SciDB

Wusuo Liu and Chia Ying Lee

5/3/2017

# Introduction

**Chia: add brief intro**

Full documentation of SciDB is provided [here](https://paradigm4.atlassian.net/wiki/display/ESD/SciDB+Documentation).

The scidb R package is available on [CRAN](https://cran.r-project.org/package=scidb), along with [documentation](https://cran.r-project.org/web/packages/scidb/vignettes/scidb.pdf).

## SciDB Data Structures and Query Languages

SciDB is a computational database management system that employs the array data model. In SciDB, dataset is organized as a multi-dimensional array. A user creates a SciDB array by specifying dimensions and attributes of the array.

An n-dimensional SciDB array has dimensions d\_1, d\_2, ..., d\_n. The size of the dimension is the number of ordered values in that dimension. For example, a 2-dimensional array may have dimensions i and j, each with values (1, 2, 3, ..., 10) and (1, 2, ..., 30) respectively.

Each combination of dimension values identifies a cell or element of the array, which can hold multiple data values called attributes (a\_1, a\_2, ..., a\_m). Each data value is referred to as an attribute, and belongs to one of the supported datatypes in SciDB.

SciDB is inherently adapted to data where the elements are sitting on regular grid of one or more dimensions, such as image, geo-spatial data. Due to its matrix-like organization of data, SciDB also has good advantage in linear algebra and statistical analysis. Several system libraries for that purpose are provided in SciDB.

## AQL and AFL

SciDB provides query languages AQL, for querying the database, and AFL, for performing function calls on database objects. They are accessed from the unix terminal.

$ iquery  
AQL% set lang AFL;  
AFL%

## scidb R package

The scidb R package provides an interface to R, allowing interaction with the database from R, primarily using the iquery function.

## Basics of SciDB R package

First, load the library, and connect to the SciDB database. By default, it connects to the localhost.

library(scidb)  
dbConnect <- scidbconnect()

The datasets in the SciDB database can be listed using the following command.

(existingArray <- iquery(db = dbConnect, query = "list('arrays')", return = T))

## No name uaid aid  
## 1 0 RATINGS 36 36  
## 2 1 RATINGS\_MATRIX 38 38  
## schema  
## 1 RATINGS<userId:int64,movieID:int64,rating:double NOT NULL,timestamp:int64> [i=0:\*,1000000,0]  
## 2 RATINGS\_MATRIX<rating:double NOT NULL> [userId=0:138493,1000,0,movieID=0:131262,1000,0]  
## availability temporary  
## 1 TRUE FALSE  
## 2 TRUE FALSE

The above expression illustrates the basic usage of the iquery function: it is told the database to connect to, and the query to run. Setting return = T will return the scidb arrary to an R data frame, which we have called existingArray. By default, return = F, meaning that the query is performed in the SciDB database, but no output is returned to R.

## Installation

### Method 1: SciDB's VM

Download the virtual machine from [here](insert%20Google%20Docs%20URL). The virtual machine has SciDB pre-installed, and R and RStudio server already set up with scidb R package installed. However, note that the SciDB installation has problems with MPI, preventing the use of some essential functions such as the gesvd for computing SVD. Also, the installed version of scidb R package is not the latest version.

### Method 2: Install from scratch

We install SciDB Community Edition 15.12 on a blank Ubuntu 14.04. SciDB Community Edition installation guide and reference manual is [here](https://paradigm4.atlassian.net/wiki/display/ESD/SciDB+Community+Edition+Installation+Guide).

The community edition does not have all the functionalities of SciDB. For example, the community edition does not have the truncated singular value decomposition tsvd in its dense\_linear\_algebra library. Functions like tsvd are only available in SciDB’s proprietary edition. Please check the function status before using it.

SciDB requires access to the root user account on the Linux system. Ubuntu, by default, does not have a root account. Please type the following in the terminal:

sudo passwd root

and set up your root password.

The following instructions are meant for SciDB installation on one computing node acting as both the host and client, assuming the username is “scidb” and the hosting website is 127.0.0.1. The instructions are similar to the official installation guide with several important extra steps.

**Download SciDB Community Edition**

To download SciDB CE, do the following:

1. [Access the 15.12 CE page](https://drive.google.com/folderview?id=0B7yt0n33Us0rT1FJdmxFV2g0OHc&usp=drive_web#list) and download the 15.12 source code tarball. Please do not just try the VM there.

cd /home; tar –xvzf scidb-15.12.1.4cadab5.tar.gz

* + Optional: create a directory under the /home called **scidbtrunk** and install the tarball there to match the installation instructions.

**Installing Expect, and SSH Packages**

To install expect, and ssh packages, run the following:

|  |
| --- |
| sudo apt-get -y install expect openssh-server openssh-client |

Starting the SSH Server

To start the SSH server, do the following:

* + Check whether ssh is running:

|  |
| --- |
| sudo service ssh status |

* + If ssh is not running, start the ssh server as follows:

|  |
| --- |
| sudo /usr/sbin/update-rc.d ssh defaults  sudo service ssh start |

* + If ssh is running restart ssh server as follows:

|  |
| --- |
| sudo service ssh restart |

**Providing Passwordless SSH**

SciDB requires passwordless ssh on all servers logging in on each server as:

* the *scidb*user, for starting and stopping SciDB.
* the *root* user, for initializing SciDB on each server.

|  |
| --- |
| cd ~  ssh-keygen # DO NOT enter a passphrase; accept the default.  # Since your machine acts both the host and the client, in /home/.ssh directory, make a copy of id\_rsa.pub, rename it as authorized\_keys. Copy authorized\_keys to /root/.ssh directory. If necessary, copy authorized\_keys to all the other users on the computing cluster put in their /.ssh directories.  chmod 755 ~  chmod 755 ~/.ssh  cd /home/scidbtrunk  deployment/deploy.sh access root 127.0.0.1 *<hostIP1>*  deployment/deploy.sh access scidb 127.0.0.1 *<hostIP1>* |

In the *deploy acces*s commands, for a single host, omit *<hostIP1>*. With more than two hosts, list all of them. This rule applies to all subsequent commands using *<hostIP1>.*

To confirm that as user *scidb* on 127.0.0.1, you can ssh to all hosts without providing a password, run the following:

|  |
| --- |
| ssh 127.0.0.1 date  ssh <hostIP1> date |

If you get error “Agent admitted failure to sign using the key”, search “Startup Applications Preferences” in Ubuntu, and uncheck “SSH key Agent”.

**Installing a Single Node SciDB Instance**

SciDB implementation involves:

* Configuring the Environment
* Building SciDB
* Installing SciDB

Perform the following tasks to install a single node instance of SciDB Community Edition.

**Configuring Environment Variables**

To configure your environment variables, add the following lines to your local shell configuration script (.bashrc), then source it:

|  |
| --- |
| export SCIDB\_VER=15.12  export SCIDB\_INSTALL\_PATH=home/scidbtrunk/stage/install  export SCIDB\_BUILD\_TYPE=Debug  export PATH=$SCIDB\_INSTALL\_PATH/bin:$PATH |

For single node installations, do not point SCIDB\_INSTALL\_PATH at /opt/scidb/$SCIDB\_VER. Doing so results in the deletion of all content in SCIDB\_INSTALL\_PATH (including useful packages in /opt/scidb/$SCIDB\_VER) on installation.

* Setting the SCIDB\_BUILD\_TYPE environmental variable is optional. It is used in *run.py setup* and *run.py plugin\_setup*. The default value is Debug. For performance measurement, use RelWithDebInfo. See run.py setup -h.
* Setting the PATH environmental variable is optional, but it is helpful to find utilities such as iquery.

**Activating and Verifying the New .bashrc File**

After saving the .bashrc file, source it, echo the environment variables, and verify the response is per the  
previous settings as follows:

|  |
| --- |
| source .bashrc  echo $SCIDB\_VER  echo $SCIDB\_INSTALL\_PATH  echo $PATH |

**Building SciDB CE**

In this step, please allocate as much memory as possible to your VM. 4GB memory as the upper limit will not have the source code built successfully. This step is very memory and time consuming. Allocating more processing cores and invoking parallel compiling would be helpful.

To build SciDB, run the following:

|  |
| --- |
| cd <dev\_dir>/scidbtrunk  ./run.py -h # to learn its usage.  ./run.py setup # to configure build directories and cmake infrastructure  ./run.py make -j4 # to build the sources. |

In the 'make' sub-command, use '-j4' to speedup build with 4 threads.

**Install *shim***

The installation manual is [here](https://github.com/Paradigm4/shim).

**Install R, Rstudio and R package “scidb”**

Do not install the CRAN package “scidb”. Instead, install the package from github devtools::install\_github("Paradigm4/SciDBR")

### Start the database

$ scidb.py startall mydb

# The Big Data Problem

We use the movie ratings data from [MovieLens](https://grouplens.org/datasets/movielens/).

There are 2 datasets provided,

* Full data set
* Small data set

**Chia: add explanation of data, describe clustering goal**

## Download data

The following script downloads the two datasets from the MovieLens website, and stores them into respective subfolders in /home/scidb/Project.

## Get the small dataset  
data.dir <- "/home/scidb/Project/ml-latest-small" # Note: the full path is specified  
if (!dir.exists(data.dir)) {  
 download.file("http://files.grouplens.org/datasets/movielens/ml-latest-small.zip",   
 destfile = "/home/scidb/Project/ml-latest-small.zip")  
 unzip("/home/scidb/Project/ml-latest-small.zip")  
   
 # System command to strip header (first line) from data file ratings.csv  
 system(sprintf("tail -n +2 %s/ratings.csv > %s/ratings\_noHeader.csv", data.dir, data.dir))  
}  
rm(data.dir)  
  
## Get the full dataset  
data.dir <- "/home/scidb/Project/ml-20m"  
if (!dir.exists(data.dir)) {  
 download.file("http://files.grouplens.org/datasets/movielens/ml-20m.zip",   
 destfile = "/home/scidb/Project/ml-20m.zip")  
 unzip("/home/scidb/Project/ml-20m.zip")  
   
 system(sprintf("tail -n +2 %s/ratings.csv > %s/ratings\_noHeader.csv", data.dir, data.dir))  
}  
rm(data.dir)

## Load Data into SciDB

Next, we load the data into the SciDB database. Because the data array is a 2D array (i.e. a matrix), whereas the data is provided as a flat array (i.e. 1D array), loading the data has two steps. We explain the steps for the small dataset.

1. First, load the CSV file as a 1D array, named ratings. This is done by first creating the array, and then using the load function to read in the CSV file.

iquery(dbConnect, "CREATE ARRAY ratings <userId:int64, movieID:int64, rating:double NOT NULL, timestamp:int64> [i=0:?,1000000,0]")  
# Note 1: It's important to make rating column NOT NULL, to enable the use of spgemm later on.  
# Note 2: The 0:? allows us to create the array without knowing the size of the input file.  
# Note 3: The chunk size is 1000000 and overlap is 0  
  
iquery(dbConnect, "load(ratings, '/home/scidb/Project/ml-latest-small/ratings\_noHeader.csv', -2, 'csv')")

1. The second step is to *redimension* the ratings array from a 1D to 2D array. The columns userId and movieID in the 1D array will become the dimensions of the 2D array.

* Additionally, to be able to use subsequent functions, we need to specify the size of each dimension.

print(u\_max <- iquery(dbConnect, "aggregate(ratings, max(userId))", return = T))

## i userId\_max  
## 1 0 671

print(m\_max <- iquery(dbConnect, "aggregate(ratings, max(movieID))", return = T))

## i movieID\_max  
## 1 0 163949

The userIds range from 1 to 671. The movieIDs range from 1 to 163949. Now, we redimension `ratings` array so that userId is the first dimension (rows) and movieID is the second dimension (columns).

iquery(dbConnect, "store(redimension(ratings, <rating:double NOT NULL>[userId=0:671,?,0,movieID=0:163949,?,0]), ratings\_matrix)")  
# Note: We start the indices from 0, not 1, to enable the use of spgemm later on.

The `store` function stores the resulting array into a new array named `ratings\_matrix`.

Note that even though ratings\_matrix is a huge matrix (671 x 163949), it is very sparse.

print(n\_ratings <- iquery(dbConnect, "aggregate(ratings, count(\*))", return = T))

## i count  
## 1 0 100004

There are 100004 ratings available. All other entries of the matrix are zero, and not explicitly stored in the database. (In fact, according to the MovieLens data description, only 9000 movies are rated by the 700 users.)

Below is the corresponding code for loading the full dataset into SciDB. The names of the data arrays are RATINGS and RATINGS\_MATRIX.

iquery(dbConnect, "CREATE ARRAY RATINGS <userId:int64, movieID:int64, rating:double NOT NULL, timestamp:int64> [i=0:?,1000000,0]")  
  
iquery(dbConnect, "load(RATINGS, '/home/scidb/Project/ml-20m/ratings\_noHeader.csv', -2, 'csv')")  
  
U\_max <- iquery(dbConnect, "aggregate(RATINGS, max(userId))", return = T)$userId\_max  
M\_max <- iquery(dbConnect, "aggregate(RATINGS, max(movieID))", return = T)$movieID\_max  
  
query <- sprintf("redimension(RATINGS, <rating:double NOT NULL>[userId=0:%d,?,0,movieID=0:%d,?,0])",  
 U\_max, M\_max)  
iquery(dbConnect, sprintf("store(%s, RATINGS\_MATRIX)", query)

## Essential statistics using SciDB

We show some essential statistical methods using SciDB functions.

What is the average rating of each user?

userAvg <- iquery(dbConnect, "aggregate(ratings\_matrix, avg(rating), userId)", return = T)  
head(userAvg)

## userId rating\_avg  
## 1 1 2.550000  
## 2 2 3.486842  
## 3 3 3.568627  
## 4 4 4.348039  
## 5 5 3.910000  
## 6 6 3.261364

Similarly, we can obtain the the average average rating of each movie.

movieAvg <- iquery(dbConnect, "aggregate(ratings\_matrix, avg(rating), movieID)", return = T)  
head(movieAvg)

## movieID rating\_avg  
## 1 1 3.872470  
## 2 2 3.401869  
## 3 3 3.161017  
## 4 4 2.384615  
## 5 5 3.267857  
## 6 6 3.884615

A note of caution, the averages are computed only on non-zero ratings.

Other aggregation functions include sum, min, max, stdev, etc.

# Clustering Users in the Small Dataset

Using the small dataset, we'll cluster users in 2 ways:

1. Compute pairwise correlation (over movieIDs) between users. Apply hierarchical clustering on the correlation matrix.
2. Compute the SVD of the ratings matrix and keep the top 10 singular values. Apply kmeans clustering on the top 10 singular component coefficients.

The correlation matrix and SVD will be computed using SciDB functionality, while kmeans and hierarchical clustering will be computed using base R functionality.

## Clustering using the correlation matrix

Now, let's try to compute the correlation between users. Since userId is in rows, we just have to normalize the rows and matrix multiply .

The SciDB function for sparse matix multiply is spgemm, in the linear\_algebra library.

iquery(dbConnect, "load\_library('linear\_algebra')")

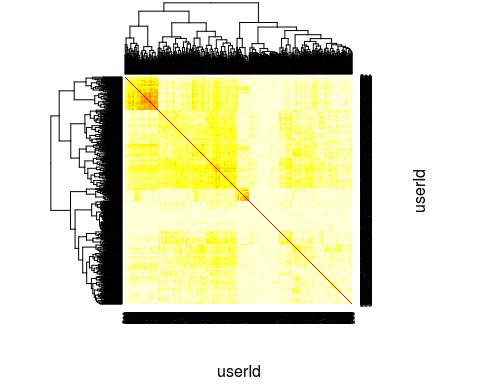
iquery(dbConnect, "store(spgemm(ratings\_matrix, transpose(ratings\_matrix)), ratings\_matrix\_MxMT)")

# Average user rating  
query\_userAvg <- "apply(aggregate(ratings\_matrix, sum(rating), userId), rating\_avg, rating\_sum/163949.0)"  
iquery(dbConnect, sprintf("store(project(%s, rating\_avg), ratings\_userAvg)", query\_userAvg))  
  
# Mean sum of square ratings  
query\_rating\_sq <- "apply(ratings\_matrix, rating\_sq, rating \* rating)"  
query\_rating\_MSS <- sprintf("apply(aggregate(%s, sum(rating\_sq), userId), rating\_MSS, rating\_sq\_sum/163949.0)",  
 query\_rating\_sq)  
iquery(dbConnect, sprintf("store(project(%s, rating\_MSS), ratings\_userMSS)", query\_rating\_MSS))  
  
# Cross join with ratings matrix  
query\_crossjoin <- "cross\_join(ratings\_matrix\_MxMT as A, ratings\_userAvg as B, A.userId, B.userId)"  
query\_crossjoin <- sprintf("cross\_join(%s as A, ratings\_userAvg as B, A.userId2, B.userId)",   
 query\_crossjoin)  
query\_crossjoin <- sprintf("cross\_join(%s as A, ratings\_userMSS as B, A.userId, B.userId)",  
 query\_crossjoin)  
query\_crossjoin <- sprintf("cross\_join(%s as A, ratings\_userMSS as B, A.userId2, B.userId)",  
 query\_crossjoin)  
iquery(dbConnect, sprintf("store(%s, ratings\_crossjoin)", query\_crossjoin))  
  
# Compute correlation matrix from M\*M^T  
query <- "apply(ratings\_crossjoin, correlation,   
(multiply/163949.0 - rating\_avg \* rating\_avg\_2) /   
sqrt((rating\_MSS - pow(rating\_avg, 2)) \* (rating\_MSS\_2 - pow(rating\_avg\_2, 2))))"  
  
iquery(dbConnect,   
 sprintf("store(project(%s, correlation), ratings\_correlationMatrix)", query))

correlationMatrix <- iquery(dbConnect,   
 "filter(ratings\_correlationMatrix, TRUE)",   
 return = T)  
  
library(Matrix, quietly = T)  
correlationMatrix <- spMatrix(nrow = 671, ncol = 671,   
 i = correlationMatrix$userId,   
 j = correlationMatrix$userId2,  
 x = correlationMatrix$correlation)

We visualize the correlation matrix using the heatmap function in R. (Actually, we'll visualize 1-correlationMatrix, which is the dissimilarity matrix.) This function also automatically computes the dendrogram and reorders the rows and columns to produce a visual clustering of the userIds.

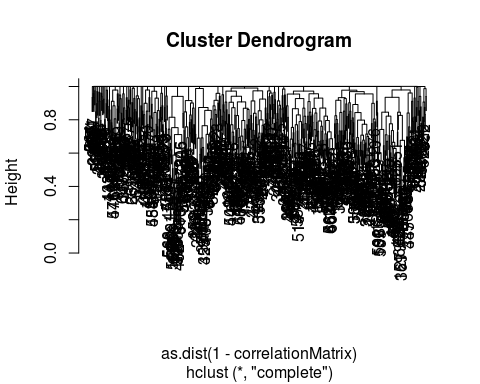
heatmap(as.matrix(1 - correlationMatrix), symm = T, xlab = "userId", ylab = "userId")



### Hierarchical clustering

We now perform hierarchical clustering on the dissimilarity matrix 1-correlationMatrix.

hc <- hclust(as.dist(1 - correlationMatrix))  
plot(hc)



## Clustering using Principle Component Analysis and the SVD

Next, we perform the SVD on the ratings matrix to obtain the top princple components of the ratings data.

SciDB provides the gesvd function in the dense\_linear\_algebra library, for computing the SVD of a dense matrix. To use this function, first load the library.

iquery(dbConnect, "load\_library('dense\_linear\_algebra')")

Next, we center the data at 0, a requirement of the PCA.

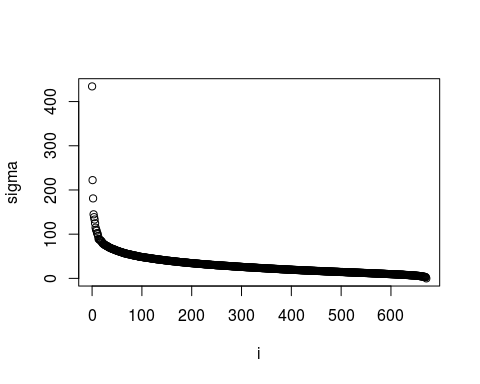
query\_userSum <- "aggregate(ratings\_matrix, avg(rating), movieID)"  
query\_crossjoin <- sprintf("cross\_join(ratings\_matrix as A, %s as B, A.movieID, B.movieID)",   
 query\_userSum)  
query\_center <- sprintf("project(apply(%s, rating\_centered, rating-rating\_avg), rating\_centered)",   
 query\_crossjoin)  
  
ratings\_matrix\_centered <- iquery(dbConnect,   
 sprintf("store(%s, ratings\_matrix\_centered)", query\_center),   
 return = T)

Now, we are ready to compute the singular values of the ratings matrix.

ratings\_singularValues <- iquery(dbConnect,   
 "store(gesvd(ratings\_matrix\_centered, 'S'), ratings\_singularValues)")  
save(ratings\_singularValues, file="/home/scidb/Project/ratings\_singularValues.Rdata")

By inspecting the plot of the singular values, we see that the top 3 singular values capture the most variation in the data.

load("/home/scidb/Project/ratings\_singularValues.Rdata")  
  
## For logistical reasons, we have saved the SVD results as Rdata format.   
## Equivalently, the singular values can be retrieved from the SciDB database  
## using the following command:  
# ratings\_singularValues <- iquery(dbConnect, "filter(ratings\_singularValues, TRUE)", return = T)  
  
plot(ratings\_singularValues)

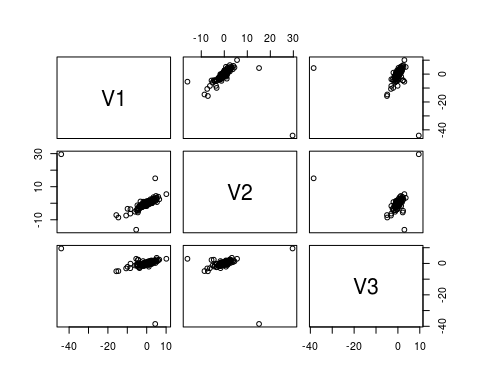


We also need the singular vectors. We'll get SciDB to compute and store the singular vectors, and then extract only the first 3 singular vectors into R for inspection.

iquery(dbConnect, "store(gesvd(ratings\_matrix\_centered, 'U'), ratingsSvdU)")

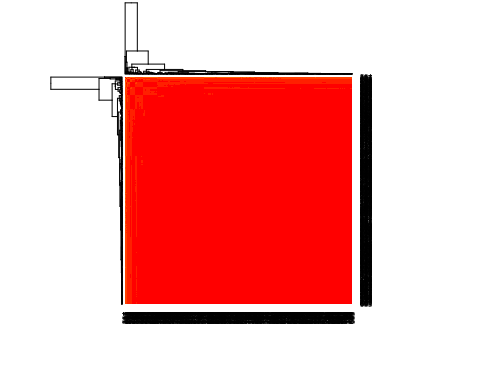
iquery(dbConnect, "store(apply(ratings\_singularValues, j, i), tmpDb)")  
iquery(dbConnect, "store(redimension(tmpDb, <sigma:double NOT NULL>[i=0:671,?,0,j=0:671,?,0]), ratingsSvdSdiag)")  
iquery(dbConnect, "remove(tmpDb)")  
  
iquery(dbConnect, "store(spgemm(ratingsSvdU, ratingsSvdSdiag), ratingsSvdUxS)")  
  
ratingsSvdUxSfirst3 <- iquery(dbConnect, "subarray(ratingsSvdUxS, 0, 0, 671, 2)", return = T)  
ratingsSvdUxSfirst3$userId <- as.integer(ratingsSvdUxSfirst3$userId)  
ratingsSvdUxSfirst3$j <- as.integer(ratingsSvdUxSfirst3$j)  
save(ratingsSvdUxSfirst3, file = "/home/scidb/Project/ratingsSvdUxSfirst3.Rdata")

load("/home/scidb/Project/ratingsSvdUxSfirst3.Rdata")  
  
library(Matrix)  
PCcoords <- spMatrix(nrow = 671, ncol = 3,   
 i = ratingsSvdUxSfirst3$userId,   
 j = ratingsSvdUxSfirst3$j + 1,   
 x = ratingsSvdUxSfirst3$multiply)  
# The +1 is due to difference in 0-indexing in SciDB and 1-indexing in R  
  
plot(as.data.frame(as.matrix(PCcoords)))

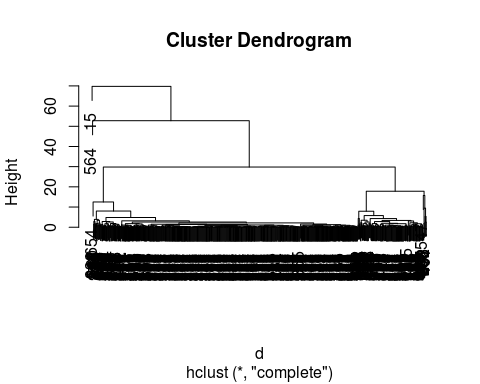


### Hierarchical clustering using PCA

d <- dist(PCcoords)  
heatmap(as.matrix(d), symm = T)



hclustTree <- hclust(d)  
plot(hclustTree)



# Big Data Computations with the Full Dataset

The computations in the previous section extend to using the full dataset. However, due to limited RAM and hard disk space on our laptops, we are unable to run the identical computations for the full dataset.

We can still show a tractable example: finding the pairwise correlations between movies.

system.time(  
 iquery(dbConnect,   
 "store(spgemm(transpose(RATINGS\_MATRIX), RATINGS\_MATRIX), RATINGS\_MATRIX\_MxMT)")  
)