Cloud Computing for Cheminformatics

Abhik Seal

Contents

1 Using Mongo DB for Cheminformatics

Database developers have faced extreme difficulties in scaling relational databases. It turns out Mongo DB is an immediate attractive alternative because of its scaling strategy and intuitive data model. Mongo DB is a document-based database in which most of the product's information can be represented within a single document. Using a documented oriented model it makes its possible to represent complex hierarchical relationships with a single record. Without a fixed schema in this these type of systems adding or removing fields as needed becomes much easier. Apart from general purpose database Mongo DB has other unique features as well :

- Mongo DB supports **indexing** which allows a variety of fast queries.
- Mongo DB supports **aggregation** which allows processing of data records and return the computed results.
- Mongo DB supports distributed storage (**sharding**) of data across several nodes of computer.

Matt Swain in his blog () already implemented chemical similarity search using MongoDB and RDKit chemical cartridge on a single node machine. According to swain Mongo DB can be used as an effective search engine. However at the end of his blog he suggested scaling Mongo DB by sharding . Sharding refers to the process of splitting data up across machines. By putting a subset of data on each machine, it becomes possible to store more data and handle more load without requiring larger or more powerful machines, just a larger quantity of less-powerful machines. MongoDB supports autosharding, which means it automates load balancing data across shards and makes it easier to add and remove capacity. MongoDB's sharding allows one to create a cluster of many machines (shards) and break up the collection across them, putting a subset of data on each shard. Sharding

of multiple machines helps to look like that working on single machine with distributed storage. A mongos server keeps track of the shards which contains information about which shards contains what type of data. mongos send the query to every shard and then gather up the results on the mongos server shell. Sharding in used because ,

- to increase available RAM,
- to increase the disk space,
- reduce load on server,
- read and write more data efficiently than a single mongod server can handle.

1.1 Configuring Shards in Cluster

For configuring Mongo DB across multiple machine Cloudmesh Python API is used . The vmstart.py code starts the ubuntu machines as shard machines in India grid and write, executes the filescript.sh script given below in the box.

```
sudo mkdir -p /data/shardb
sudo chown ubuntu /data
sudo chown ubuntu /data/shardb
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen'
      | sudo tee /etc/apt/sources.list.d/mongodb.list
sudo apt-get update
sudo apt-get install -y mongodb-org
ssh-keygen -b 4096 -t rsa -f /home/ubuntu/.ssh/id_rsa -P ""
```

Config servers are lie the brains of the cluster: they hold all of the metadata about which servers hold what data.Thus, they must be set up first and the data they hold is extremely important. Each config server should be on a separate physical machine . For the config server and mongos router separatelya VMs are started and Mongo DB is installed on it. The config servers must be started before any of the mongos processes, as mongos pulls its configuration from them. Config servers are standalone mongod processes, and it is started as the same way as normal mongod processes. For production purposes it is recomended to use 3 config servers and for testing puporses one config server is fine to go with. Also we start the shard servers on each of the VMs using the command given below. After shard servers,config servers and and router server is ready we copy the ssh keys to each of the machines for secure data transfer between machines.

```
mongod --configsvr --dbpath=/data/configdb
mongod --shardsvr --dbpath=/data/shardb
```

Also when we start up config servers, we do not use the –replSet option: config servers are not members of a replica set. The –configsvr option indicates to the mongod that VM is used as a config server. It is not strictly required, as all it does is change the default port mongod which listens on to 27019 and the default data directory to /data/configdb. Once we have the shard servers and confif servers running we start the mongos router server which needs to know where the config servers are so mongos is started with the –configdb option:

```
mongos --configdb config-1:27019 > -f /var/lib/mongos.conf
```

mongos runs on port 27017. Note that it does not need a data directory. Adding shards to the mongos is easy once in the shell the command below adds three shards,

```
> sh.addShard("server-1:27017,server-2:27017,server-3:27017")
```

Mongo DB won't distribute your data automatically until we explicitly tell which database and collections to shard. We explicitly shard the chembldb database,molecules collection and mfp_counts collection. On the mongos shell the database and collection are sharded by,

```
>db.runCommand({enablesharding: "chembldb"});
>sh.shardCollection("chembldb.molecules", { "_id": "hashed" } )
>sh.shardCollection("chembldb.mfp1_counts", { "_id": "hashed" } )
```

1.2 Building and Querying Chemical database

The basic objective of chemical similarity searching tasks is to find molecules in a database that are most similar to a given query molecule. Most simply, this involves finding molecules with a similarity to the query molecule over a certain threshold. One of the naive approaches would be to calculate fingerprint for the query molecule, then iterates through the entire database, calculating the fingerprint for each molecule and the Tanimoto coefficient of that fingerprint with the query fingerprint, returning those where the Tanimoto coefficient is above the threshold. Whilst simple, this approach is impracticably slow on any sizeable database, taking many minutes for a single query. To improve the search efficiency we can pre-calculate the fingerprint for every molecule in the database so it doesn't have to be redundantly calculated every time a search is performed. The same goes for the total number of bits in the fingerprint. To futher enhance the search speed one can filter out unsuitable molecules before going through the expensive process of calculating the exact Tanimoto coefficient. Two tricks to speed up the search efficiency :

- Given a query fingerprint with N_a bits, any molecule in the database must have N_b bits within the range $TN_a \leq N_b \leq N_a/T$ to have a similarity above a threshold T. This is because if the number of bits in each fingerprint differ by too much, it would be impossible for the similarity to be above the threshold even in the most ideal case where one fingerprint is an exact subset of the other.
- Out of any $N_a - TN_a + 1$ bits chosen at random from the query fingerprint a, at least one must be present in any other fingerprint b to achieve a similarity above the threshold T. This is because even if all the remaining other $TN_a - 1$ bits are in common, it still would not be enough for the similarity to be above the threshold.

For the second option, the bits chosen from the query fingerprint don't actually need to be random — in fact, we can optimise even further by ensuring we choose the rarest fingerprint bits, thus restricting the query set as much as possible. However, this does mean we need to know how rare or common each fingerprint bit is in the entire molecule collection. We can do this by building an additional fingerprint counts collection, which ideally should be updated periodically if the main molecules collection changes.

To build a chemical database of fingerprints use the db_build.py program in the codes folder. Before using db_build program one needs to install the RDKit chemical toolkit . In ubuntu machines it is installed using

```
sudo apt-get install python-rdkit librdkit1 rdkit-data
```

db_build.py is a command line argument program where one can input .sdf,.sdf.gz and .smi format, the pattern of fingerprint, fingerprint length and fingerprint tag name . Currently morgan type, RDKFingerprint and rdkit maccs keys are supported. To use it install script Pymongo and RDKit need to be installed.To use automated generation of mongo database using db_build.py check the snippet below.

```
$ python db_build.py -h
usage: db_build.py [-h] --i I --db DB [--tag TAG] [--fpSize FPSIZE]
                   [--fpname FPNAME]
                   {morgan,rdkfp,rdmaccs} ...

Build a Database of fingerprints in MongoDB

optional arguments:
  -h, --help            show this help message and exit
  --i I                 input the structure file .sdf,.sdf.gz and .smi
  --db DB               Input Database Name
  --tag TAG             Give tag name. Must be present in structure file. Eg
                        'chembl_id'
  --fpSize FPSIZE       Length of the fingerprints
  --fpname FPNAME       Name of the fp Eg: mfp1,mfp2 .. etc

subcommands:
  valid subcommands

  {morgan,rdkfp,rdmaccs}
                        additional help
    morgan              Generate Morgan type fingerprints
    rdkfp               Generate RDKFingerprint
    rdmaccs             Generate MACCS Keys

# To generate morgan type fingerprints with default parameters and fpSize 1024
$ python dbbuild.py --db moltest --i benzodiazepine.smi --tag chembl_id
                    --fpSize 1024 --fpname mfp2 morgan

# To generate morgan type fingerprints with fpSize 1024 and radius of 4
$ python db_build.py --db moltest --i benzodiazepine.smi --tag chembl_id
                    --fpSize 1024 --fpname mfp3 morgan --radius 4
```

In our study we use the chembl.sdf file, morgan fingerprints with radius 2 and stored in chembldb database in molecules and mfp1_counts collection.

```
# To generate morgan type fingerprints with radius 2 into chembldb database.
python db_build.py --db chembldb --i chembl.sdf --tag chembl_id fpname mfp1 morgan
```

Once the database is built with one fingerprint addfps.py can be called to generate more fingerprints over the molecular data. This way multiple fingerprints can be generated and stored. The code for addfps.py is stored in codes folder. The code is almost similar to db_buildy but here

you need to specify the database for which you will generate the fingerprint. Currently addfps.py supports morgan type, RDKFingerprint and maccs keys fingerprints. To execute it,

```
python addfps.py --db moltest --fpSize 1024 --fpname mfp2 morgan
```

To see everything working fine a sample smi file benzodiazepine.smi is given and is explained below.

```
# Generating maccs keys
$ python db_build.py --i benzodiazepine.smi --db chemtest
                    --tag chembl_id --fpname mfp1 rdmaccs
fingerprints mfp1 done ...
Building Indices...

# Go to the mongo terminal and check moltest is created
> show dbs
admin       (empty)
chembldb    0.078GB
local       0.078GB
> use chemtest
# Shows the mfp_1 counts are generated.
> show collections
mfp1_counts
molecules
system.indexes

# Adding morgan fingerprints with length 1024 bits to the existing collection
$ python addfps.py --db chemtest --fpSize 1024
                    --fpname mfp2 morgan
fingerprints mfp2 done ...
Building Indices...

# In mongo terminal shows mfp_2 counts added

> show collections
mfp1_counts
mfp2_counts
molecules
system.indexes
```

The query to the mongo database can be done using the Query.py script in the codes folder. In Query script is a command line program where user should give smiles string(smi) , database name to search(db) , the tag name of ids which is given for generation of original database(tag) , size of the fingerprint(fpsize) and its parameters for generation of similar type of fingerprint and fingperprint name (fpname) as given in the search database (ex: mfp1). Below shows the script how it is executed. MongoDB version 2.6 introduced some new aggregation features that may have better performance. Of particular interest is the $setIntersection operator, which is exactly what we need to calculate the number of bits in common between two fingerprints.

Here's a similarity search aggregation pipeline that uses these new features:

```
qn = len(qfp)
qmin = int(qn * threshold)
qmax = int(qn / threshold)
reqbits = [count['_id'] for count in db.mfp_counts.find({'_id': {'$in': qfp}}).
sort('count', 1).limit(qn - qmin + 1)]
aggregate = [
    {'$match': {'mfp.count': {'$gte': qmin, '$lte': qmax}, 'mfp.bits': {'$in':
    reqbits}}},
    {'$project': {
        'tanimoto': {'$let': {
            'vars': {'common': {'$size': {'$setIntersection': ['$mfp.bits', qfp]}}},
            'in': {'$divide': ['$$common', {'$subtract': [{'$add': [qn,
            '$mfp.count']},'$$common']}]}
        }},
        'smiles': 1,
        'chembl_id': 1
    }},
    {'$match':  {'tanimoto': {'$gte': threshold}}}
]
response = db.molecules.aggregate(aggregate)
for result in response['result']:
    print '\%s : \%s : \%s' \% (result['tanimoto'] * 100, result['chembl_id']

#-h open the help file
$ python Query.py -h
usage: Query.py [-h] --db DB --smi SMI [--fpSize FPSIZE] --fpname FPNAME
                    [--t T] [--tag TAG]
                    {morgan,rdkfp,rdmaccs} ...

Search MongoDB database

optional arguments:
  -h, --help            show this help message and exit
  --db DB               Input Database Name
  --smi SMI             Enter the smiles string
  --fpSize FPSIZE       Length of the fingerprints
  --fpname FPNAME       Name of the fp ex:mfp1,mfp2 .. etc
  --t T                 Similarity threshold
  --tag TAG             tag name in the original database Ex:chembl_id

subcommands:
  valid subcommands

  {morgan,rdkfp,rdmaccs}
                        additional help
    morgan              Generate Morgan type fingerprints
    rdkfp               Generate RDKFingerprint
    rdmaccs             Generate MACCS Keys
```

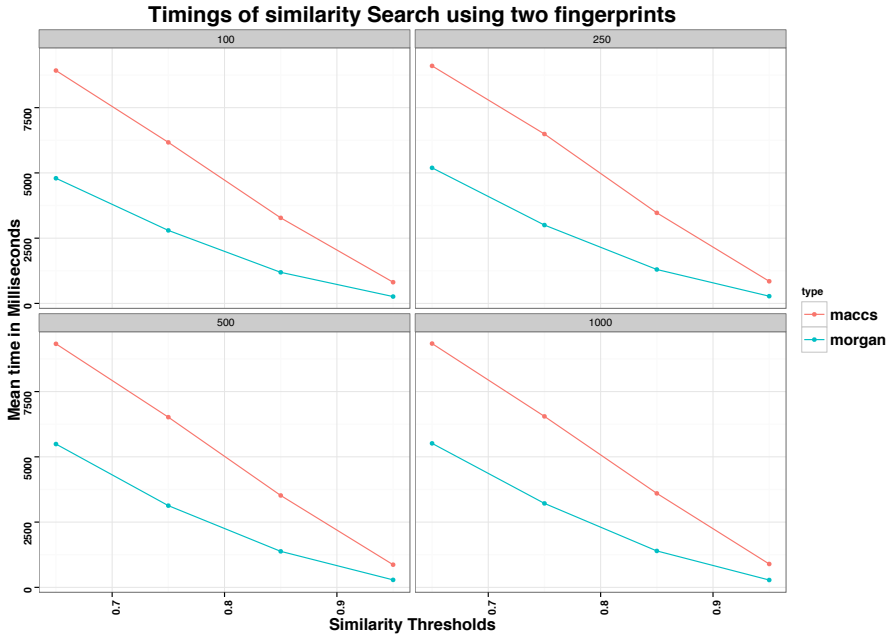

Figure 1: Mean timing of search using MongoDB using different fingerprints with different sets of molecules.

```
# Searching the database
$ python Query.py --db chembldb --smi
'CC1=NN=C2N1C3=C(C=C(C=C3)Cl)C(=NC2)C4=CC=CC=C4' --fpSize 512
--fpname mfp1 --t 0.8 --tag chembl_id morgan --radius 2
fingerprints mfp1 done ...
start Aggregate ..
response done ..
Hits:  6 Time :  0.0532460212708
1.0: 450819
0.952380952381: 19002642
1.0: 2118
0.952380952381: 178274
0.813953488372: 12562523
0.818181818182: 21489341
```

1.3 Results and Analysis of Chemical Search

Some benchmarks were ran by using 100,250,500 and 1000 random molecules from the database as query molecules and taking the median query time. The profile.py script computes the mean and median query times for different sets of molecules at different similarity thresholds. Figure 1 and 2 shows the mean and median query times using maccs keys and morgan fingperprints. Morgan fingerprints(features) is of length 2048 bits in length and is larger in size than the 166 maccs keys. We see from the Figure 3 using 166 maccs keys takes more time to search than a long 2048 keys.

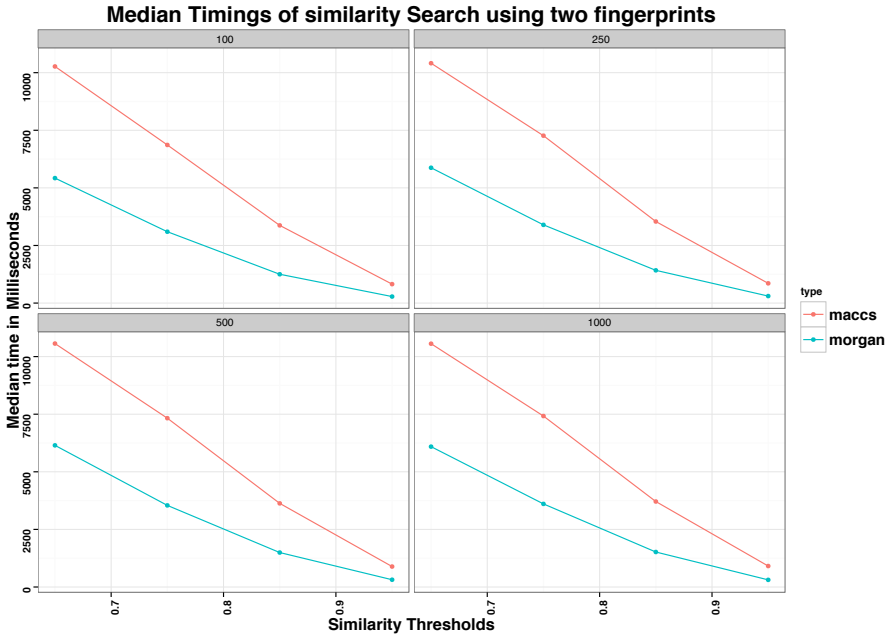

Figure 2: Median timing of search using MongoDB using different fingerprints with different sets of molecules.

This is probably due to information carried on morgan is more than maccs keys. Even the number of hits retrieved is less for the morgan fingerprint than maccs keys because maccs has 166 feature to define a compound and it not as good as morgan fingerprint to retrieve good results.

2 e-Science Cloud for Cheminformatics

2.1 Introduction

e-Science Central is a cloud-based Science Platform as a Service that allows scientists to store, analyze and share data in the cloud. It can operate concurrently on hundreds of computers and is fully accessible using only a web browser. It provides a unified environment for storing data, code and workflows and can process data at scale on hundreds of servers using the Amazon or private cloud computing platforms. The web interface provides a file manager for uploading and managing data, a workflow editor for the creation and execution of workflows, a code development environment that can be used to create new workflow blocks using R, Gnuplot, javascript and a collection of utilities that allow the system to be managed. The Workflow Engine differs from workflow systems such as Taverna in that it provides both the environment to host application code and also to co-ordinate exchanging and and processing data using this code. However, commercial applications such as Knime and pipeline pilot server provides the same advantages with more advanced frameworks for develop web applications over it. Workflows are creating using blocks. Blocks can have any number of inputs and outputs, however only one connection can be present on any given Block input whereas any number of connections can be present on any given Block output. In order for a Block to execute,

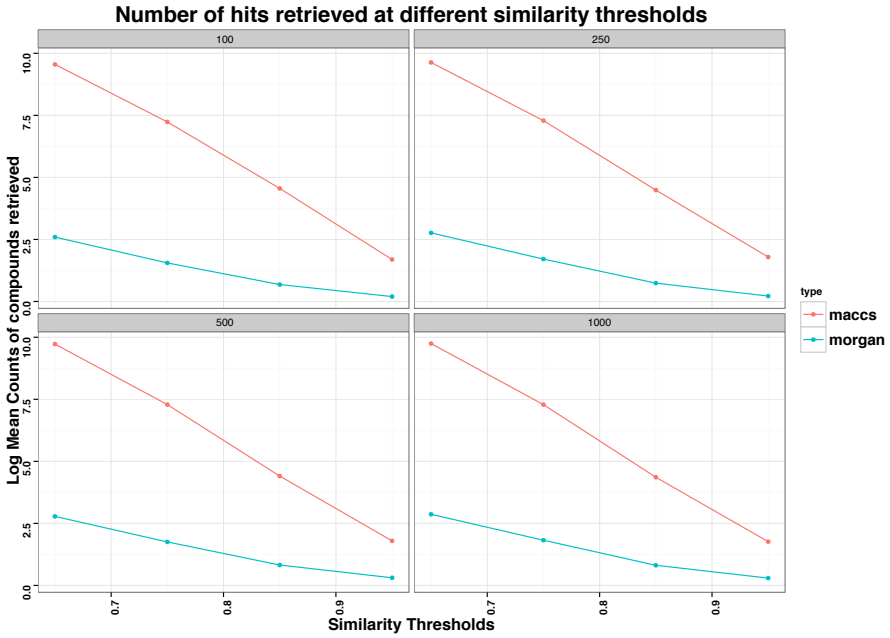

Figure 3: Number of hits retrieved search using MongoDB using different fingerprints with different sets of molecules.

data must be present on all of its Input connections.

2.2 Installation of e-Science cloud system

For Mac OSX Subversion and Maven (you need to install Xcode for Subversion and - depending on OS version - possibly Homebrew to install Maven, e.g.brew install Maven). Steps to install e-science cloud on ubuntu

- Install maven and subversion
- Download the code from source-forge

  ```
  $ svn checkout svn://svn.code.sf.net/p/esciencecentral/code/trunk esciencecentral-code
  ```

- install ansible

  ```
  sudo pip install ansible --quiet
  ```

- Edit inventory.ini and set your target Ubuntu machine(s) instead of the default example addresses.

  ```
  [all:vars]
  # Set a password for the sudo command
  ```

```
ansible_sudo_pass=
# Set the user name for the remote host
remote_user=ubuntu

# This represents a group where the postgres database engine will be running.
# Please change the address according to your target environment.

# [TODO] Support for clustered postgres installation
[db_server]
# Currently, this section cannot include more than one server address
ubuntu@ip-address
# This represents a group where the e-Science Central server is going to be
# placed. Please change the address according to your target environment.

# [TODO] Support for clustered jBoss AS installation

[esc_server]
# Currently, this section cannot include more than one server address
ubuntu@ip-address

# This represents a group of hosts where e-SC engines will be running.
# Please change the addresses according to your target environment.

[engines]
# List zero or more hosts where engine is going to be placed
# Can include the server address
ubuntu@ip-address
# engine-1.mydomain.co.uk
# engine-2.mydomain.co.uk
# engine-3.mydomain.co.uk
```

- exchange keys between the mac OS and target machines (ubuntu), e.g. with ssh-copy-id.
- run mvn clean install inside the code folder
- run ansible-playbook standalone.yml -i inventory.ini from the ansible folder. This will install all the libraries and packages required in ubuntu machine.
- Once it is installed it can be accessed using http://ip-address:8080/beta. One needs to register to the side with username and password.
- Regarding the block palette, to make it full of blocks one needs to build and upload libraries and blocks to the server. Currently, it's still only semi-automated process. First,create file /.inkspot/maven.props with contents:

```
serverURL=http://<SERVER_ADDRESS>:8080/APIServer
hostname=<SERVER_ADDRESS>
port=8080
username=<USERNAME>
password=<PASSWORD>
```

Then log in to the server and create folder "Services" and "Libraries" in your home folder. Finally, build maven tools, libraries and blocks:

- Run 'mvn clean install' in the 'code/tools/maven' directory
- Run 'mvn clean install' in 'code/webflow/WorkflowItems/libraries'.
- Run 'mvn clean install' in 'code/webflow/WorkflowItems/blocks'.

Note that the order of the build is important.

2.3 Creating workflows using R script engine

An example workflow in figure 4 is shown which is created using the block creation made using R scripting language . The workflow reads a csv file (a qsar dataset with descriptors) and then prepare and clean the file (replace NA with mean or median or 0 and remove near constant columns from the dataset and the learn a partial least squares model and output the predicted the original dataset values and r^2 values. Creating an R block using block creation tools in e-science central web application is based some xml and R scripts. Below shows the dependencies.xml file for rPLS block it takes Trainset% from the user for the training set size.

```
<!-- This is a simple dependencies file that just specifies the core library -->
<dependencies>
    <dependency>
        <libraryname>core</libraryname>
        <uselatestversion>true</uselatestversion>
        <runtimeonly>false</runtimeonly>
    </dependency>

    <dependency>
        <libraryname>r-bin</libraryname>
        <uselatestversion>true</uselatestversion>
        <runtimeonly>true</runtimeonly>
    </dependency>
</dependencies>
```

library.xml file is used to notify the workflow engine what type of block a given package contains. It provides details of other R libraries that this block requires in order to run. Given below shows the library.xml file where pls and e1071 libraries are installed.

```
<!-- Template for R Service library -->
<library>
    <type>RService</type>

    <!-- Additional CRAN packages that this service needs -->
    <cran-install>
        <package>pls</package>
        <package>e1071</package>
    </cran-install>

    <!-- Properties for library -->
```

```
    <properties>
        <property name="cran-repository" value="http://cran.us.r-project.org"/>
        <property name="cran-quiet-install" value="true"/>
    </properties>
</library>
```

The service.xml file declares a block input named x a corresponding variable x is also created in the workspace so as for output output1 and output2 is created also corresponding variables are also created.Given below the some settings made in service.xml file which created one input stream and two output stream.

```
<Inputs>
     <Input name="x" type="data-wrapper"/>
</Inputs>

<Outputs>
    <Output name="outfile1" type="data-wrapper"/>
    <Output name="outfile2" type="data-wrapper"/>
</Outputs>
```

R initialisation code is executed in the init.r file. Given below shows the initialization of libraries and r2se function.

```
library(pls)
library(e1071)
# r2se function
r2se<- function (obs,pred){
  rmse<-(mean((obs -pred)^2))^0.5
  ssr<-sum((obs - pred)^2)
  sst<-sum((obs - mean(obs))^2)
  R2<-1-(ssr/sst)
  output<-list(RMSE=rmse,RSquared=R2)
  return(output)
}
```

Below shows the main.R in the prepare and clean data block where it remove the constant columns and replace ”NA”s with mean or median or 0 which the user takes it as input.

```
# remove constant columns
df<- x ;
print (dim(df))
d <- data.frame(df[,1:49])
constcut<-getProperty("constcut")
dropc <- apply(d, 2, function(x) { length(which(x == 0))/length(x) > constcut })
d <- d[, !dropc]
print (dim(d))
# function to remove NA with mean , median and 0
f=function(x){
  x<-as.numeric(as.character(x)) #first convert each column into numeric if it is from factor
```

```
 if (getProperty("replaceNAs")=="mean"){
    x[is.na(x)] =mean(x, na.rm=TRUE)
    return(x)
  }
  else if (getProperty("replaceNAs")=="median"){

  x[is.na(x)] =median(x, na.rm=TRUE) #convert the item with NA to median value from the column
  return(x) #display the column
  }
  else {
    x[is.na(x)] <- 0
    return(x)
  }
}

# remove missing values with mean or median or 0
ss=data.frame(apply(d,2,f))
df<-cbind(ss,data.frame(df[,50]))
print (dim(df))
outfile <- data.frame(ss)
```

Then the main.r execution code is executed only once . main.r contains the code to run partial least squares and produce two output dataframes output1 and output2 where output1 contains the test and predicted values and output2 has r^2 dataframe. Below shows the main.r script

```
# This command simply copies input x to q.data
# This command simply copies input x to q.data
q.data <- x1;
# Load library
library(caret)
library(pls)
print(dim(q.data))
c<-dim(q.data)[2]
colnames(q.data)[c]<- "Activity"
descs <- q.data
# pls code
#-----------------------------------------------------
# Preprocessing data and normalizing data

T<-preProcess(descs[,1:dim(descs)[2]],method = "BoxCox")
data<- predict(T,descs[,1:dim(descs)[2]])

#-----------------------------------------------------
# split train and test set
tr<-getProperty("Trainset%")
ts<- 1-tr
ind<-sample(2,nrow(data),replace=TRUE,prob=c(tr,ts))
trainsetX<-data[ind==1,1:dim(data)[2]]
```

```
trainY<-trainsetX$Activity
testsetX<-data[ind==2,1:dim(data)[2]]
testY<- testsetX$Activity
#-----------------------------------------------------
# Use pls to model the data
plsFit <- plsr(Activity ~ ., data = trainsetX)

# Using the first ten components
pls.test<-data.frame(predict(plsFit, testsetX, ncomp = 1:10))
pls.train<-data.frame(predict(plsFit, trainsetX, ncomp = 1:10))

#-----------------------------------------------------
# Summarizing results of test set
rlmValues <- data.frame(obs = testY,pred = pls.test$Activity.10.comps)
r2<- r2se(rlmValues$obs,rlmValues$pred)
outfile1 <- rlmValues
outfile2 <- data.frame(r2)
```

Figure shows a simple workflow created using e-science

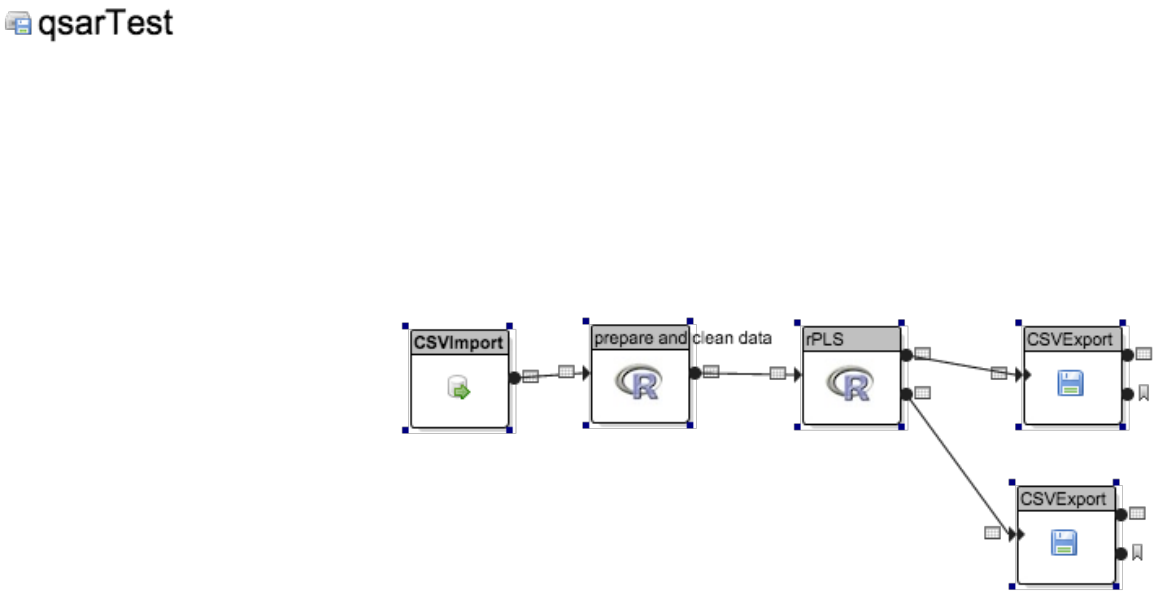

Figure 4: Generalized workflow of implementing the e-Science workflow.