Helsinki Institute of Physics, HIP-TEK project
7.10.2013

Joni Anttalainen

SICX and dCache file I/O benchmark

Contents

1 Terminology

RAIC = Redundant Array of Inexpensive Clouds
SL6 = Scientific Linux 6
UI = User Interface
LHC = Large Hardon Collider
SRM = Storage Resource Manager
JVM = Java Virtual Machine
JRE = Java Runtime Environment
HTTP = Hypertext Transfer Protocol
HTTPS = Hypertext Transfer Protocol Secure
NFS = Network File System
API = Application Programming interface
pNFS = Parallel Network File System

2 Introduction

This document describes how to install dCache[3] and SICX distributed data storage systems and introduces two I/O performance benchmarks on these systems. The first benchmark is performed without using security features of SICX or dCache. For the second benchmark the security features are turned on and the same tests will be performed.

The aim of this work is to compare the duration of file write and read operations with different sized files on SICX and dCache[3] systems both with and without security features.

SICX supports WebDAV[23] protocol to allow reading and writing data to and from it. dCaches data can be accessed with several protocols, but this benchmark compares the speed of dCache's NFSv4.1[16] implementation to SICX's performance.

Benchmarking setup section presents instructions of how to install and run dCache and SICX with a suitable configuration for this benchmark. Also the script and the tools used to perform the actual benchmark are presented.

2.1 dCache

dCache is a data storage middle-ware that was created to fulfill the data storage needs of the LHC[4] experiment at CERN[5]. It is a highly scalable data storage system that allows storage and exhange of large amounts of data rapidly. DCache provides several standardized protocols for storing and retrieval of data from the system.

Different types of storage systems, e.g hard disk drives or tape storage systems, can be attached to a dCache installation. These storage systems associated with a certain dCache system can be heterogenous and still be accessed with stardardized protocols via dCaches interfaces. DCache also enables its data repositories to appear under a single virtual filesystem tree and it supports standard tape storage systems.[2]

A dCache system includes one or more dCache servers. A dCache server runs one or more so called domains. DCache domains are Java Virtual Machines[11] run within dCache that host at leat one service. Services are abstractions used by dCache to represent atomic units that can be added to a domain and provide some beneficial features and . Services are usually implemented with one or several cells. For example a pool is a type of dCache cell that provides physical data storage services.

dCache is a highly configurable system. Parts that often need to be configured in dCache are load balancing and file replication.

Clients can mount the dCaches file system tree with NFS to achieve a uniform representation of dCache's contents. To allow usage direct I/O -operations, such as *cp* linux command, NFS 4.1[16] door must be opened on dCache server.[1]

2.2 SICX

SICX is a prototype system developed in the HIP-TEK[7] project by Helsinki Institute of Physics[6]. It implements the redundant array of inexpensive clouds (RAIC) consept. RAIC[8] denotes that with SICX it is possible to combine multiple cloud storage services to achieve a large inexpensive storage platform with high availability. SICX encrypts a file to be stored, stripes it to several parts and stores the parts on different clouds so no cloud service provider has access even to the complete encrypted file.

Files are stored with redundancy so unsatisfying cloud service providers can be dropped off instantly at any time without losing data. This decreases the problem of vendor lock-in[9].

SICX system consists of four parts. The meta server stores the meta data, the cloud array stores the actual encrypted data stripes, Hydra stores the parts of the encryption key and the client program performs the crypting, striping and file transfers. The encryption key is stored in several parts with Hydra[18] so to have access to the complete encryption key one must have access to most of the Hyra servers. It is possible to configure the key part count and how many Hydra servers are required to construct the key. Hydra's source code is freely available on GitHub[19]. Client side source code is not public, but meta servers source code can be obtained from GitHub[20].

SICX has a built-in graphical user interface that is shown in figure 1, but it also possible to use SICX with WebDAV[23] protocol from a web browser or command line.

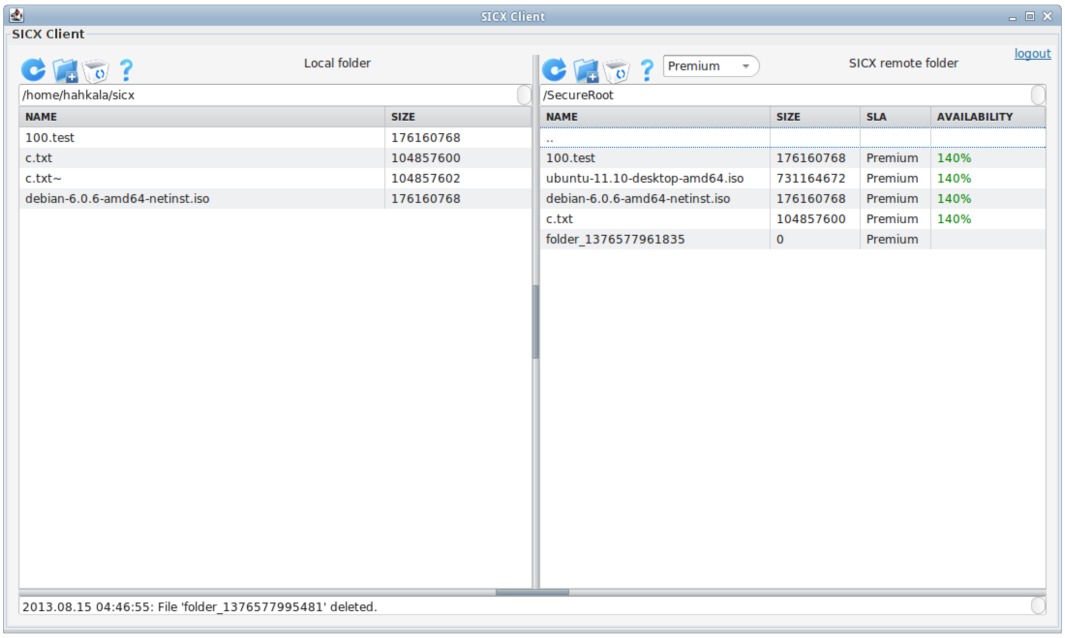

Figure 1: SICX's graphical user interface in use on Ubuntu[10].

3 Benchmarking setup

Both systems will be installed on a single server node and benchmarks are run from the same machine. Files of different sizes will be both written to and read from SICX, dCache and a local directory. In addition to dCache and SICX also the local directory I/O speed is measured to get a reference value. Each read and write operation will be performed 100 times on each system and an arithmetic average is calculated for each operation. Files of sizes 1 MB, 10 MB, 100 MB, 500 MB, 1 GB and 2 GB will be tested. First both systems will be benchmarked without security features and then with security features on to see how much security slows down the I/O speed.

The test machine will run Scientific Linux 6[15] with Intel©Xeon©dual core CPU running at 2.00 GHz, 4 GB of memory and 50 GB's of disk space.

SICX system uses WebDAV[23] server. Davfs2[21] linux driver is used in the benchmark setup to mount the server in a way that allows client to access the servers data similarly as if it was a local hard drive. SICX has a built-in debug option to store the data on the machine where client runs instead of a cloud array. This option is used to get a similar environment as dCache has.

dCache's file tree will be mounted to local file system hiearchy with NFSv4.1 protocol[16] in a similar way as SICX, but no additional driver is required in addition to those included in SL6[15] distribution.

3.1 Setting up a dCache

3.1.1 Configuration files

Configuration of dCache is done by defining configuration parameters in */etc/dcache/dcache.conf* file. After installation the configurable parameters and their default values can be found from */usr/share/dcache/defaults*. To modify a configuration parameters value the parameter must be redefined in *dcache.conf* file with desired value. The default values must not be altered.

Domains and services run by a dCache server are defined in a layout file. Layout files are stored in */etc/dcache/layouts* directory and the used layout file can be defined in *dcache.conf* -file. The default layout file is */etc/dcache/layouts/single.conf*.[1]

3.1.2 Installing prequisities

In order to install dCache JRE[11] (Java Runtime Environment) 1.7 has to be installed on the virtual machine. OpenJDK[12] Runtime Environment 7 is used and it is included in SL6.

PostgreSQL[13] must be installed and running as it is used to store dCaches state information. Recommended version is 9.2, but required version is 8.3. Version 8.4.13 is used as it is included in the SL6's *sl6-os* repository. To make the installation easier the PostgreSQL will be configured to allow local users to connect to it without a password.

Basic installation of PostgreSQL can be done on SL6 with following command sequence:

```
[root] # yum -y install postgresql-server
[root] # /etc/rc.d/init.d/postgresql initdb
[root] # /etc/rc.d/init.d/postgresql start
[root] # chkconfig postgresql on
```

Now PostgreSQL Server is installed and running on the system.

3.1.3 Configuring PostgreSQL

Next the PostgreSQL database will be configured to serve dCache's needs. To make the installation easier the local access to database can be allowed without a password. This can be established by modifiyng the file */var/TODO*. The values of *METHOD* column have to be modifed from *ident* to *trust*. After the modifications the lines should look equivilent to this[1, p. 196]:

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
local   all         all                               trust
host    all         all         127.0.0.1/32          trust
host    all         all         ::1/128               trust
```

PostgreSQL server needs to be restarted for changes to take effect:

```
[root] # /etc/rc.d/init.d/postgresql restart
```

Then it is required to create a user and a database for Chimera. Chimera is a library used by dCache that stores the metadata of the files[14]. A password in not to required for the users we create so enter can be pressed when prompting for a password. Create the Chimera user and database:

```
[root] # createdb -U postgres chimera
[root] # createuser -U postgres --no-superuser --no-createrole
--createdb --pwprompt chimera
```

Several dCaches management components use PostgreSQL user *srmdcache* to access dCache and store their state information. This user has to be created:

```
[root] # createuser -U postgres --no-superuser --no-createrole
--createdb --pwprompt srmdcache
```

A database *dcache*, that is used to store the state information, and a database *billing*, for the billing plots, have to be created:

```
[root] # createdb -U srmdcache dcache
[root] # createdb -O srmdcache -U postgres billing
```

One has to make sure that each one of created databases has *plpgsql* procedural language installed. Installed languages of a database can be listed with command:

```
# createlang -l database_name
```

If *plpgsql* is not listed it can be installed with command:

```
# createlang plpgsql database_name
```

3.1.4 Installing dCache

When the prequisities have been installed and PostgreSQL server is up and running the dCache software may be installed. Version 2.6.4 of dCache is used.

The installation package may be obtained in *rpm* format from the dCache home page[3]. When the rpm package has been downloaded it can be installed with command:

```
[root] # rpm -ivh dcache-2.6.4-1.noarch.rpm
```

Now both PostgreSQL Server and dCache are installed and the databases may be updated:

```
[root] # dcache database update
```

Before starting dCache gPlazma's[1, p. 6] configuration file must be replaced with an empty file:

```
[root] # mv /etc/dcache/gplazma.conf /etc/dcache/gplazma.conf.bak
[root] # touch /etc/dcache/gplazma.conf
```

As dCache won't be attached to a tape system he following lines need to be added to */etc/dcache/dcache.conf* file:

```
DefaultRetentionPolicy=REPLICA
DefaultAccessLatency=ONLINE
```

Finally dCache can be started:

```
[root] # dcache start
```

3.1.5 Creating a basic layout file

By default the layout that dCache uses is defined in */etc/dcache/layouts/single.conf*. Custom layout file */etc/dcache/layouts/bm.conf* will be used. dCache can be configured to use this layout file by adding the following line to */etc/dcache/dcache.conf*:

```
dcache.layout=bm
```

The base for the new layout file can be created by copying the contents of *single.conf* to a new file *bm.conf*.

3.1.6 Enabling use of the administration interface

DCache comes with an administration interface which can be accessed via ssh1 and ssh2[25] procols from the command line. Command line usage of admin user interface with ssh1 will be enabled. ssh1 is insecure, but the intefrace will be accessed only from within the dCache server so safety doesn't need to be considered.

Before using admin CLI with ssh1 we need to define ssh version to be ssh1 in the *dcache.conf* -file by adding this line to the end of the file:

```
sshVersion=ssh1
```

Then we need to restart dCache.

```
[root] # dcache restart
```

Now the following command is used on the server to access and start the admin interface:

```
# ssh -c blowfish -p 22223 -l admin 127.0.0.1
```

The initial password is *dickerelch*. Now we can execute administrator commands through the CLI.

3.1.7 Web interface

By default the dCache's web user interface service *httpd* is running and it can be accessed by typing

```
<dcache server address>:2288/
```

to a locally running web browser. The web interface is an intuitive way to monitor dCache.

3.1.8 Creating and configuring a pool

Now a pool of size 20 GB will be created. Type the following commands on the dCache server:

```
[root] # dcache pool create --size=20000000000  --meta=db
--lfs=precious   /pools/pool1 pool1 dCacheDomain
```

Pool is added to the *bm* layout file, but dCache has to be restarted to get it running. A pool has been created, but the default gap size is 4 GB so 4 GB:s of the pools space can't be used. If the available space in a pool is equal or less than the gap the pool is considered full. To maximize performace the gap size should be set to the size of the smallest files that are frequently moved in and out of dCache[1, p. 61]. The gap can be adjusted to one giga byte by typing the following commands to dCaches CLI with admin account:

```
(pool1@dCacheDomain) admin > cd pool1@dCacheDomain
(pool1@dCacheDomain) admin > set gap 1G
Gap set to 1073741824
(pool1@dCacheDomain) admin > save
```

Gap size has now been set to 1 giga byte in *pool1*.

3.1.9 Configuring NFSv4.1 door

Mounting of Chimera's root will be allowed locally. Add the following lines to servers */etc/exports* -file:

```
/ localhost(rw,sync)
/data localhost(rw,sync)
```

dCache has to be restarted for changes to take effect. Next the root will be mounted locally on the dCache server machine and the root of the Chimera's namespace will be created:

```
[root] # mkdir /mnt/dcache
[root] # mount localhost:/ /mnt/dcache
[root] # mkdir -p /mnt/dcache/data
[root] # mkdir /mnt/dcache/data/world-writable
[root] # chmod  777 /mnt/dcache/data/world-writable
[root] # umount localhost:/
```

To allow I/O actions to dCache server via NFSv4.1 the NFS v4.1 door service needs to be defined in the layout file. The following line can simply be added to the layout file below the only domain (dCacheDomain):

```
[dCacheDomain/nfsv41]
```

nfs.domain value has to be added to *dcache.conf*:

```
nfs.domain=CERN.CH
```

If a separate client machine is used the *nfs.domain* value has to match the *Domain* -value in clients machines */etc/idmapd.conf* -file.

After dCache is restarted NFSv4.1 door should be running on port 2049.

3.1.10 Mounting dCaches data with NFS

Mounting can be done on the local machine in root mode with *mount* command. The following mount command is used:

```
[root] # mount -o intr,minorversion=1 localhost:/data /mnt/dcache
```

Parameter *minorversion=1* defines that NFSv4.1 mount is performed. The option *intr* has to be defined when mounting dCache in order for I/O to work. It defines that client can interrupt NFS requests in case server cannot be reached. To check if dCache was mounted correctly the following command can be used:

```
# mount | grep nfs
```

The output should look eqvivalent to:

```
localhost:/data on /mnt/dcache type nfs
(rw,intr,minorversion=1,vers=4,addr=127.0.0.1,clientaddr=127.0.0.1)
```

Now the contents of dCache should appear in */mnt/dcache/world-writable* directory.

Now a large file should be copied to dCache to test that everything works as expected:

```
cp /some_large_file.img /mnt/dcache/world-writable/
```

Moving big files over NFS to dCache may cause an I/O-error because of dCaches caching. If this happens it can be fixed in the CLI:

```
(local) admin > cd pool1@dCacheDomain
(pool1@dCacheDomain) admin > csm set policy -onwrite=off -ontransfer=on
-enforcecrc=on
(pool1@dCacheDomain) admin > save
(pool1@dCacheDomain) admin > ..
(local) admin > logoff
```

Now when the transfer is finished the file file with correct size should appear in pool1. It may take a while for the file to appear in pool after transfer is finished. To list pool1:s contents the following command can be typed to dCache's CLI:

```
(pool1@dCacheDomain) admin > rep ls
000...2EA <C-------X--L(0)[0]> 726970368 si={<Unknown>:<Unknown>}
```

The commnand lists all files in dCache's *pool1* and gives information about their properties.

3.1.11 Enabling security

dCache uses Kerberos[27] protocol for secure file access. This section covers how to enable security in dCache for the second benchmark.

The local key distribition center (KDC) server can found from */etc/krb5.conf* -file. After you know your KDC and kerberos realm add the following lines to */etc/dcache/dcache.conf* -file.

```
nfs.rpcsec_gss=true
kerberos.realm=CERN.CH
kerberos.jaas.config=/etc/dcache/gss.conf
kerberos.key-distribution-center-list=CERNDC.CERN.CH
```

gPlazma[1, p. 89] service also needs to be defined in our *bm* layout file:

```
[dCacheDomain/gplazma]
gplazma.authzdb.file=/mnt/authzdb.txt
```

The gPlazma's authorization process needs to be configured in */etc/dcache/gplazma.conf*:

```
map requisite krb5
map requisite authzdb
```

Then the file */home/joni/authzdb.txt* needs to be created:

```
version 2.1
authorize nfs read-write 1001 100 / / /
```

Domain = CERN.CH has to be added to /etc/idmapd.conf.

Both client and server have to have a proper kerberos configuration. Configuration file can be found from */etc/krb5.conf*. Both machines also have to have the following Kerberos principal in their key tab:

```
nfs/machine_address@CERN.CH
```

This principal has to be imported to the server and all machines accessing it. In CERN this can be done with following command:

```
[root] # cern-get-keytab --service nfs --isolate --keytab /etc/dcache.keytab
Keytab file saved: /etc/dcache.keytab
```

It might be required to change the file access permissions of the created keytab file. After the principal has been added one should make sure that the new principal is included in the *dcache.keytab*. The principals in *dcache.keytab* can be listed with command:

```
[root] # klist -e -k /etc/krb5.keytab
Keytab name: FILE:/etc/krb5.keytab
   ......
   5 nfs/bclient.cern.ch@CERN.CH (arcfour-hmac)
   5 nfs/bclient.cern.ch@CERN.CH (aes128-cts-hmac-sha1-96)
   5 nfs/bclient.cern.ch@CERN.CH (aes256-cts-hmac-sha1-96)
```

Then Javas security module has to be defined in file */etc/dcache/gss.conf*:

```
com.sun.security.jgss.accept {
com.sun.security.auth.module.Krb5LoginModule required
doNotPrompt=true
useKeyTab=true
keyTab="${/}etc${/}krb5.keytab"
```

```
debug=false
storeKey=true
principal="nfs/hiptek-bm.cern.ch@CERN.CH";
};
```

The Linux's exports file needs to be modified to use Kerberos:

```
/ localhost(rw,sec=krb5,sync)
/data localhost(rw,sec=krb5,sync)
```

To allow secure nfs mounting */etc/sysconfig/nfs* has to be modified:

```
SECURE_NFS="yes"
```

It is required to update the */etc/exports* -file:

```
/ localhost(rw,sec=krb5p)
/data localhost(rw,sec=krb5p) client_ip(rw,sec=krb5p)
```

Next the system and dCache have to be restarted for changes to take effect. To mount dCaches contents with Kerberos the following command is used:

```
[root] # sudo mount -o intr,sync,minorversion=1,sec=krb5p hiptek-bm.cern.ch:/dat
```

Then the *gPlazma.conf* -file needs to be configured:

```
map requisite krb5
map optional authzdb
session requisite authzdb
```

The default file used by the authzdb plugin can be defined by adding the following line to the used layout file below gPlazma:

```
gplazma.authzdb.file=/mnt/authzdb.txt
```

The mapping from user name to user id and group id has to be done in authzdb file:

```
version 2.1
authorize janttala read-write 500 500 / /data /
```

Now dCache can be mounted with command:

```
[root] # mount -o intr,minorversion=1,sec=krb5p hiptek-bm.cern.ch:/data /mnt/dca
```

If dCache raises an IO-Error when transferring data the client is propably mapped to a wrong ip address. This can be fixed by adding an entry to */etc/hosts* -file. Login failure is printed to log when dCache is mounted, but this can be ignored since there is no actual failure.

3.1.12 Allowing access to dCache's contents using WebDAV

It is also possible to access dCaches contents with WebDAV[**?**] protocol.To enable use of WebDAV the following lines have to be added below a domain in the used layout file:

```
[dCacheDomain/webdav]
webdavAnonymousAccess=FULL
```

The dCacheDomain has to be restarted to start the WebDAV service. After the restart the files can be accessed from a local web browser with this address *127.0.0.1:2880*.

3.2 Setting up SICX

3.2.1 Installing prequisities

Maven[17] package has to be installed. To download and extract the 3.0.5 version of Maven the following command sequence can be used:

```
# wget http://mirror.switch.ch/mirror/apache/dist/
maven/maven-3/3.0.5/binaries/apache-maven-3.0.5-bin.tar.gz
[root] # tar -xf apache-maven-3.0.5-bin.tar.gz
```

Then the following lines need to be to users *~/.bashrc* -file:

```
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk.x86_64/
export M2_HOME=<path_to_maven_dir>
export M2=$M2_HOME/bin
export PATH=$M2:$PATH
```

To take the new variables into use the *~/.bashrc* file needs to be re-sourced:

```
# source ~/.bashrc
```

Next it is required to install *davfs2*[21] linux driver to mount the WebDAV server. *davfs2* requires *neon-devel*[24] package to be installed. It can be installed with *yum* command:

```
[root] # yum install neon-devel
```

Then the *davfs2* source files are downloaded from the *davfs2* home page[22]. To install *davfs2* execute:

```
# ./configure
# make
# make install
[root] # groupadd --system davfs2
[root] # useradd davfs2 --gid davfs2 --shell /bin/false
--system --comment "davfs2 system user"
```

To allow the used user account to mount SICX it needs to be added to *davfs2* group:

```
[root] # usermod -a -G davfs2 <our_user_account_name>
```

Next the *davfs2*'s configuration parameters need to be configured. They are located in two directories. First the values from file */var/cache/davfs2/davfs2.conf* are read and then if some parameters are defined in users */.davfs2/davfs2.conf* file the first values are overriden by these values. Here we configure the parameters in the user specific configuration file */.davfs2/davfs2.conf*:

```
delay_upload    0
read_timeout    3000
ask_auth        0
```

The *delay_upload* parameter defines the time in seconds before DavFS starts the actual transfer of a modified file to the WebDAV server. Value 0 is used so the transfer starts instantly. The *ask_auth* defines that the user isn't prompted for authentication when perfoming a WebDAV mount. The large value in *read_timeout* is defined to prevent read timeouts.

3.2.2 Installing SICX client

SICX's client code is not yet available freely. These instructions cover SICX client installation from *hhydra_integration* branches files. Instructions describe how to install and run SICX wihout graphical user interface or Hydra security to establish a minimal installation for benchmarking.

Modify the source file *src/main/java/fi/hip/sicx/vaadin/SICXUploader.java* parameter *use_encryption* from *true* to *false* to use SICX without Hydra.

For downloads from SICX to work without Hydra another boolean value has to be modified. Change *src/main/java/fi/hip/sicx/webdav/FileResource.java* files *true* boolean to *false* on this row:

```
mc.downloadFile(meta, man, target, file, true, null);
```

Also the <sign> block of *pom.xml* file in the root of clients sources needs to be removed to prevent double signing when packaging the client.

Move to the root folder of SICX clients source files. If client is not installed in users own directory *chmod*[26] linux command needs to be used to adjust file access permissions. The client is packaged with Maven:

```
# cd <client_directory>
# chmod 777 .
# mvn package
```

Used certificates need to be copied to *~/.sicx_data* folder:

```
# mkdir ~/.sicx_data
# cd <client_dir>
# cp src/test/cert/trusted_client.cert ~/.sicx_data
# openssl rsa -in src/test/cert/trusted_client.priv
> ~/.sicx_data/trusted_client.priv
```

Enter a password that will be used later when creating the *~/.sicx* file.

3.2.3 Installing SICX meta server

The source code can be downloaded from GitHub:

```
https://github.com/jhahkala/meta
```

The code can be packaged with the following command (it might be necessary to modify meta directories permissions similarly as in client installation):

```
# cd <meta_directory>
# mvn -DskipTests package
```

A *~/.sicx* file has to be created to users home directory with following contents:

```
sslKey=/home/<user_name>/.sicx_data/trusted_client.priv
sslCertFile=/home/<user_name>/.sicx_data/trusted_client.cert
trustStoreDir=/home/joni/.sicx_data/truststore
metaService=https\://localhost\:40669/MetaService
sslKeyPasswd=<your_password>
folder.local=/home/joni/sicx
```

It is also necessary to change port to match 40669 in *<sicx_meta_dir>/src/test/meta-purge.conf* -file:

```
port=40669
```

3.2.4 Starting SICX

First the meta server is started in one terminal window:

```
# cd <path_to_meta_server_root>
# java -Djavax.net.debug=all -jar target/meta.jar src/test/meta-purge.conf
```

Next a user is added to SICX's meta server using another terminal window. This step has to be repeated every time meta server is started when following these instructions. A test certificate is provided in SICX's client files. This can be used or new certificates may be generated. To find out test certificates user name *openssl* linux command can be used:

```
# openssl x509 -in ~/.sicx_data/trusted_client.cert -text -noout | less
```

You should see output similar to:

C=FI, DC=slcs, O=HIP, OU=Tech, CN=<some_name>

To add this user to meta server the following commands are executed (meta server has to be running):

```
# cd <meta_directory>
# java -cp target/meta.jar org.joni.test.meta.client.MetaClient
-c src/test/meta-client-trusted.conf addUser --name "CN=<some_name>,OU=Tech,O=HI
--root SecureRoot --sla Open
```

To start the client move to the directory *<sicx_client_directory>/target/jnlp*. Then to start the client without GUI execute:

```
# java -cp "lib/*" fi.hip.sicx.vaadin.Launcher headless
```

If a *ClassNotFound* exception is raised it can be fixed by adding *slf4j* jar to SICX clients *lib* directory. First the *slf4j* package needs to be downloaded from the official web site. Then the package can be extracted and *slf4j-nop* jar may be copied to clients *lib* directory. Here version 1.7.5 is used:

```
# wget http://www.slf4j.org/dist/slf4j-1.7.5.tar.gz
# tar -xf slf4j-1.7.5.tar.gz
# cd slf4j-1.7.5
# cp slf4j-nop-1.7.5.jar <client_dir>/target/jnlp/lib/
# cd ..
# rm -r slf4j*
```

Alternately to start the client with graphical user interface this command can be used in the *jnlp* directory:

```
# javaws sicx.jnlp
```

To allow mounting of SICX's contents without root rights the following line needs to be added to */etc/fstab* -file:

```
http://localhost:8081/webdav/SecureRoot   /mnt/sicx davfs rw,user,noauto 0 0
```

Now SICX can be mounted:

```
# mount /mnt/sicx
```

If you encounter a null pointer exception logging out, logging back, restarting meta and client and re-adding the user should solve the problem.

3.2.5 Enabling security

In the second benchmark SICX and dCache will be tested with their security features on. This section describes how to enable security features in SICX. To make SICX use Hydra a row needs to be added to *.sicx* file in the home directory:

```
hydraConfig=/home/joni/.sicx_data/hydras.properties
```

Then the file *hydras.properties* needs to be created with following content when using three hydra servers:

```
servers=test1,test2,test3
test1.url=<hydra_server_1_address>
test2.url=<hydra_server_2_address>
test3.url=<hydra_server_3_address>
```

The clients source code needs be recompiled since two files will to be modified. Modify the value of *src/main/java/fi/hip/sicx/vaadin/SICXUploader.java* files parameter *use_encryption* from *false* to *true* to use SICX with Hydra.

Also change clients *src/main/java/fi/hip/sicx/webdav/FileResource.java* files *false* boolean back to *true* on this row:

```
mc.downloadFile(meta, man, target, file, false, null);
```

Now client can be recompiled similarly as without security:

```
# cd <client_directory>
# mvn package
```

SICX can now be started as without security.

3.3 Testing script

The testing script takes as parameters the file to be tested, the operation type and the repeat count. The operation type can be either read or write. The script moves the specified file to SICX, dCache and loacla folder as many times as the repeat count defines and measures the operation speed. In the end of a run an average duration for each system is calculated, printed and saved in a file *results.txt*.

To measure the actual duration of the WebDAV transfers the command from *inotify-tools* package is used. This has to be done since *davfs2* performs the file transfers asynchronously. The *inotifywait* command can be set to terminate when the WebDAV transfer is actually finished. The *inotify-tools* package can be installed with yum:

```
[root] # yum install inotify-tools
```

When using the local file storage option of SICX and running the client and meta as described above the stipes of the stored files are saved in *<client_dir>/target/jnlp/local/filesystemstorage/sicxhiptek/* directory.

By running inotifywait command with proper options in the *filesystemstorage/sicxhiptek* directory it is possible to wait for *write_close* operation to happen to the last file stripe part:

```
# inotifywait -e close_write --exclude .{34}d2[0-5] .
```

File is written in the seven stripes and when the last stripe which's file name ends with *6* is closed the *inotifywait* command terminates and the script moves forward.

Before each run the scripts removes all additional files, clears caches and remounts the storage systems. The files that are cached by *davfs2* are saved to */var/cache/davfs2/localhost-webdav-SecureRoot+mnt-sicx+root/* or */.davfs2/cache/localhost-webdav-SecureRoot+mnt-sicx+<user_name>/* directory and these are cleared before each run.

The complete test script can be found as attachment. TODO

4 Results

4.1 Results without encryption

Table 1 represents the calculated average durations of write operations for each system and file size without encryption. Table 2 shows the corresponding read durations. Each average has been calculated from 100 individual measurements using arithmetic average.

Table 1: Average write durations of different sized files in seconds without security.

File size (MB)	SICX (s)	dCache (s)	Local (s)
1	0.995060466	0.22608877	0.08712682
10	1.432974452	0.329647206	0.164678673
100	5.852578117	1.481485593	0.902978076
500	33.39584062	9.789962394	7.681851061
1000	81.94414454	19.7371116	16.01614241
2000	181.2868072	43.10176151	35.12676846

Table 2: Average read durations of different sized files in seconds without security.

File size (MB)	SICX (s)	dCache (s)	Local (s)
1	0.279338557	0.097962935	0.077306779
10	0.499196564	0.177126015	0.149156386
100	3.050278099	1.050536875	0.867041153
500	14.10778754	8.147476652	7.776042616
1000	44.37758243	17.56032526	15.93490227
2000	116.6204704	37.00713193	35.04297088

Figures 2 and 3 represent the write and read durations without encryption relatively to the slowest system.

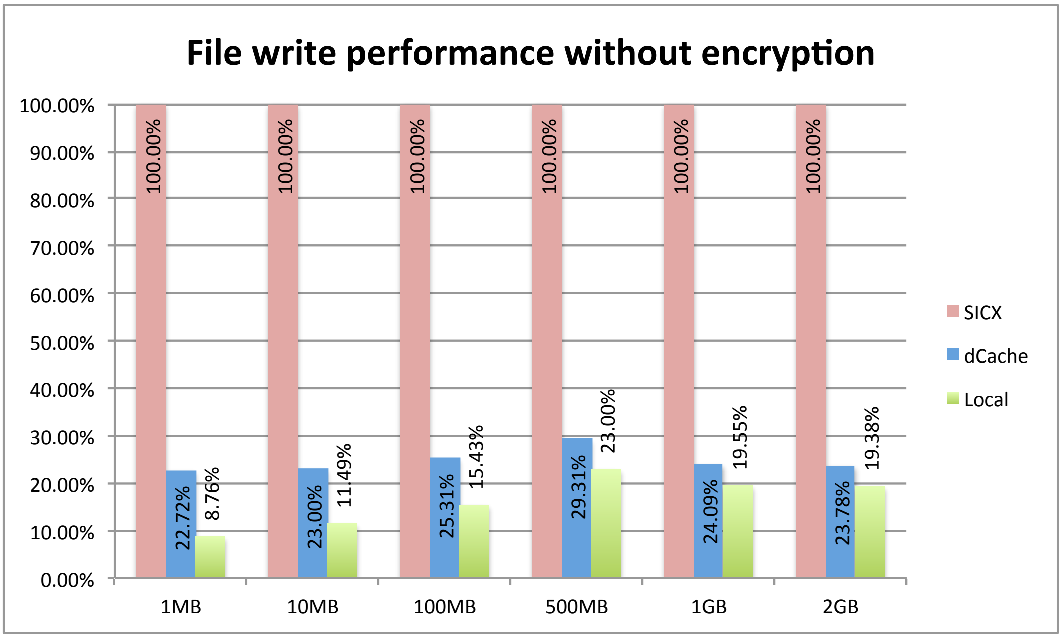

Figure 2: Average write speeds of different sized files represented relatively to the slowest system without security. Smaller value means better performance.

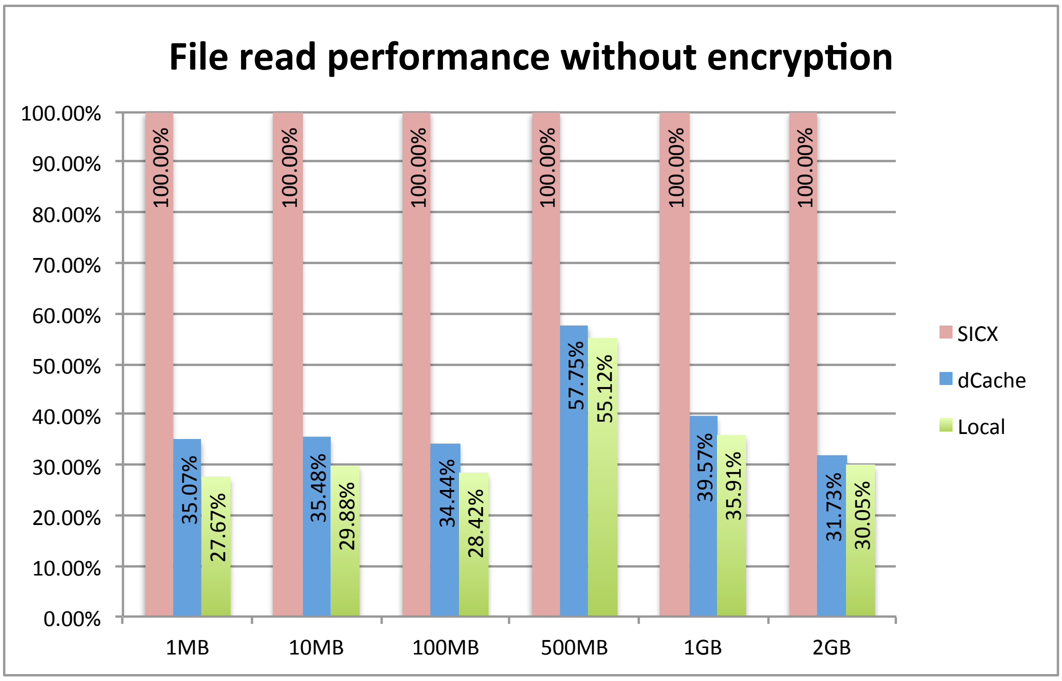

Figure 3: Average read speeds of different sized files represented relatively to the slowest system without security. Smaller value means better performance.

4.2 Results with encryption

Table 3 represents average durations of write operations for each system and file size with security features turned on. Table 4 shows the corresponding read durations. Averages have been calculated from 100 individual measurements.

Table 3: Average write durations of different sized files in seconds with security on. Smaller value means better performance.

File size (MB)	SICX (s)	dCache (s)
1	1.579089461	0.318504657
10	2.197966071	0.825348357
100	6.44395082	5.806640365
500	34.48881149	26.9359947
1000	71.7147843	50.02805607
2000	182.8368527	97.76291727

Table 4: Average read durations of different sized files in seconds with security on. Smaller value means better performance.

File size (MB)	SICX (s)	dCache (s)
1	0.936693082	0.133272618
10	1.31002582	0.53176591
100	5.005096579	4.898502056
500	22.17771329	24.68942203991
1000	57.81379303	49.22540283329
2000	132.9188101	115.547600952

Figures 4 and 5 represent the write and read durations with encryption turned on relatively to the slowest system.

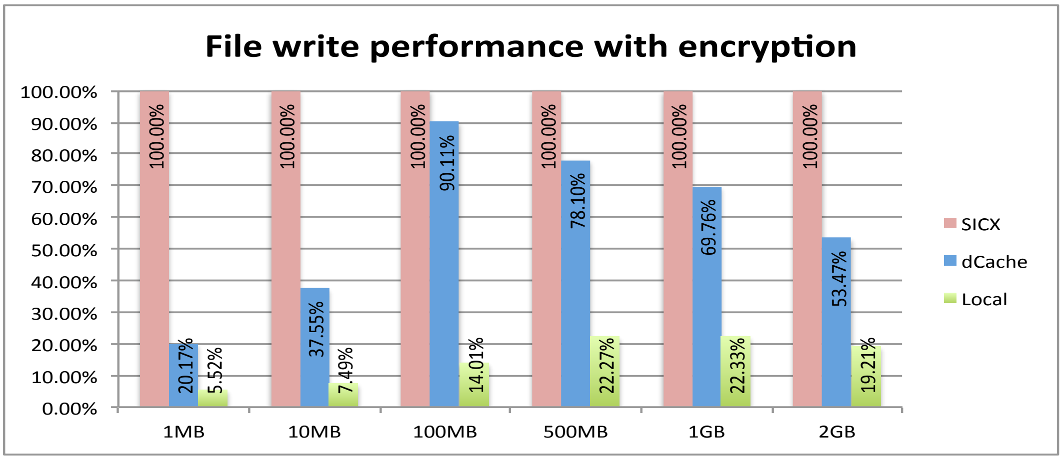

Figure 4: Average write speeds of different sized files represented relatively to the slowest system with encryption. Smaller value means better performance.

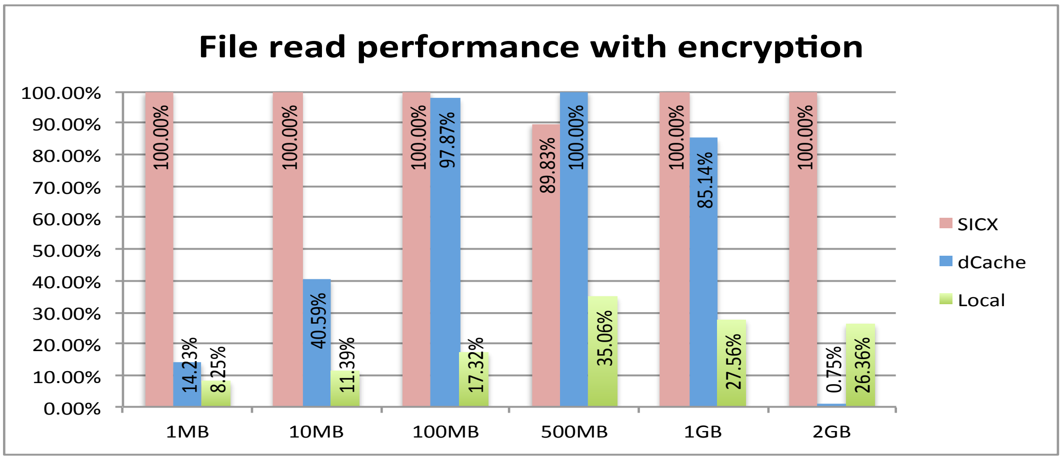

Figure 5: Average read speeds of different sized files represented relatively to the slowest system with encryption. Smaller value means better performance.

4.3 Combined results

Figure 6 illustrates the write results with and without encryption. Figure 7 shows corresponding results for read operations.

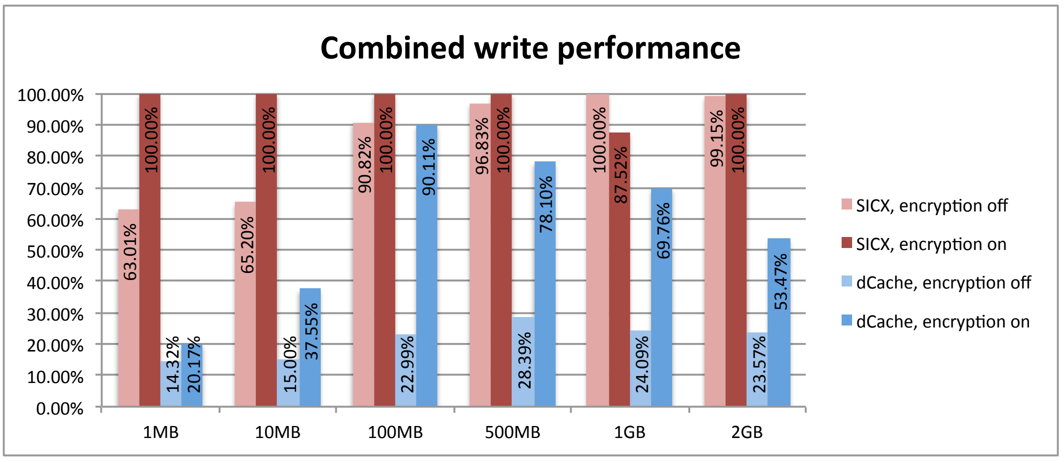

Figure 6: Average write speeds of different sized files represented relatively to the slowest system with and without encryption.

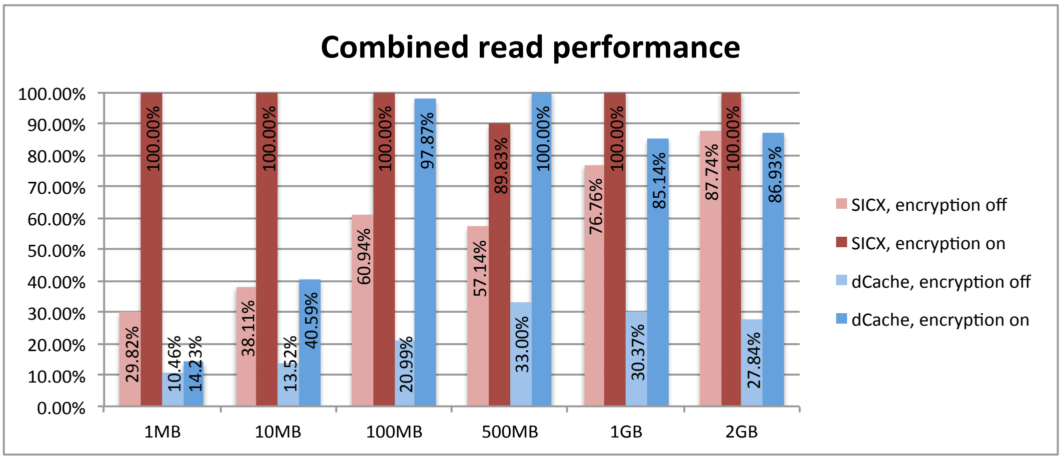

Figure 7: Average read speeds of different sized files represented relatively to the slowest system with and without encryption.

5 Conclusions

Write results clarify that without encryption SICX is significantly slower than dCache. When SICX's encryption is turned on, the write performance with small files decreases. When the file size is grown, the encryption doesn't have a substantial effect to write durations of SICX. With 2 GB files the difference with and without encryption is only marginal.

Same effect can be seen in SICX's write durations. As the file size is grown the relative overhead, caused by encryption, decreases.

This cannot be seen with dCache, because correlation of results with and without encryption is weaker. dCache is slower than SICX only in one test case, when reading 500 MB files with encryption. This gives a clue that files of size close to 500 MB are least efficent to retrieve from dCache.

If the write results of dCache and SICX are compared with encryption on, it can be seen that dCache's relative duration to SICX decreases as the file size grows. This suggests that encryption of SICX takes more time for each mega byte than the Kerberos encryption of dCache.

Overall the results state that dCache is a really efficient storage middle-ware. When writing files, SICX is significantly slower than dCache. When files are retrieved from the systems, the differences are smaller, especially with large files. SICX's encrytion seems to add almost a fixed time penalty for each file processed. This leads to the fact that the cost of SICX's encryption is only marginal with large files.

References

[1] *The dCache book for 2.6-series (FHS layout)* 20.8.2013
URL: http://www.dcache.org/manuals/Book-2.6/Book-fhs-a4.pdf

[2] *dCache, the Overview, Patrick Fuhrmann* 20.8.2013
URL: http://www.dcache.org/manuals/dcache-whitepaper-light.pdf

[3] *dCache Home Page.* 20.8.2013
URL: http://www.dcache.org

[4] *The Large Hadron Collider* 20.8.2013
URL: http://home.web.cern.ch/about/accelerators/large-hadron-collider

[5] *About CERN* 20.8.2013
URL: http://home.web.cern.ch/about

[6] *Helsinki Institute Of Physics* 20.8.2013
URL: http://www.hip.fi/

[7] *HIPTEK project* 20.8.2013
URL: http://hiptek.web.cern.ch/hiptek/

[8] *Redundant array of independent clouds* 20.8.2013
URL: http://www.google.com/patents/US20120047339

[9] *Vendor Lock-in Definition* 20.8.2013
URL: http://www.linfo.org/vendor_lockin.html

[10] *Ubuntu Home Page* 20.8.2013
URL: http://www.ubuntu.com/

[11] *The Java® Virtual Machine Specification* 20.8.2013
URL: http://docs.oracle.com/javase/specs/jvms/se7/html/

[12] *OpenJDK Home Page* 20.8.2013
URL: http://openjdk.java.net/

[13] *PostgreSQL - About* 20.8.2013
URL: http://www.postgresql.org/about/

[14] *dCache, Chimera* 20.8.2013
URL: https://www.opensciencegrid.org/bin/view/Storage/DCacheChimera

[15] *Scientific Linux Home Page* 20.8.2013
URL: https://www.scientificlinux.org

[16] *Network File System (NFS) Version 4 Minor Version 1 Protocol* 20.8.2013
URL: http://tools.ietf.org/html/rfc5661

[17] *Apache Maven* 20.8.2013
URL: http://maven.apache.org/

[18] *Hydra service* 20.8.2013
URL: http://wiki.grid.auth.gr/wiki/bin/view/Groups/ALL/UsingTheHydraServiceForDataEncryption

[19] *Hydra implementation on GitHub* 20.8.2013
URL: https://github.com/jhahkala/hhydra

[20] *SICX meta server on GitHub* 20.8.2013
URL: https://github.com/jhahkala/meta

[21] *davfs2 - Summary* 20.8.2013
URL: http://savannah.nongnu.org/projects/davfs2

[22] *davfs2 - Downloads* 20.8.2013
URL: http://download.savannah.gnu.org/releases/davfs2/

[23] *HTTP Extensions for Distributed Authoring – WEBDAV* 20.8.2013
URL: http://tools.ietf.org/html/rfc2518

[24] *neon* 20.8.2013
URL: http://www.webdav.org/neon/

[25] *OpenSSH - Goals* 20.8.2013
URL: http://www.openssh.org/goals.html

[26] *File Permissions - chmod* 20.8.2013
URL: http://www.linux.org/threads/file-permissions-chmod.4094/

[27] *Kerberos: The Network Authentication Protocol* 20.8.2013
URL: http://web.mit.edu/kerberos/