Storing and Retrieving Images by Relational Data Bases vs Key-Value store

Ehsan Azizi Khadem[1], Emad Fereshteh Nezhad[2]

[1.] MSc of Computer Engineering, Department of Computer Engineering, Lorestan University, Iran
[2.] MSc of Computer Engineering, Communications Regulatory Authority of I.R of Iran
emad_fereshtehnejad@yahoo.com

Abstract: Today, we implement very huge collections of data that should be stored in several storage devices. These data collections consist of various types have different behavior. For this requirement, we use database systems. Now, there are two main approaches for design database those are used more than others: Relational and NoSQL. Relational databases are used for many years as main design pattern for data storage. But in 21th century, some concepts like Web2 make problems for this approach. Increasing size and amount of data based on user defined information and social networks with tremendous huge data sets lead database designers to NoSQL. NoSQL databases handle horizontal scaling and support dynamic change of values. It is schemaless and allow designers to construct various structures according to data objects. One of the challenges in data storage is storing and retrieving images. In relational databases, we can only save name of image in table and must store image file in separate folder out of our database on storage device and then retrieve it by matching name in table and name of file in folder. In Key-Value store systems like Redis that is one of methods of NoSQL, we can store file in database and retrieve it from. In this paper, we compare these two approaches by attending to memory and time consuming of each methods and analyze the results for present a suitable algorithm to storing and retrieving an image in a database. We use Python programming for interact with database systems.

[Ehsan Azizi Khadem, Emad Fereshteh Nezhad. **Storing and Retrieving Images by Relational Data Bases vs Key-Value store.** *Rep Opinion* 2016;8(2):9-16]. ISSN 1553-9873 (print); ISSN 2375-7205 (online). http://www.sciencepub.net/report. 2. doi:10.7537/marsroj08021602.

Keywords: Relational Data Bases, Key-Value Store, Storing and Retrieving Image

1. Introduction

A software is not imagine without database. Relational databases have used for many years with designing tables, views, triggers, store procedures and etc. This approach has a strong theory in its backbone. The ER model consists of Entity and Relationship concepts generate tables and their dependencies. But for some data structures it cannot works accurately [1]. For example when we want to save a file in our database, tables can only save its name and can't save whole data of file. In the other words, if we want to save user profile and picture is one of profile information, we must save picture in table of user and the picture must be saved in a folder on hard disk. When we want to retrieve of user profile, we should extract name from table and files from folder then match name of file by name in table. Furthermore, interaction between users like friendship and chat is very difficult to implement in relational databases. In recent years we against new approach in database design called NoSQL [2]. It is very flexible and schema less and has horizontal scaling. There are several categories for design NoSQL databases like document based, key-value stored and graph based. NoSQL databases can save a file as a simple data in their storage units and handle complex relationship between data structures with exciting techniques [3]. In this paper, we use Redis that is a key-value storage NoSQL database for saving images. It can save and retrieve images as a data in itself and not required to save it in separate folder. Furthermore we use MySQL from relational database tools to compare saving and retrieving images in relational and NoSQL with attending to time and size consuming. We use Python programming as an application to manipulate data in MySQL and Redis[4].

1. Relational Data Base: MySQL

Relational model offers a very mathematically-adapt way of structuring, keeping and using data. It began in 1970 and develops for several years because of its powerful model to interpret the operational environment. It is based on three main concept: entity, attribute and relationship. Entity is an object or anything that has a role in an environment which we must design a database for it. For example, in university, student is an entity. Course is too. Attributes are properties that present details about entity. A Student has attributes like name, address, degree, major, etc. Relationships are interactions between entities. A relationship occurs between an entities with itself or between two or more entities those interacting each other. For two entities student and course, selection is a relationship. One student can select one or more courses and a course can be selected by one or more students. A relationship has its own attributes. Selection relationship between

student and course has some attributes: year of selection, term of selection and mark that a student gets from a course. In the other words, relationship is an entity that called relational entity. Relational model has several tools to implement. One on the most popular tools of this model is MySQL. MySQL is a RDBMS. RDBMS is a Relational Database Management System; A software that enables you to implement a database with tables, columns and indices guarantees the relational integrity between rows of various tables, updates the indices automatically and interprets an SQL query and combines information from various tables [1]. MySQL is a fast, easy to use RDBMS being used for many small and big businesses. MySQL is becoming so popular because of many reasons: MySQL is released under an open source license. It is very powerful program in its own right and handles a large subsets of the functionality of the most expensive and powerful database packages. It uses a standard form of well-known SQL data language and works on many platforms and interact with several programming languages like Python, PHP, Java, C, etc. It is very fast even with large data sets. It supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but it can be increased to 8 TraBytes. It is customizable. To save images in MySQL, we create a database named "compareredismysql" in it. Then we create a table named "imageinfo" with two columns. One column named "id", is an integer unique number assigned to each image. Another column named "name" that contains name and postfix of image. It is a primary key for our table. When we save an image, its name saved in column "name" and an integer number assigned to it and saved in column "id". But we cannot save a file in table. We can only save its name and assign an id to it but cannot save its content and must create a folder named "uploads" and save image file there with name match to name in "imageinfo" table. When we want to retrieve images, muse extract name of each image from table, match it to name of images that saved in folder "uploads" then show image to user. It wants two referring by application to folder and database.

2. NoSQL: Redis

NoSQL databases are fast becoming the standard data platform for applications that make heavy use of telecommunication or internet enabled devices (browser based, sensor-driven or mobiles) as a front-end. The term NoSQL was first used in 1998 for a relational database that omitted the use of SQL and picked up again in 2009[5][6]. The computer word article summarize reasons commonly given to develop and use NoSQL data stores. First, avoidance of unneeded complexity. Relational databases provides a variety of features and strict data consistency. But this rich features set and the ACID properties implemented by RDBMSs might be more than necessary for particular applications and use cases. Second, high throughput. Some NoSQL databases provide a significantly higher data throughput than traditional RDBMSs. Third, horizontal scalability and running on commodity hardware [7][8]. In contrast to relational database management systems, most NoSQL databases are designed to scale well in the horizontal direction and not reply on highly available hardware. Machines can be added or removed (or crashed) without canceling the same operational efforts to perform sharding in RDBMS cluster-solutions. Forth, avoidance of expensive object-relational mapping. Most of NoSQL databases are designed to store data structures that are either simple or more similar to the ones of object oriented programming languages compared to relational data structures. Fifth, complexing and cost of setting up database clusters [9][10]. NoSQL databases are designed in a way that pc clusters can be easily and cheaply expand without the complexity and cost of sharding. Sixth, compromising reliability for better performance [11]. Because of generating several backups of data in nodes in NoSQL, it is more reliable system than relational. Seventh, movement in programming languages and development frameworks [12]. Eighth, requirements of cloud computing consist of high until almost ultimate scalability in horizontal direction and low administration overhead [13]. NoSQL have three main categories: document store, key-value store and graph based. Each of these categories has their own tools to implement a DBMS [14]. Redis is a key-value store tool often described as an in memory persistent. Redis docs hold all data in memory and it does write that out to disk for persistence, but it's much more than a simple key-value store [15]. Redis exposes five different data structures, only one of them is a typical key value structure. If we were to apply this data structure concept to the relational world, we could say that databases expose a single data structure tables [16]. Tables are both complex and flexible. There isn't much you can't model, store or manipulate with tables. Specially, not everything is as simple, or as fast, as it thought to be. If we want to use specific data structures to specific problems, Redis is a suitable approach. If we are dealing with scalars, lists, hashes, or sets, we can store them as scalars, lists, hashes, and sets by Redis. In Redis, we can save and retrieve an image as a byte stream. We can set its name as a key and its content as a value. By a single "set" statement, we can save it and by a "get" statement retrieve it.

3. Application: Python

Python is a general purpose, interpreted, interactive, object oriented and high level programming language. Python was designed to highly readable which uses English keywords frequently where as other languages use punctuation and it has fewer syntactical constructions than other languages. It is interpreted and processed at runtime by the interpreter and you do not need to compile your program before executing it. It is interactive and you can actually sit at a Python prompt and interact with the interpreter directly to write your programs. It is object oriented and supports object oriented style or technique of programming that encapsulates code within objects [4]. It is a great language for the beginner programmers and supports the development of a wide range of applications from simple text processing to web browsers to games. It connects to MySQL by pymysql library. This library is a connector that provides the ability to write and execute SQL statements in Python. It can connect to Redis too by "redis" library. We can write Redis statement in Python for interacting with DBMS. Python has a library named PIL (Python Imaging Library) that adds image processing capabilities to its interpreter. The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

4. Algorithm and Implementation:

We want to compare size and time of saving and retrieving images in Redis and MySQL. Keep in mind Redis stores data in RAM and we must execute commands and queries that their data overloading less than capacity of RAM. We have a picture in JPEG format. Size of the image is 800 kilo bytes. We run storing and retrieving programs for 10 to 7000 times, then compare the size and the time of executing each codes.

First, we execute the Python code for saving images in MySQL database:

```
import tornado.httpserver, tornado.ioloop,
tornado.options, tornado.web, os.path, random,string
import pymysql
import os,time
conn = pymysql.connect(host="localhost",
user="root", passwd="" ,
db="compareredismysql")
cur = conn.cursor()
def getFolderSize(folder):
    total_size = os.path.getsize(folder)
    for item in os.listdir(folder):
        itempath = os.path.join(folder, item)
        if os.path.isfile(itempath):
            total_size += os.path.getsize(itempath)
        elif os.path.isdir(itempath):
            total_size += getFolderSize(itempath)
    return total_size/(1024*1024)

class Application(tornado.web.Application):
    def __init__(self):
        handlers = [
            (r"/", IndexHandler),
            (r"/upload", UploadHandler)
        ]
        tornado.web.Application.__init__(self, handlers)

class IndexHandler(tornado.web.RequestHandler):
    def get(self):
        self.render("tornadoUpload.html")

class UploadHandler(tornado.web.RequestHandler):
    def post(self):

        try:
            file1 = self.request.files['file1'][0]
        except:
            file1=None
        original_fname = file1['filename']
        m = original_fname.split('.')

        sql = "SELECT table_schema
,Round(Sum(data_length + index_length) , 1)
FROM   information_schema.tables WHERE
table_schema='compareredismysql' "
        cur.execute(sql)
        for r in cur:
            start_size2 = int(r[1])/(1024*1024)
            print("start_size2 db= ",start_size2)
        start_time = time.time()
        start_size1 = getFolderSize("static/uploads/")
        print("start_size1 folder= ",start_size1)
        for i in range(7000):
            sql = "INSERT INTO imageinfo (id,name)
VALUES (%s, %s)"
            n = str(i)+'.'+m[1]
            cur.execute( sql, ( i,n) )
            output_file = open("static/uploads/" + n,
'wb')
            output_file.write(file1['body'])
        conn.commit()
        print("MySQL Time = " , time.time()-
start_time)
        size1 = getFolderSize("static/uploads/")-
start_size1
        print("size1 folder = ",size1)
        sql = "SELECT table_schema
,Round(Sum(data_length + index_length) , 1)
FROM   information_schema.tables WHERE
table_schema='compareredismysql' "
        cur.execute(sql)
        for r in cur:
```

```
            size2 = (int(r[1])/(1024*1024)-start_size2)
            print("size2 db = ",size2)
        conn.close()
        print ("Size: " + str(size1+size2))
        self.render('image.html',n=n)

settings = {
'template_path': 'templates',
'static_path': 'static',
"xsrf_cookies": False

}
application = tornado.web.Application([
  (r"/", IndexHandler),
        (r"/upload", UploadHandler)

], debug=True,**settings)

print ("Server started.")
if __name__ == "__main__":
   application.listen(8888)
   tornado.ioloop.IOLoop.instance().start()
```

Second, we execute the Python code for saving images in Redis:

```
import tornado.httpserver, tornado.ioloop,
tornado.options, tornado.web, os.path, random, string

from PIL import Image

import redis

from io import BytesIO

import base64

import time

import string

def converttostr(x):

   return str(x)

class Application(tornado.web.Application):

   def __init__(self):

      handlers = [

         (r"/", IndexHandler),

         (r"/upload", UploadHandler)

      ]

      tornado.web.Application.__init__(self, handlers)

class IndexHandler(tornado.web.RequestHandler):

   def get(self):

      self.render("tornadoUpload.html")

class UploadHandler(tornado.web.RequestHandler):

   def post(self):

      try:

         file1 = self.request.files['file1'][0]

      except:

         file1=None

      original_fname = file1['filename']

      #im = Image.open(file1)

      #output = BytesIO(file1['body'])

      #im.save(output,im.format)

      r = redis.StrictRedis(host='localhost')

      redisdbinfo1=r.info()

      size1 = (redisdbinfo1['used_memory_human'])

      print("size1 : ",size1)

      size1 = float(size1[0:(len(size1)-1)])

      m = original_fname.split('.')

      start_time = time.time()

      for i in range(7000):

         output = BytesIO(file1['body'])

         s = converttostr(i)

         n = s+'.'+m[1]

         r.set(n, output.getvalue())

      print("Redis Time = " , time.time()-start_time)

      h=r.get(n)
```

```
        output = BytesIO(h)
        s = output.getvalue()
        str = base64.b64encode(s)
        output.close()
        self.render("image.html",img_tag = str)
        redisdbinfo2=r.info()
        size2 = (redisdbinfo2['used_memory_human'])
        print("size 2 : ",size2)
        size2 = float(size2[0:(len(size2)-1)])
        print("total size : ",size2-size1)
settings = {
'template_path': 'templates',
'static_path': 'static',
"xsrf_cookies": False
}
application = tornado.web.Application([
  (r"/", IndexHandler),
        (r"/upload", UploadHandler)
], debug=True,**settings)
print ("Server started.")
if __name__ == "__main__":
   application.listen(8000)
   tornado.ioloop.IOLoop.instance().start()
```

For saving, these results are generated:

Table 1: Time for saving

Rows	MySQL	Redis
10	0.31	0.061
100	0.30	0.43
1000	14.05	4.31
2000	27.31	8.35
3000	60.16	12.42
4000	86.64	16.83
5000	147.80	21.05
6000	161.91	25.41
7000	204.46	26.64

Table 2: Size for saving

Rows	MySQL	Redis
10	8.067m	9.05 m
100	80.68	81.65 m
1000	806.97	807.82 m
2000	1.61G	1.58G
3000	2.42G	2.36G
4000	3.22G	3.15G
5000	4.03G	3.94G
6000	4.84G	4.73G
7000	5.64G	5.52G

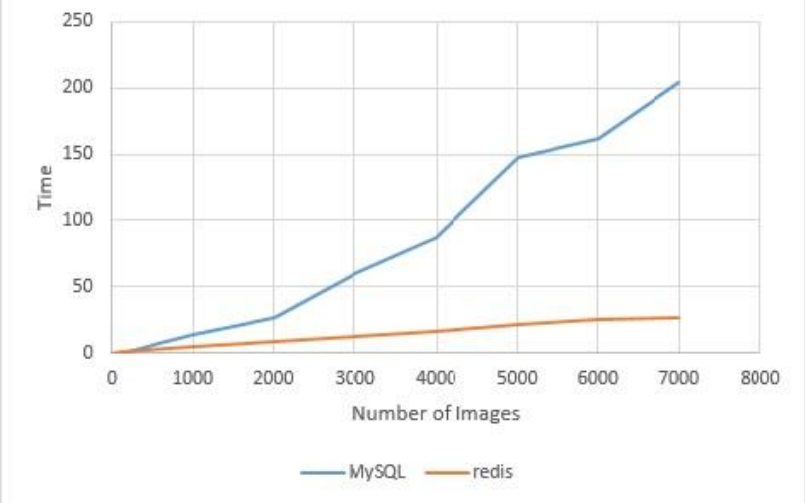

Figure 1: Time for saving in Redis and MySQL (Seconds)

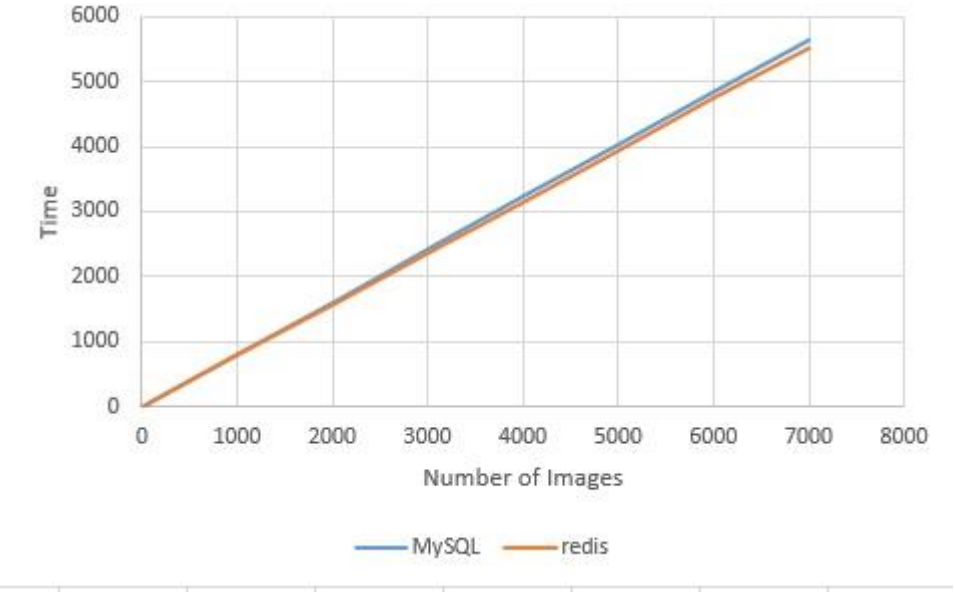

Figure 2: Size for saving in Redis and MySQL (Mega Bytes)

Now, we execute retrieval Python code in MySQL:

```
from PIL import Image

import time,pymysql,os

import redis

from io import BytesIO

import shutil

#from tkinter import PhotoImage

conn = pymysql.connect(host="localhost",
user="root", passwd="" ,
db="compareredismysql")

cur = conn.cursor()


for i in range(7000):

        sql = "INSERT INTO imageinfo (id,name)
VALUES (%s, %s)"

        n = str(i)+'.jpg'

        cur.execute( sql, ( i,n) )

        name = "static/uploads/" + n

        shutil.copy2('C:/Desert.jpg', name)

conn.commit()

t1 = time.time()

for j in range(7000):

   sql = "SELECT name FROM imageinfo
WHERE id=%s"

   cur.execute(sql,(j))

   for r in cur:

      name = "static/uploads/" + r[0]

      im = Image.open(name)


t2 = time.time()-t1

print("Image Retrieval in MySQL : ",t2)


im.show()

conn.close()
```

Then we execute retrieval Python code in Redis:

```
from PIL import Image

import redis

from io import BytesIO
```

```
import time

im = Image.open('C:/Desert.jpg')
output = BytesIO()
im.save(output,'JPEG')

r = redis.StrictRedis(host='localhost')
for i in range(7000):
   k = str(i)+'.jpg'
   r.set(k, output.getvalue())
t1 = time.time()
for j in range(7000):
   k = str(j)+'.'+'jpg'
   h=r.get(k)
   output = BytesIO(h)
t2 = time.time()-t1
print("Image Retrieval in Redis : ",t2)
i = Image.open(output)
i.show()
output.close()
```

For retrieving, these results are generated:

Table 3: Time for retrieving

Rows	MySQL	Redis
10	0.007	0.01
100	0.05	0.03
1000	29.18	0.37
2000	8.67	0.86
3000	4.07	1.21
4000	3.21	1.65
5000	4.54	1.98
6000	15.59	2.18
7000	8.32	2.69

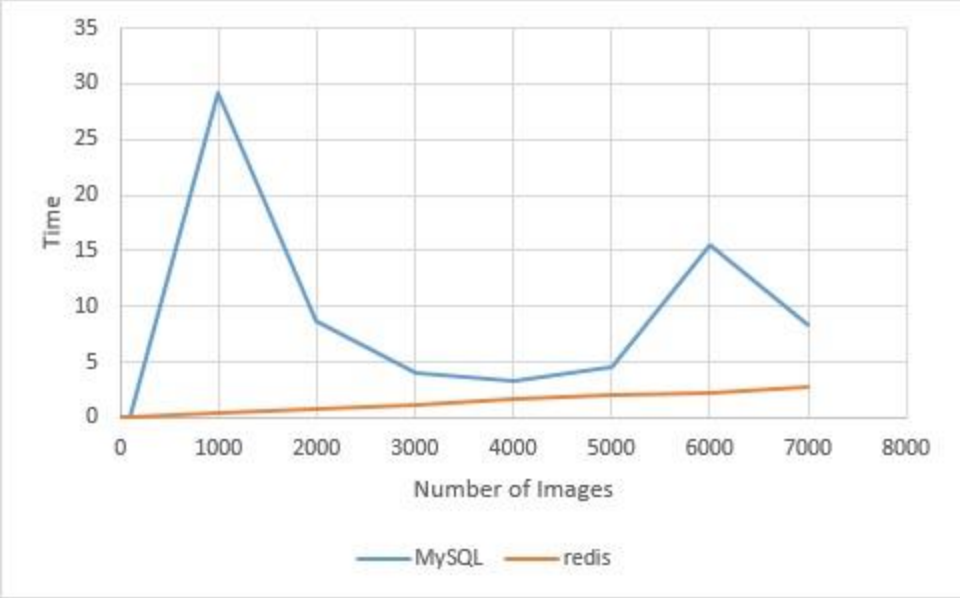

Figure 3: Time for retrieving in Redis and MySQL (Seconds)

For saving, Redis is very faster than MySQL and it consumes less memory on disk. But take care that if data volume is higher than RAM capacity, Redis has lower speed than MySQL.

In retrieving, Redis is faster than MySQL but not very much. When we study both saving and retrieving, mine that Redis has better functionality in time and memory consuming. In storing, Redis and MySQL have predictable performance and their increasing have a static ratio but in retrieving MySQL has tolerant time consuming and it is not predictable, Redis has a normal growing and it seems better for large image databases.

Conclusion:

We know that NoSQL are faster and simpler than relational databases for large datasets if data is text or has simple structure. When data structure is complex like image, key-value store is a good choice for storing our data. In order to results of our implementation, Redis has better performance for saving and retrieving images in database than MySQL and we can predict its behavior with growing data volume. When we have sufficient RAM capacity, Redis is suitable. In future, we can study saving and retrieving image and other file formats with other NoSQL tools.

References:

1. Acharya, S., Carlin, P., Galindo-Legaria, C.,Kozielczyk, K., Terlecki, P., and Zabback, P.2008. Relational support for flexible schemascenarios. Proc. VLDB Endow. 1, 2 (Aug. 2008), 1289-1300. DOI = http://doi.acm.org/10.1145/1454159.1454169.
2. Bogdan Tudorica, Bucur Cristian – A comparison between several NoSQL databases with comments and notes,The proceedings of ”2011 – Networking in Education and Research” IEEE International Conference, June 23, 2011– June 25, 2011, Alexandru Ioan Cuza University from Iasi.
3. Python Software Foundation, 2012. Python/C API Reference Manual. Python Software Foundation.
4. Message in Redis mailing list http://groups.google.com/group/redisdb/msg/ca398a90ea78bfc5.
5. Barabasi, A. (2003). *Linked: How everything isconnected to everything else and what it means*. New York: Plume.
6. Bishop, S., Helbing, D., Lukowicz, P., & Conte, R. (2011). FuturICT: FET flagship pilot project. *Procedia Computer Science, 7,* 34–38.
7. Cointet, J. P., & Roth, C. (2009). Sociosemantic dynamics in a blog network.*International Conference on Computational Science and Engineering*. doi:10.1109/CSE.2009.105.
8. Conte, R., Gilbert, N., Bonelli, G., & Helbing,D. (2011). FuturICT and social sciences: BigData, big thinking*Zeitschrift für Soziologie, 40,*412–413.
9. Snijders, C., & Weesie, J. (2009). Reputation in an online programmers' market. In K. S. Cook, C. Snijders, 23.V.Buskens, & C. Cheshire (Eds.), *Trust and reputation* (pp. 166–185). New York: Russel Sage Foundation.
10. Prof. Walter Kriha, (2012). NoSQL Databases, (pp. 85-110).
11. Redis home page http://code.google.com/p/redis.
12. Active Redis home page http://www.activeredis.com.
13. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G.,Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. 2007. Dynamo: amazon's highly available key-value store. In Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles (Stevenson, Washington, USA, October 14 - 17, 2007). SOSP '07. ACM, New York, NY, 205-220. DOI = http://doi.acm.org/10.1145/1294261.1294281.
14. Agneeswaran VS, Tonpay P, Tiwary J (2013) Paradigms for realizing machine learning algorithms. Big Data 1(4):207–214.
15. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Spark SI (2010) Cluster Computing with Working Sets. In:Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing., pp 10–10.
16. Berkeley Data Analysis Stack. https://amplab.cs.berkeley.edu/software.

2/13/2016