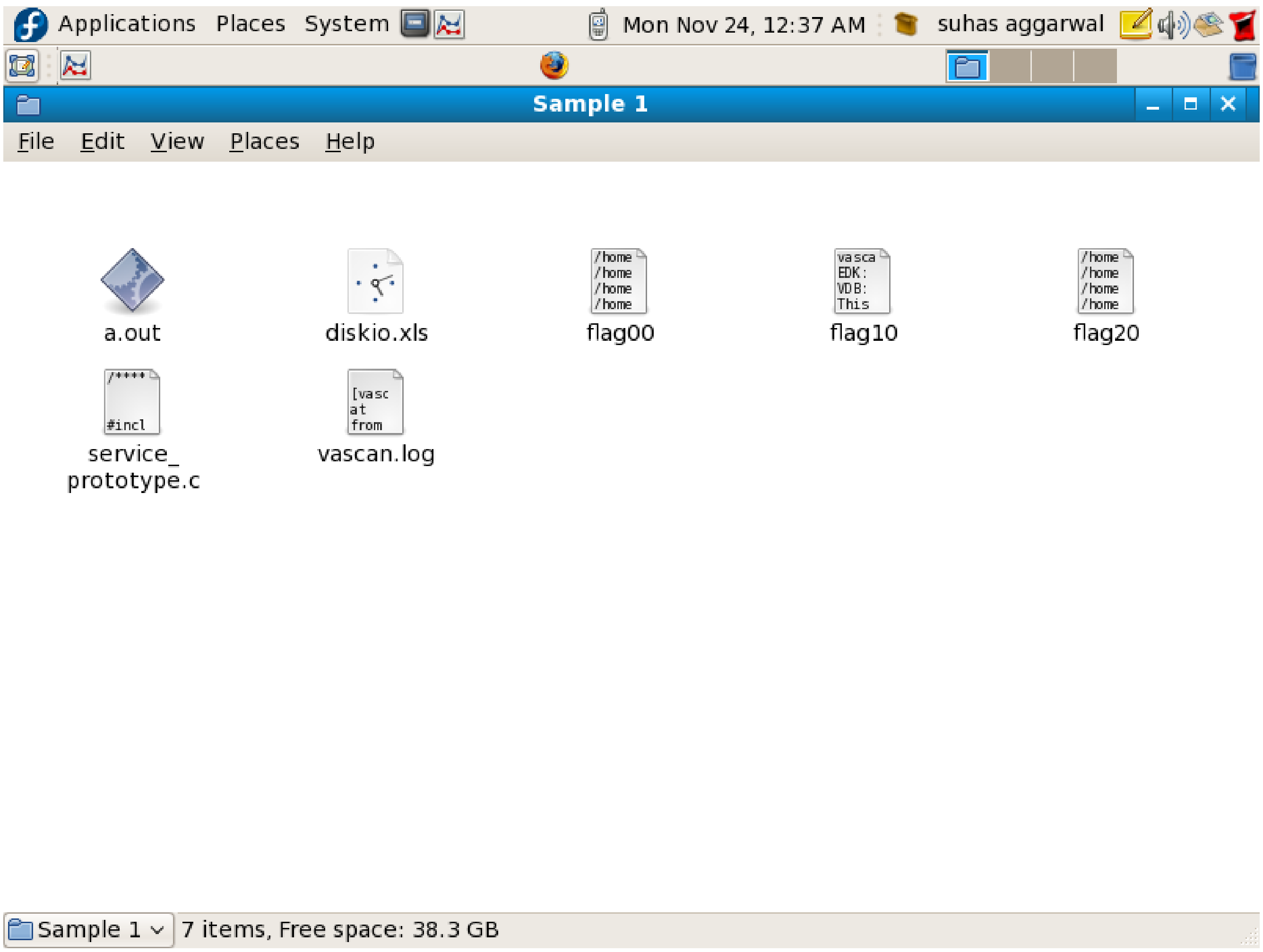


be observed from the code)

One can observe higher rounds of round robin (a round robin scan is about to be completed..)



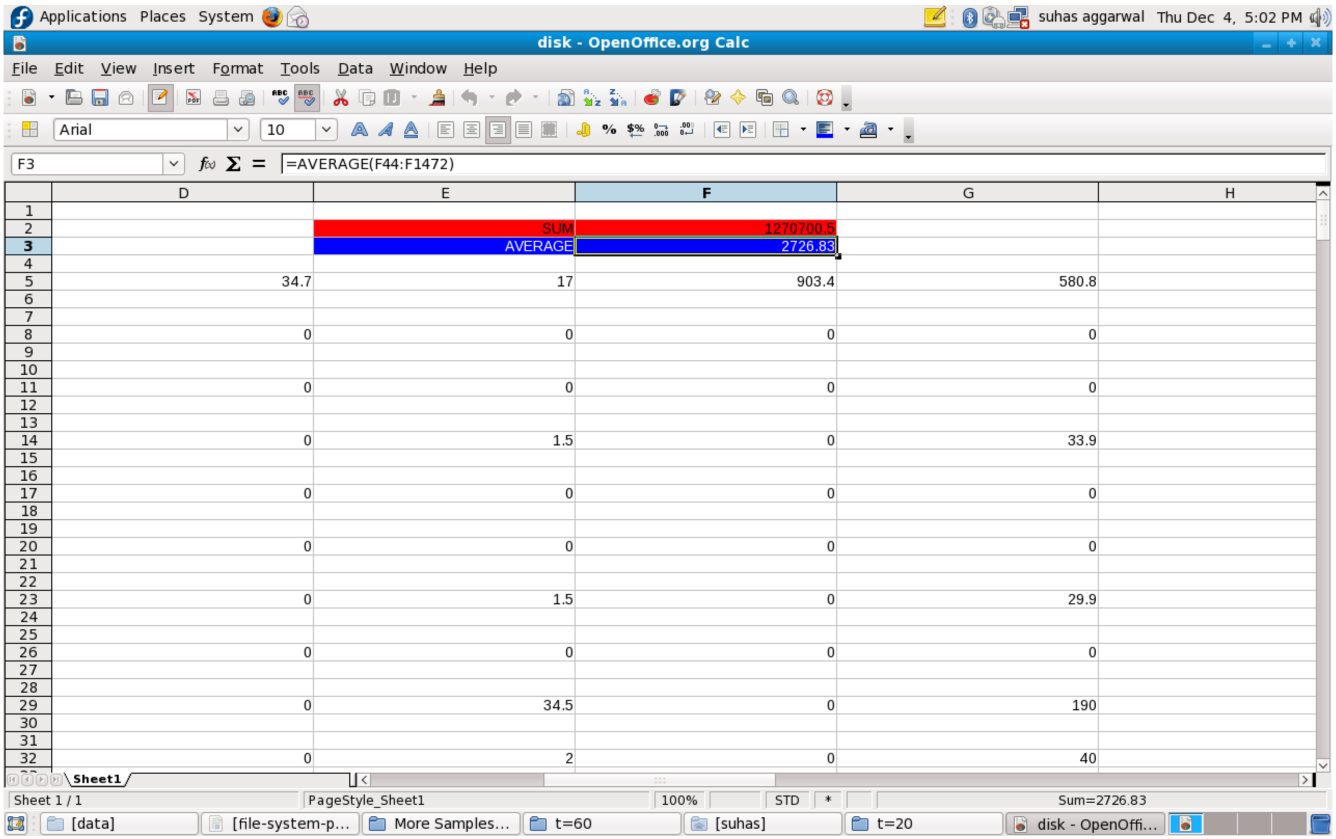
One can observe the directory when scan has been completed (Complete scan reports are stored in 3 files - flag 00,flag10,flag20 by the schedular,Disk I/O activity report during the scan is stored by iostat tool in diskio file.



4.2.1 Round Robin scan Anaylsis (t =20 seconds)

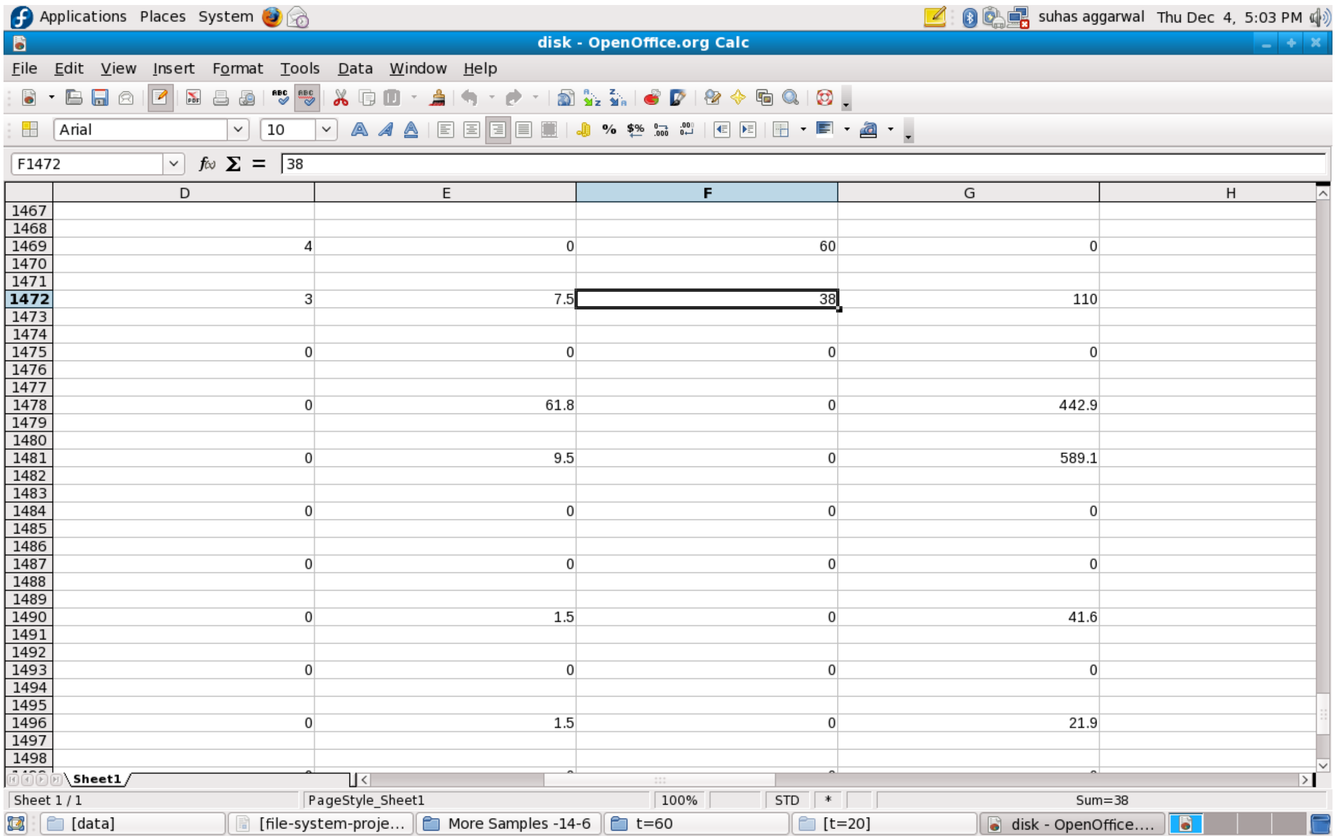
Disk I/O scan report generated by iostat tool for round robin scheme with time quanta t= 20 seconds –

t=20 seconds



	D	E	F	G	H
1					
2		SUM	1270700.5		
3		AVERAGE	2726.83		
4					
5	34.7	17	903.4	580.8	
6					
7					
8	0	0	0	0	
9					
10					
11	0	0	0	0	
12					
13					
14	0	1.5	0	33.9	
15					
16					
17	0	0	0	0	
18					
19					
20	0	0	0	0	
21					
22					
23	0	1.5	0	29.9	
24					
25					
26	0	0	0	0	
27					
28					
29	0	34.5	0	190	
30					
31					
32	0	2	0	40	

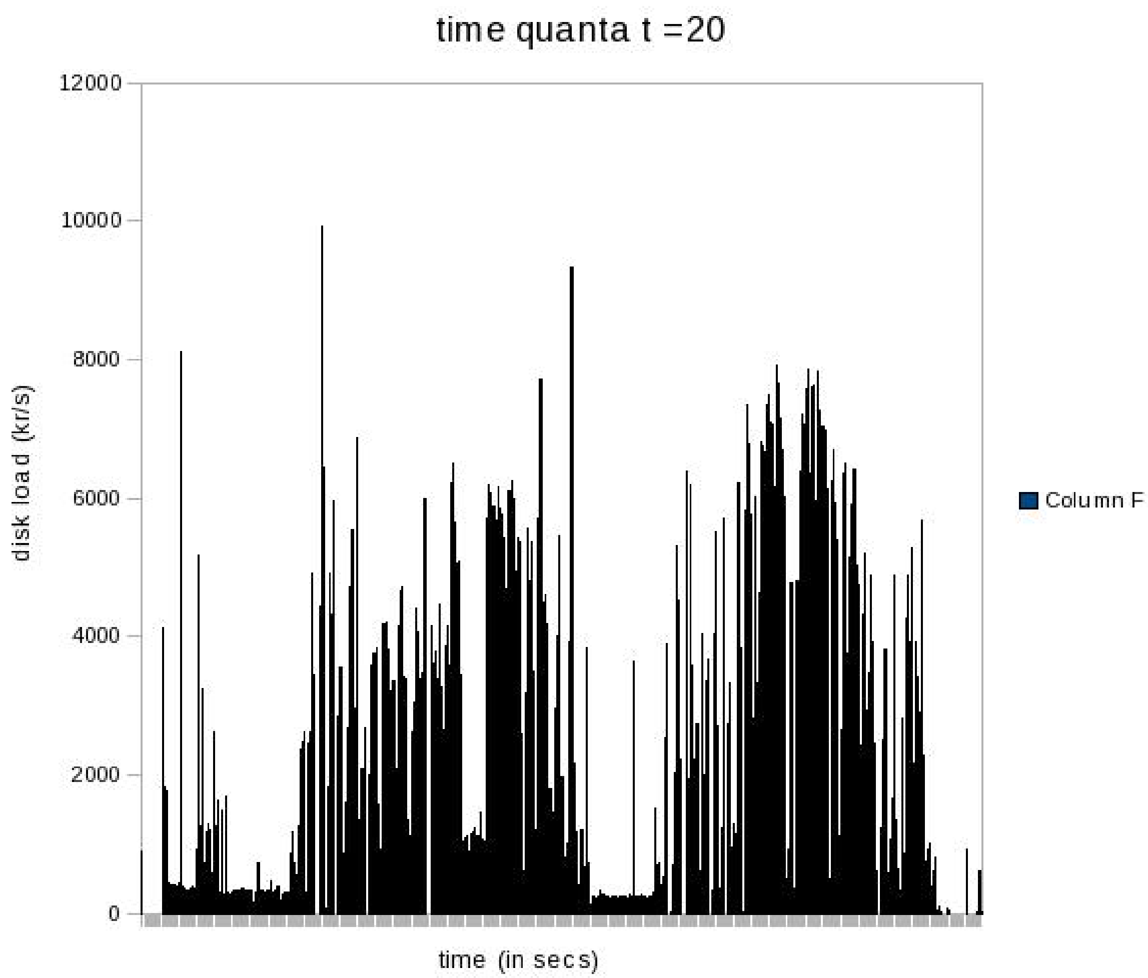
Note: For complete scan sheets, please refer to Data Samples.



	D	E	F	G	H
1467					
1468					
1469	4	0	60	0	
1470					
1471					
1472	3	7.5	38	110	
1473					
1474					
1475	0	0	0	0	
1476					
1477					
1478	0	61.8	0	442.9	
1479					
1480					
1481	0	9.5	0	589.1	
1482					
1483					
1484	0	0	0	0	
1485					
1486					
1487	0	0	0	0	
1488					
1489					
1490	0	1.5	0	41.6	
1491					
1492					
1493	0	0	0	0	
1494					
1495					
1496	0	1.5	0	21.9	
1497					
1498					

Net Disk I/O caused by three scanners when scheduled by round robin scheme with time quantum -20 seconds – 1270700.5KB*2=2541401KB

Scan plot -

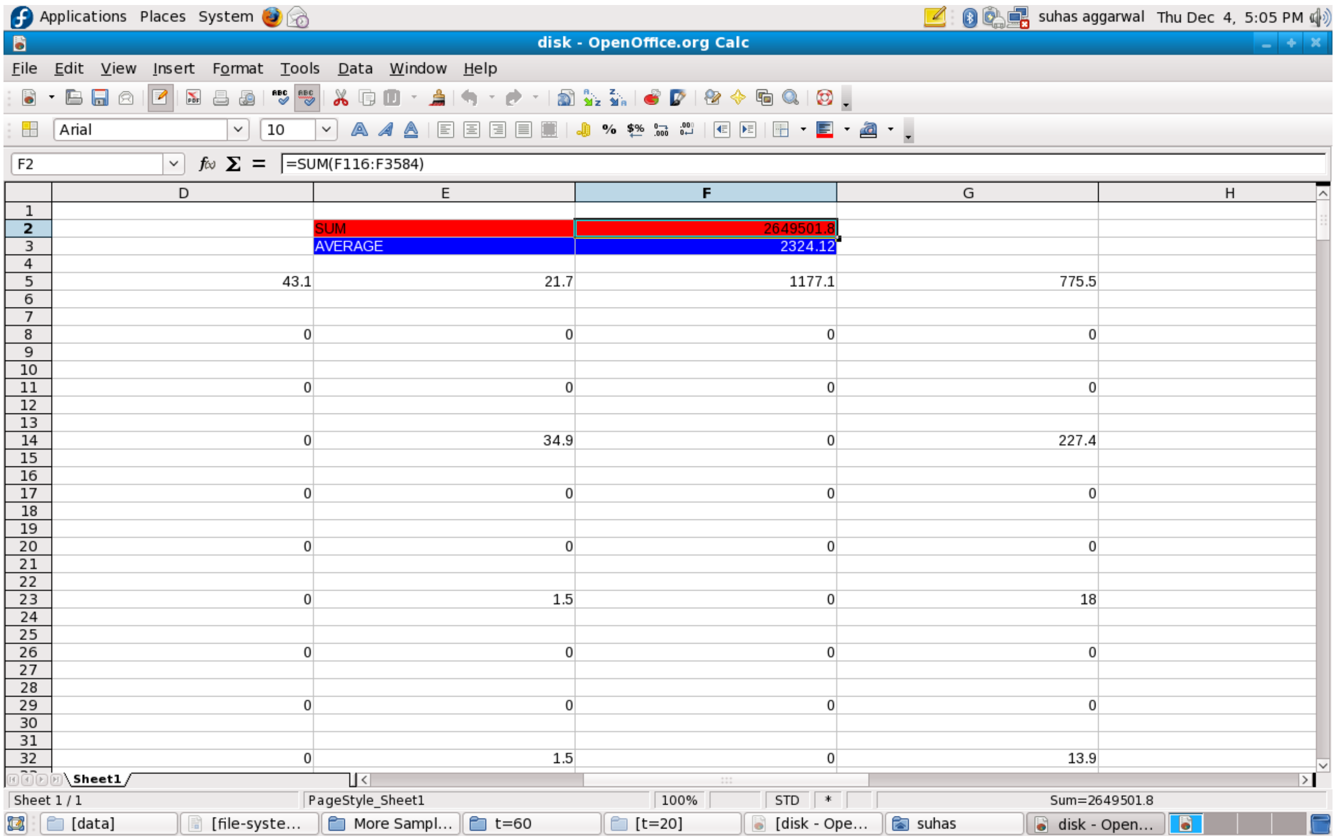


This plot shows disk reads done by 3 scanners -avast ,vascan and clamav when scheduled with round robin scheme with time quanta t =20 seconds.

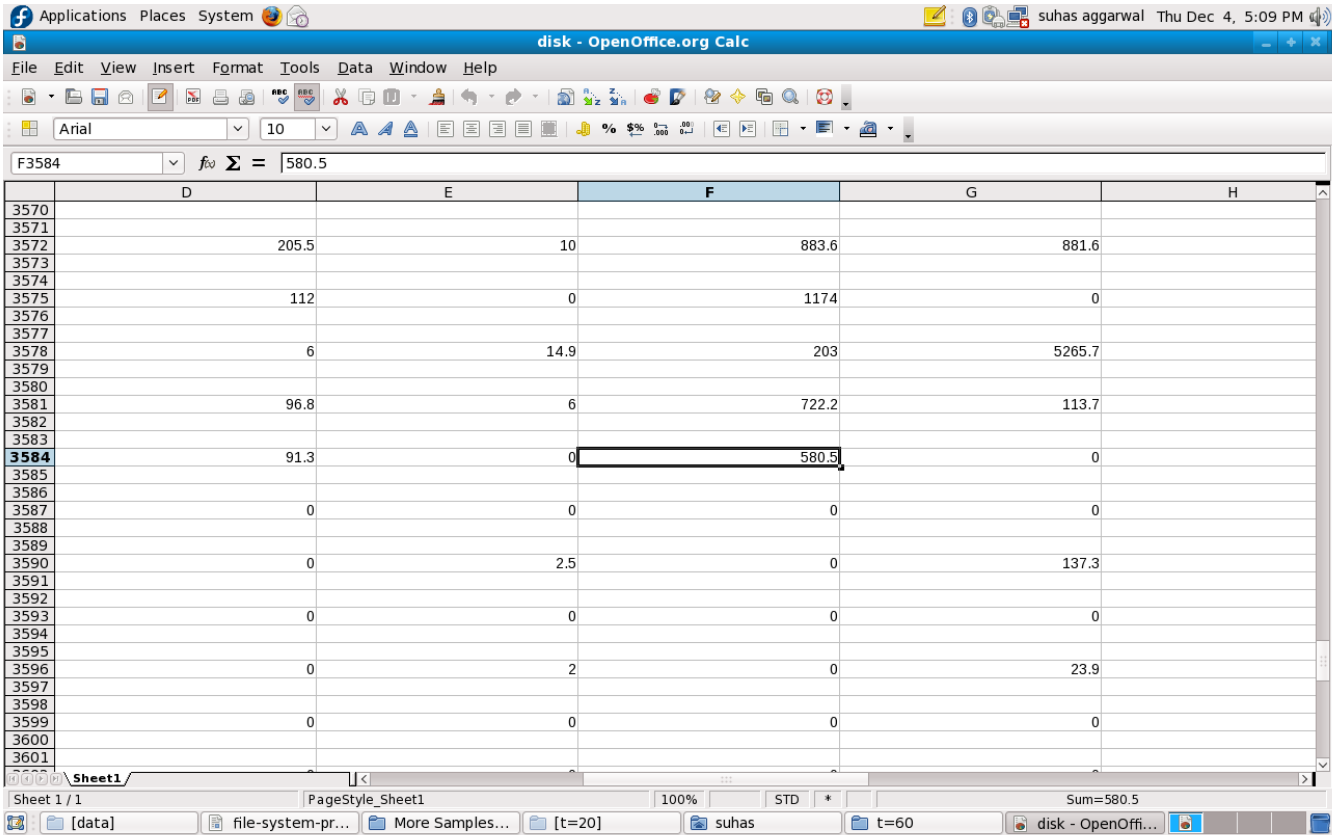
4.2.2 Round Robin scan Anaylsis (t =60 seconds)

Disk I/O report generated by iostat for time quantum t = 60 secs -

t=60 seconds



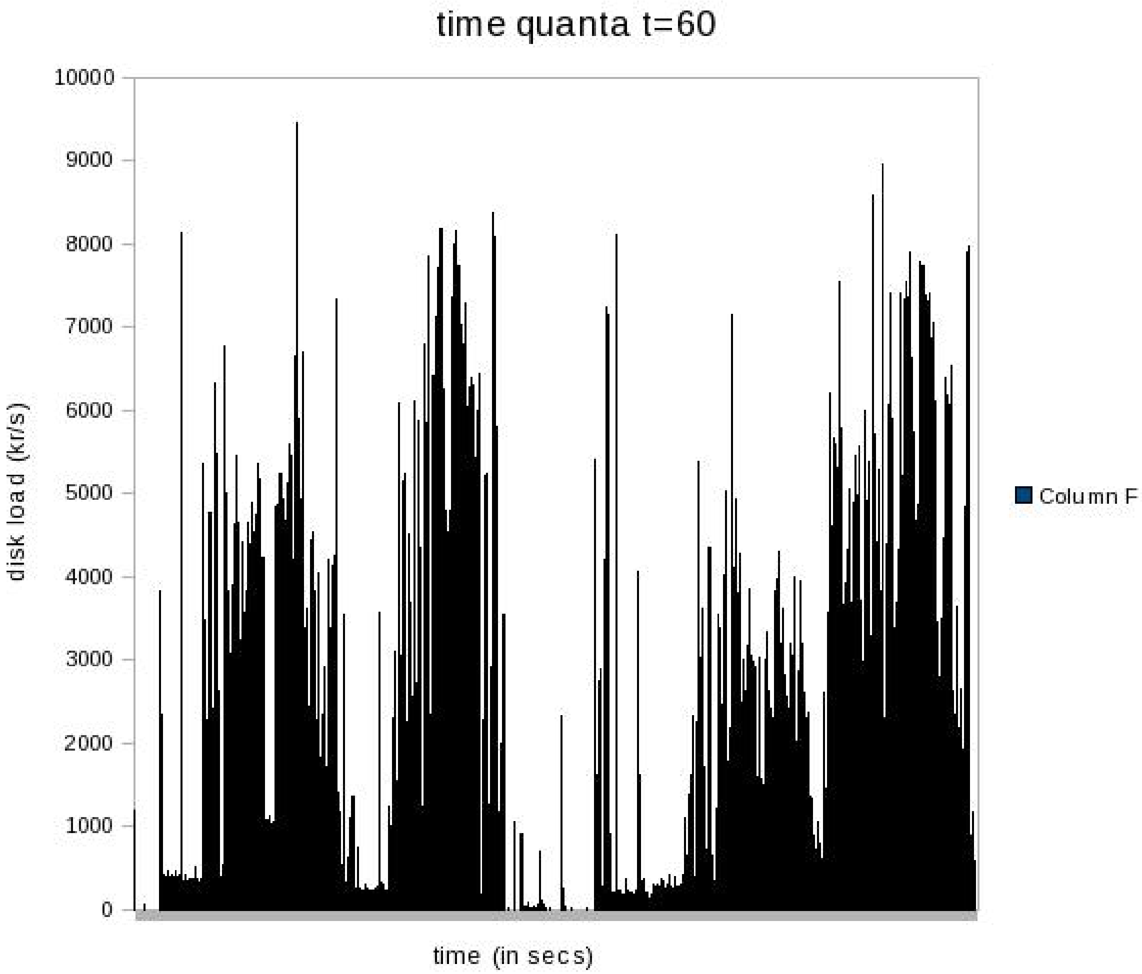
	D	E	F	G	H
1					
2		SUM	2649501.8		
3		AVERAGE	2324.12		
4					
5	43.1	21.7	1177.1	775.5	
6					
7					
8	0	0	0	0	
9					
10					
11	0	0	0	0	
12					
13					
14	0	34.9	0	227.4	
15					
16					
17	0	0	0	0	
18					
19					
20	0	0	0	0	
21					
22					
23	0	1.5	0	18	
24					
25					
26	0	0	0	0	
27					
28					
29	0	0	0	0	
30					
31					
32	0	1.5	0	13.9	



	D	E	F	G	H
3570					
3571					
3572	205.5	10	883.6	881.6	
3573					
3574					
3575	112	0	1174	0	
3576					
3577					
3578	6	14.9	203	5265.7	
3579					
3580					
3581	96.8	6	722.2	113.7	
3582					
3583					
3584	91.3	0	580.5	0	
3585					
3586					
3587	0	0	0	0	
3588					
3589					
3590	0	2.5	0	137.3	
3591					
3592					
3593	0	0	0	0	
3594					
3595					
3596	0	2	0	23.9	
3597					
3598					
3599	0	0	0	0	
3600					
3601					

Net disk I/O caused by 3 scanners when scheduled with round robin scheme with time quanta t =60 seconds -2649501.8*2 =5299003.6KB

Scan plot -

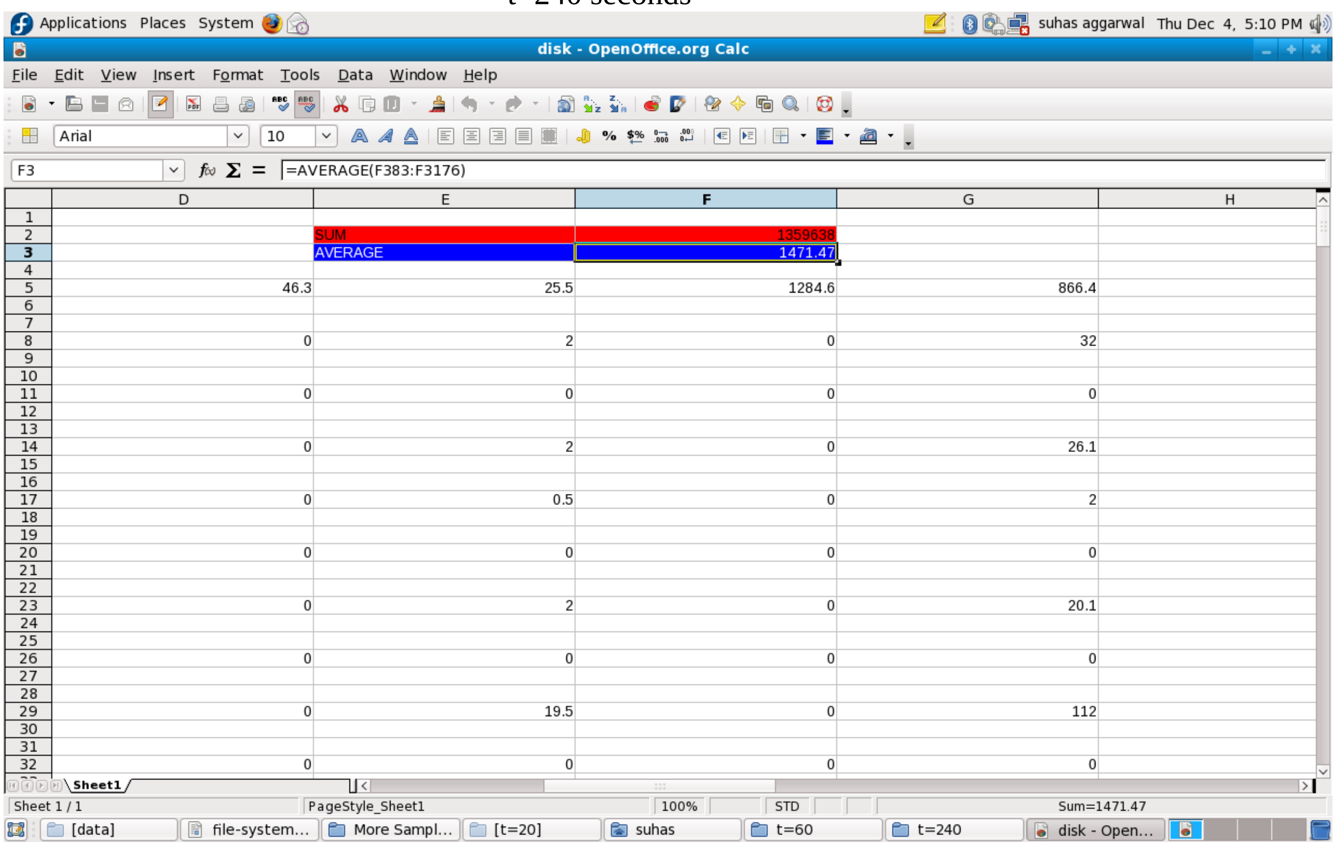


This plot depicts disk read done by 3 scanners when scheduled using round robin scheme with time quanta t= 60 seconds .

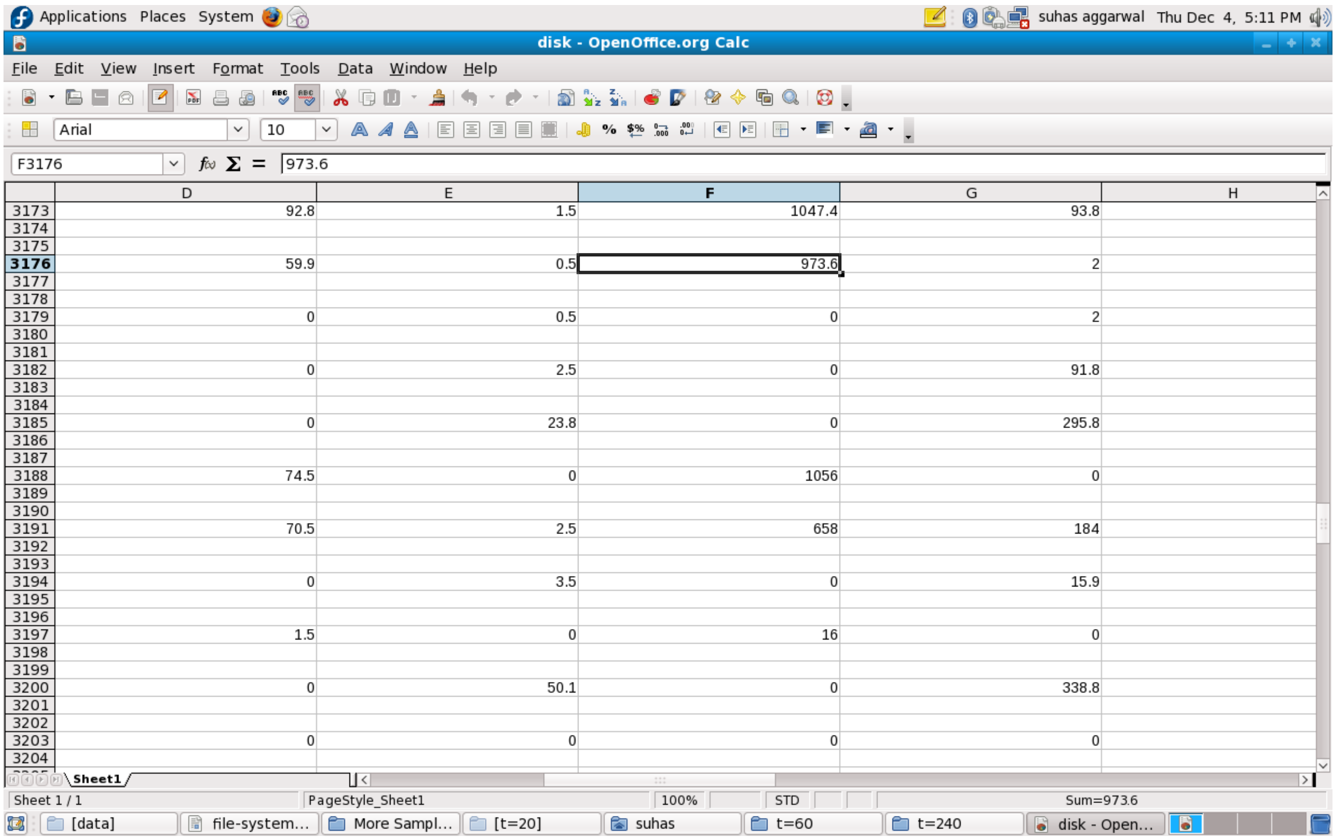
4.2.3 Round Robin scan Anaylsis (t =240 seconds)

Disk I/O report generated by iostat when 3 scanners were scheduled with round robin scheme with time quanta t =240 seconds.

t=240 seconds

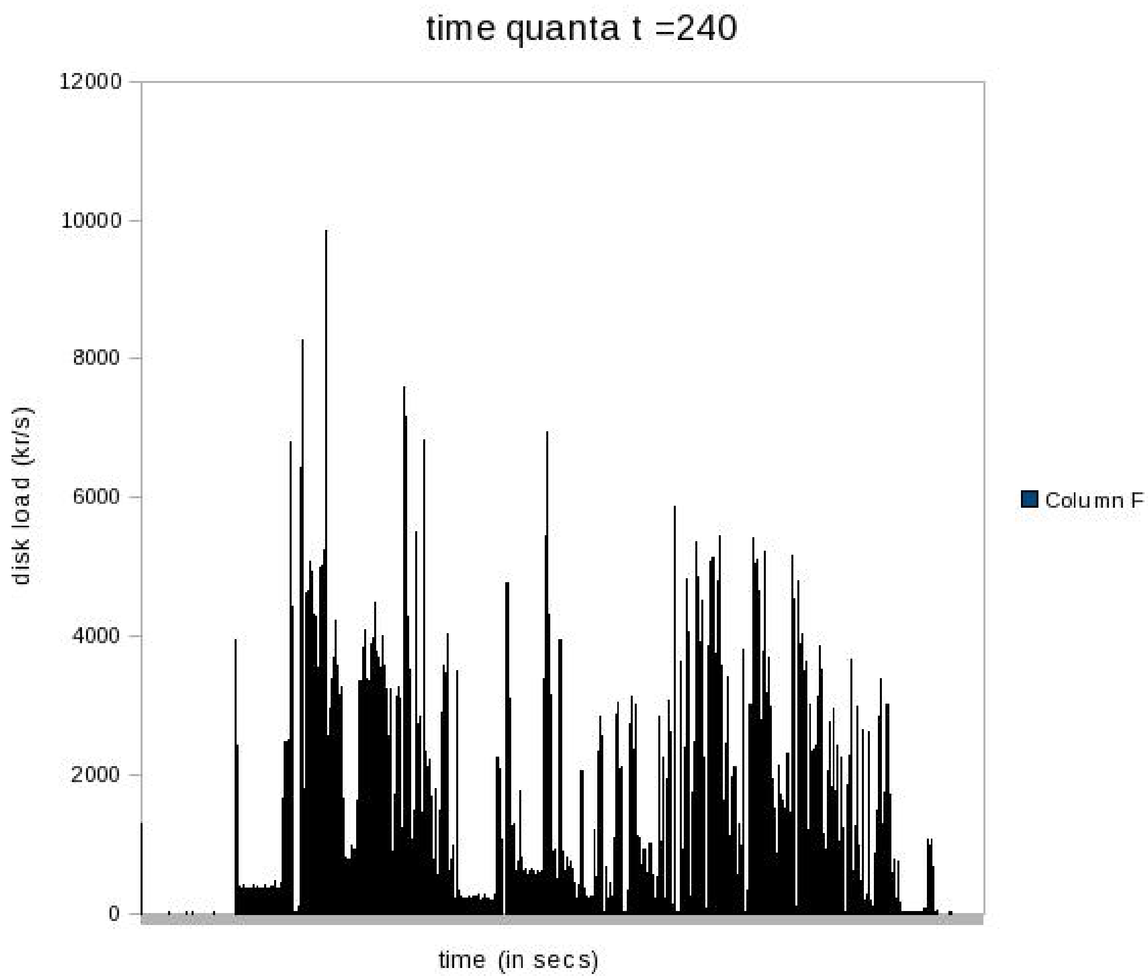


	D	E	F	G	H
1					
2		SUM	1359638		
3		AVERAGE	1471.47		
4					
5	46.3	25.5	1284.6	866.4	
6					
7					
8	0	2	0	32	
9					
10					
11	0	0	0	0	
12					
13					
14	0	2	0	26.1	
15					
16					
17	0	0.5	0	2	
18					
19					
20	0	0	0	0	
21					
22					
23	0	2	0	20.1	
24					
25					
26	0	0	0	0	
27					
28					
29	0	19.5	0	112	
30					
31					
32	0	0	0	0	



Net Disk I/O caused by 3 scanners when scheduled with round robin scheme with time quanta t =240 seconds -1359638 * 2 KB

Scan plot -



This plot depicts disk reads done by scanners when 3 scanners avast ,vascan,CLAMAV were scheduled with round robin scheme with time quanta t= 240 seconds.

4.3. Net Disk I/O caused by scanners in different cases –

1)When scheduled individually (equal to net Disk I/O for a **normal periodic scan**)

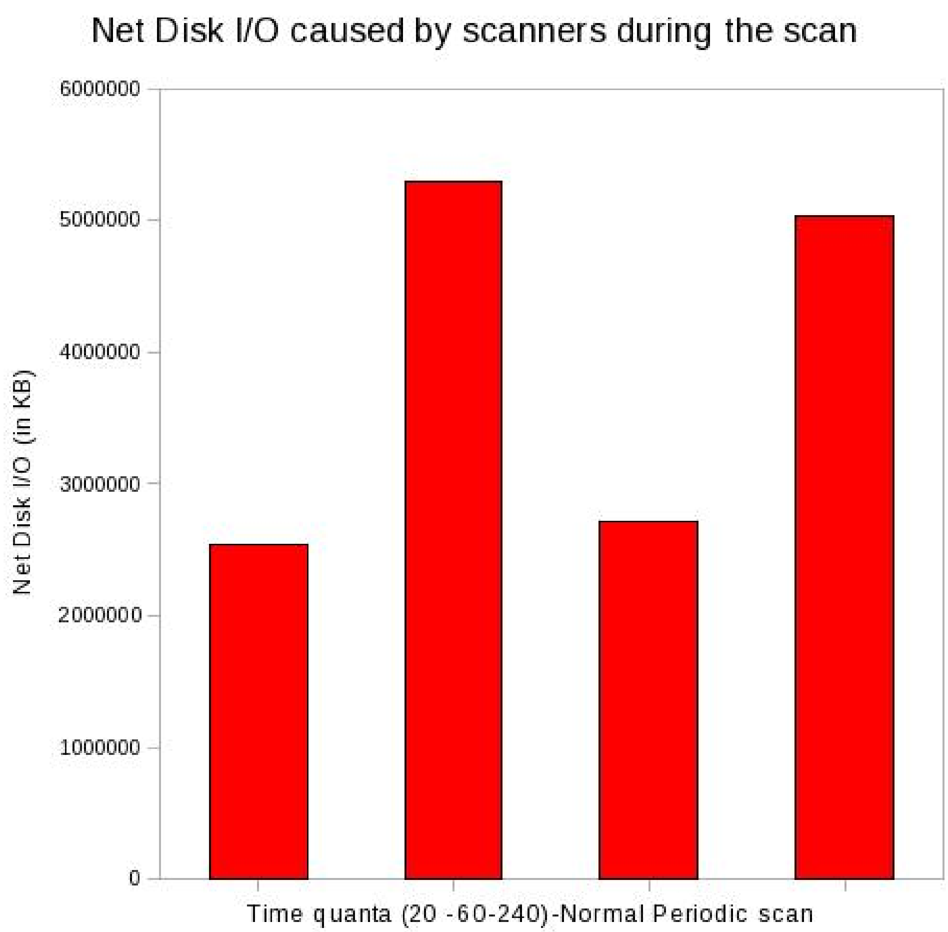
Net Disk I/O=Net Disk I/O of (Avast Individual scan+ Vascan Individual scan + Clamscan Individual scan) = 5034655.4 KB

Redundant Data = (5.04-2.25)GB = 2.79GB

2)
Net Disk I/O caused by scanners when they are scheduled according to a round robin scheme using **Reducer**.
Scans were conducted for three different time quanta values.
Time quanta t =20 seconds
Time quanta t =60 seconds
Time quanta t=240 seconds

Net Disk I/O for t=20 sec -1270700.5 KB *2 = 2541401, Redundant Data = (2.54 - 2.25)GB=0.29GB
Net Disk I/O for t=60 sec - 2649501.8 KB *2 = 5299003.6, Redundant Data = (5.3 – 2.25)GB =3.05GB
Net Disk I/O for t=240 sec -1359638 KB *2 = 2719276,Redundant Data =(2.72-2.25)GB= 0.47 GB

One can observe significant reduction in redundant data ,drops from 2.79 GB to 0.29GB (for time quanta t= 20) and 0.47 GB ,(time quanta t=240 seconds)



Net Disk I/O caused by scanners during the scan
6000000
5000000
4000000
3000000
2000000
1000000
0
Net Disk I/O (in KB)
Time quanta (20 -60-240)-Normal Periodic scan

5. Application : Fast virus checking

Anti-virus software,anti spywares ,backup programs have become an integral part of one's life.These are the essential utilities which one must have on his system ,to keep his system safe and secure and avoid system failures and downtime.One concern regarding these applications ,are that they need to scan the system as part of their functioning.Consequently,these tasks are disk I/O intensive.People often experience slow downs when these applications are running, firstly because these tasks are Disk I/O intensive and secondly as they follow fig1 below , some aware people follow fig2 .But,as these tasks need to be performed regularly,they often become a problem for the people and they tend to avoid it.With the help of Reducer ,we can make the process of scanning more efficient ,by reducing redundant disk I/O activity ,reducing downtime (time when scans are being conducted by these applications.)

Principle:

Reducer works on the principle ,first application reads file blocks from the disk and later application which also need access to these file blocks, access them from buffer cache,rather than reading from the disk. As accessing file blocks from buffer cache ,is much faster than accessing file blocks from the disk ,applications are able to read large no. of file blocks in less time and their scanning efficiency improves as compared to when they were reading file blocks from the disk.

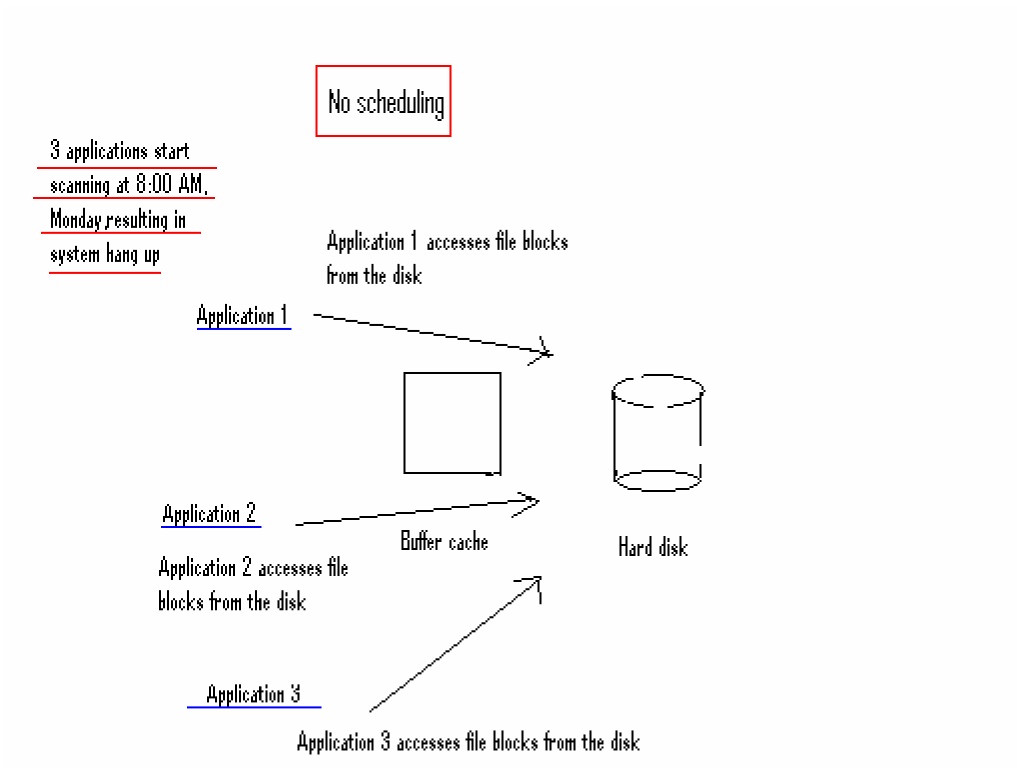

No scheduling
3 applications start scanning at 8:00 AM, Monday resulting in system hang up
Application 1 accesses file blocks from the disk
Application 1
Application 2
Buffer cache
Hard disk
Application 2 accesses file blocks from the disk
Application 3
Application 3 accesses file blocks from the disk

Fig 1

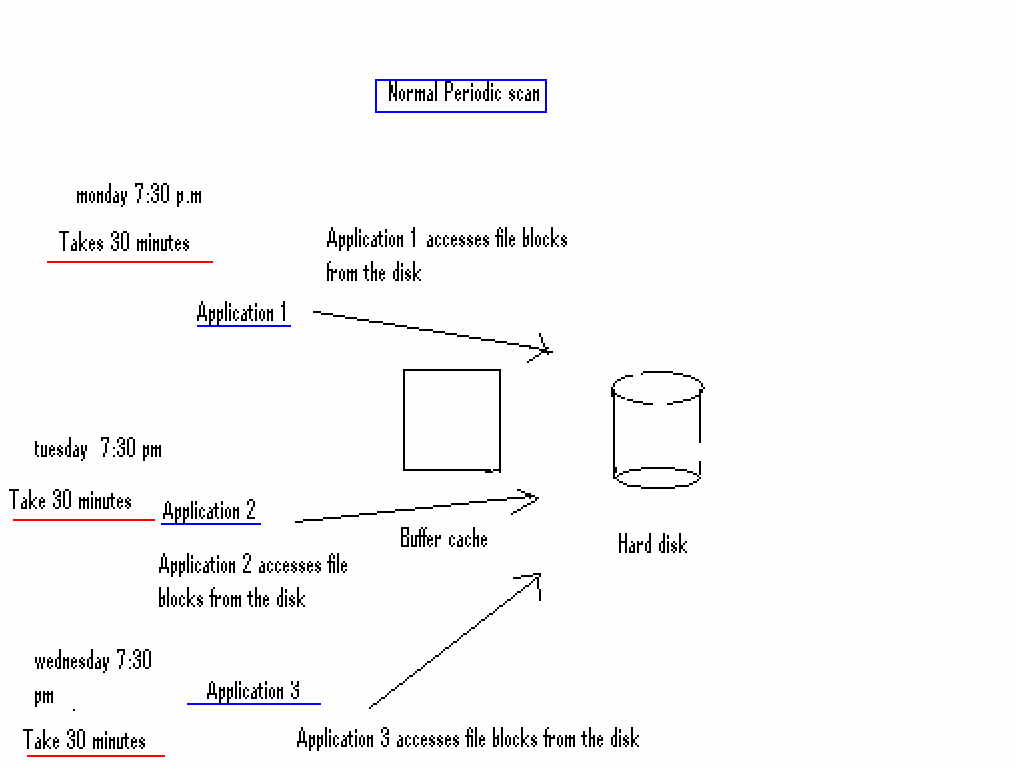

Normal Periodic scan
monday 7:30 p.m
Takes 30 minutes
Application 1 accesses file blocks from the disk
Application 1
tuesday 7:30 pm
Take 30 minutes
Application 2
Buffer cache
Hard disk
Application 2 accesses file blocks from the disk
wednesday 7:30 pm
Application 3
Take 30 minutes
Application 3 accesses file blocks from the disk

Fig 2

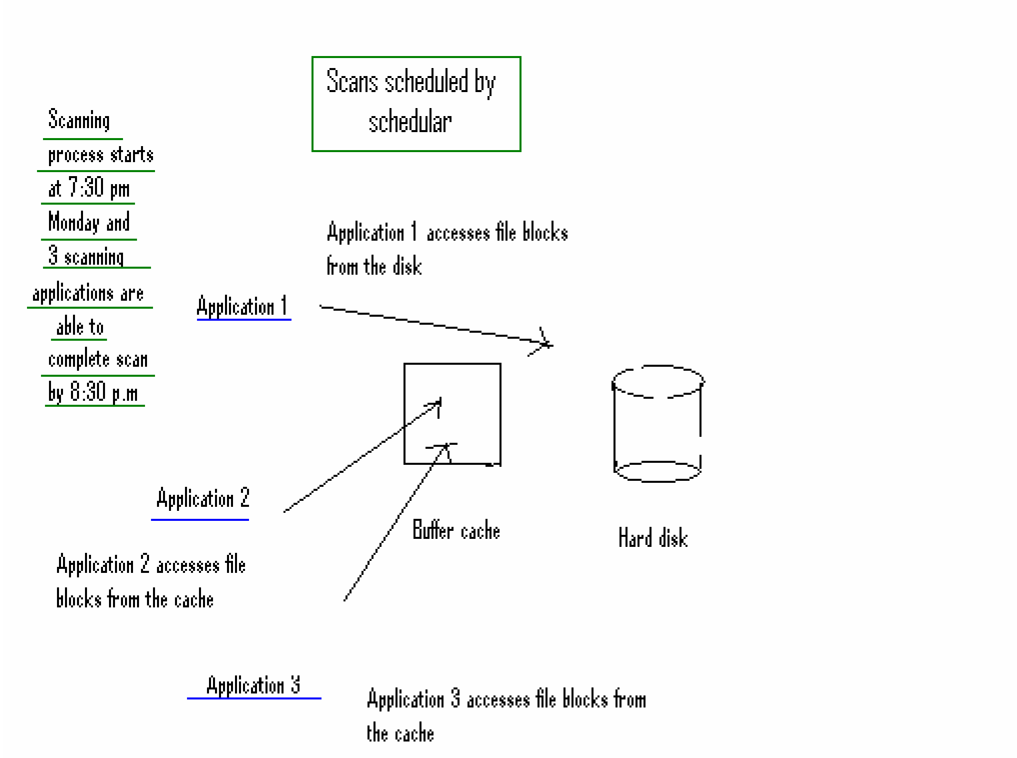


Fig 3

Fig 3 illustrates,how we can improve scanning efficiency using Reducer.
As one can observe,one will be able to complete the three scans in a single day in approximately half the time.

Earlier, we conducted experiment to schedule the scans of three different virus scanners, Avast,Vascan,ClamAV using round robin scheme with different time quanta settings 20, 60 ,240 seconds using Reducer and measured Disk I/O activity.
Data sheets and scan plots can be referenced from previous section.

Figures below show scanning time period computations (how much time scanning activity takes) for different time quanta settings,
Using our computations we will show,how it is possible to achieve scenario described in figure 3.

NOTE: Application should make use of **same** file scanning algorithm for proper functioning of reducer.
Here ,we considered **three virus scanner** which use same file scanning algorithm.

5.1 Time spent in scanning -

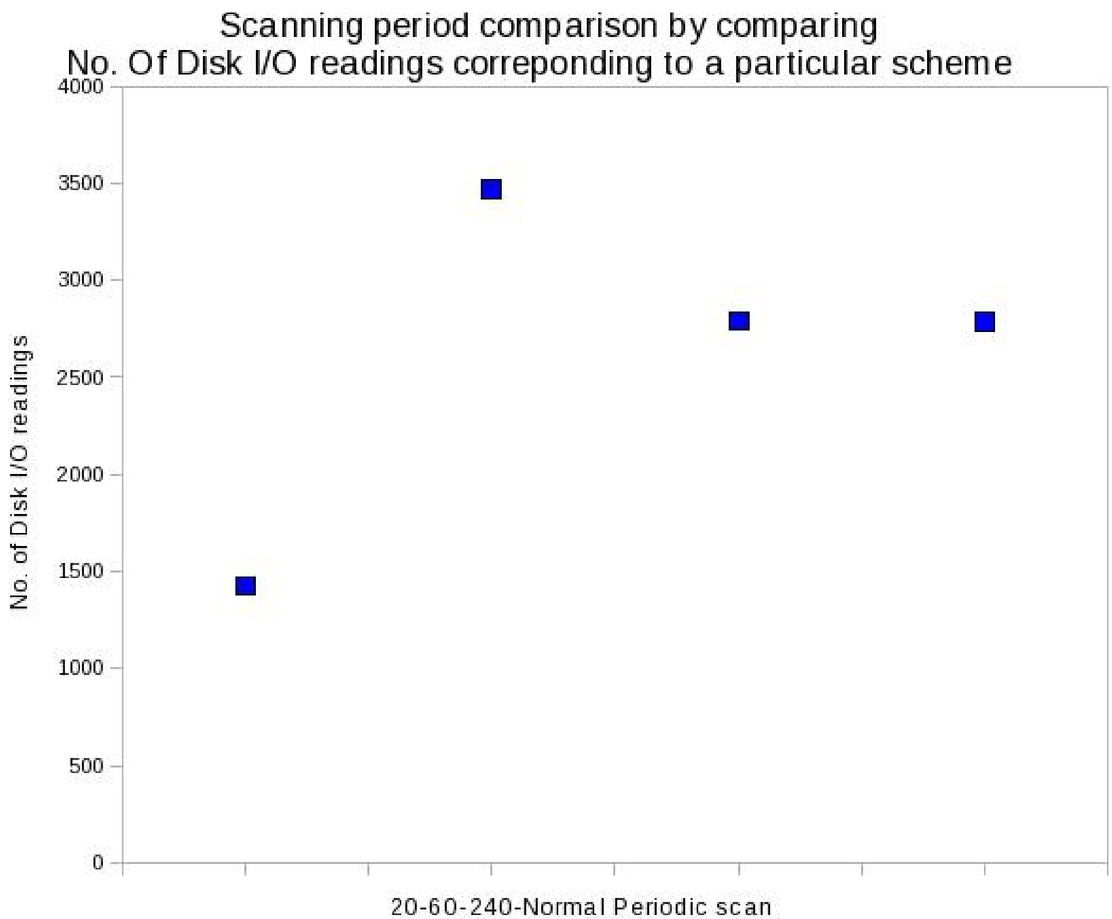
We can use the No. of Disk I/O readings (those corresponding to disk reads by scanners)in a scan sheet as a measure of scanning period (defined between non idle reading slots (idle slots correspond to period when kr/s was zero ,scan was about to begin or has ended.))When iostat will trap non-null disk reads (those which are done by scanner while scanning),it implies scan is still in progress so larger are no. of disk reads ,larger is the scanning period.

So,we conclude that No. of Disk I/O readings in a scan sheet corresponding to a particular scheme are directly proportional to time spent in scanning ,when that particular scheme was employed for scheduling scans. Suppose ,if round robin scheme with time quanta t=20 has less no. of Disk I/O readings (corresponding to disk reads by scanners)in its scan sheet as compared to time quanta t=60 ,we can safely,say that former scheme takes less time to scan as compared to latter .

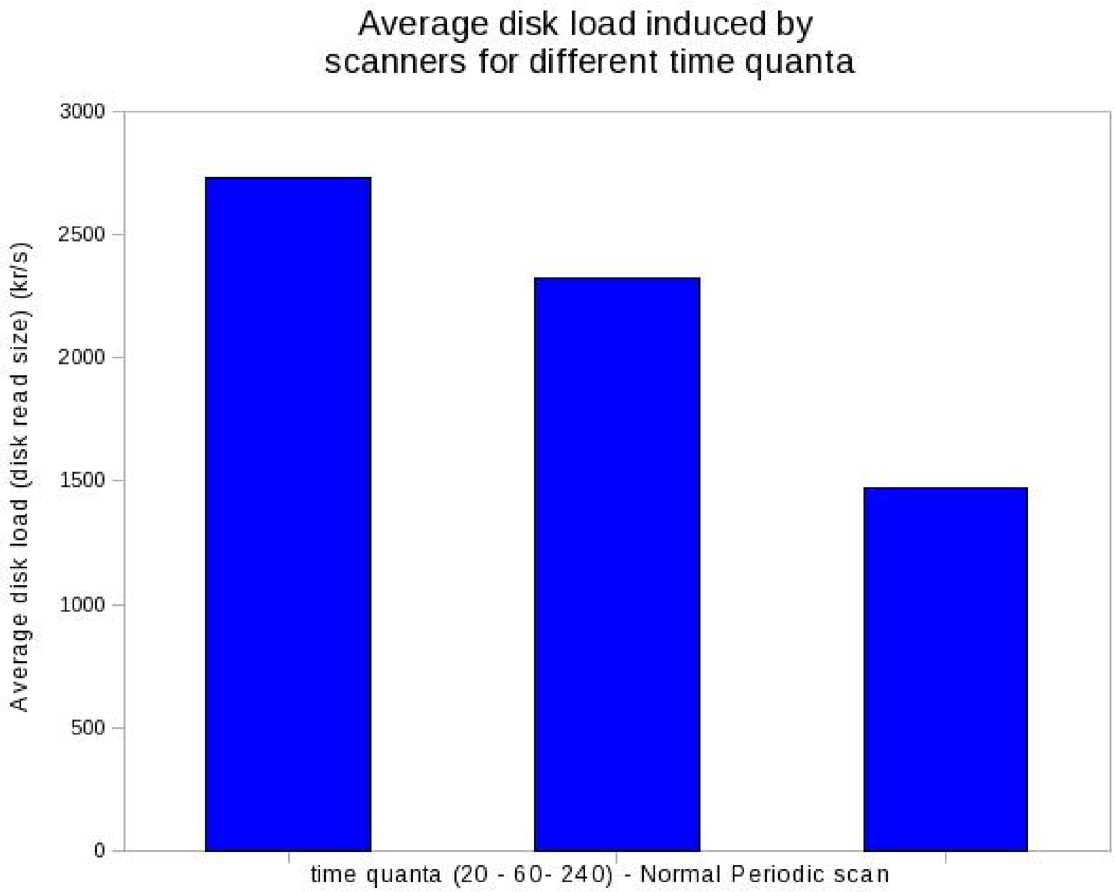
No. of Disk I/O readings obtained for different schemes -
Normal Periodic scan -
1)Avast - (764-20)=744
2)Vascan - (548-53)=495
3)ClamAV - (1514-5)=1509

Time quanta t=20 - (1472-44)=1428
Time quanta t=60 - (3584-116)=3468
Time quanta t=240 – (3176-383)=2793

As ,can be observed ,**scanning time reduces to half when time quanta is set to 20 seconds**.
So ,we can achieve our scenario (fig 3) if time quanta is set to 20 seconds.
One can schedule the scans using reducer ,on Monday 7:30 PM ,and can experience performance gain.


Scanning period comparison by comparing
No. Of Disk I/O readings correponding to a particular scheme
4000
3500
3000
2500
2000
1500
1000
500
0
No. of Disk I/O readings
20-60-240-Normal Periodic scan

5.2.



Above figure shows disk load which different scheduling schemes induce on the disk(Average disk read size for different time quanta).One can observe that average size of disk read done ,when time quanta is t =20 seconds is more as compared to 60 and 240 seconds and time quanta 240 seconds has least average disk read size of 1471.47 KB/s.This indicates system will be bit slower when scans are scheduled with time quantum settings of 20 seconds as to when value is set to 60 and 240 respectively.

There is one more scenario possible which can be utilized by user depending on his choice -
On selecting time quanta t =240 seconds ,we achieve a drop in average disk read size done by scanners during the scan,it reduces to half, but net scanning time remains same.User can engage in a lightweight activity during the scan without any system slowdowns,an option he didn't have earlier.

6. CONCLUSION:

Finally we conclude that we that we can reduce Disk I /O redundancy using reducer.It can be used for fast virus checking, as we have shown earlier that we were able to complete the full scan by three virus scanners in half the time for time quantum setting of t = 20 setting as compared to a normal periodic time.Thus it helps in reducing downtime.We also looked at a scenario,in which user has a possibility to do some work,when scans are going on,case corresponding to t =240 seconds.

In future ,we aim to study integration of Reducer at kernel level .We also aim to study impact of disk read size,categorize processes which can be executed while scans are going on, without any slowdowns.Finally,we also plan to do study of a practical case when virus scanner ,a security application ,like Microsoft Soft anti spyware and a backup program are scheduled using Reducer and do performance analysis.

Reference:

Operating System Concepts:
Seventh Edition

Avi Silberschatz
Peter Baer Galvin
Greg Gagne