Sādhanā Vol. 41, No. 8, August 2016, pp. 837–846
DOI 10.1007/s12046-016-0523-6

Weighted Moore–Penrose generalized matrix inverse: MySQL vs. Cassandra database storage system

DANIJELA MILOŠEVIĆ[1], SELVER PEPIĆ[2], MUZAFER SARAČEVIĆ[3,*] and MILAN TASIĆ[4]

[1]Faculty of Technical Sciences Čačak, University of Kragujevac, 32000 Čačak, Serbia
[2]Information Technology in Engineering, The School of Mechanical Engineering of Applied Study, 37240 Trstenik, Serbia
[3]Department of Computer Science, University of Novi Pazar, 36300 Novi Pazar, Serbia
[4]Faculty of Science and Mathematics, University of Niš, 18000 Niš, Serbia
e-mail: danijela.milosevic@ftn.kg.ac.rs; p_selver@yahoo.com; muzafers@uninp.edu.rs; milan1t@ptt.rs

MS received 4 November 2015; revised 10 March 2016; accepted 15 March 2016

Abstract. The research in this paper refers to two areas: programming and data storage (data base) for computing the weighted Moore–Penrose inverse. The main aim of this paper analysis of the execution speed of computing using PHP programming language and data store. The research shows that the speed of execution gives considerable difference between the Procedural programming and Object Oriented *PHP* language, on the middle layer in the three tier of the web architecture. Also, the research concerning the comparison of relation database system, *MySQL* and *NoSQL*, key value store system, *ApacheCassandra*, on the database layer. The CPU times are compared and discussed.

Keywords. Generalized inverse; weighted Moore–Penrose inverse; *PHP* programming; *MySQL* database; *NoSQL Cassandra* database storage system.

1. Introduction

The main aim and motivation is to achieve improved results concerning the speed of computing weighted Moore–Penrose (MP) inverse, and in particular its storage. In our previous paper [1] developed a client/server-based model for computing the weighted Moore–Penrose inverse (MP inverse) using the partitioning method as well as for storage of generated results. The web application is developed in the Procedural programming *PHP* and MySQL environment.

The research in this paper shows that the speed of execution gives considerable difference between the Object-oriented and Procedural programming *PHP* language for computing the weighted Moore–Penrose inverse, on the middle layer in the three tier of the web architecture. Also, the research concerning the comparison of relation database system, *MySQL* and key value store system, *Cassandra*, on the database layer. In our article [2] we have stated the conditions which characterize the weighted Moore–Penrose inverse.

The *Cassandra* system was designed to run on cheap commodity hardware and handle high write throughput while not sacrificing read efficiency. *Cassandra* does not support a full relational data model, but instead, it provides clients with a simple data model that supports dynamic control over data layout and format. The *Cassandra* system indexes all data based on primary key. The data file on disk is divided into a sequence of blocks. Each block contains at most 128 keys and is demarcated by a block index. The block index captures the relative offset of a key within the block and the size of its data. When an in-memory data structure is dumped to disk a block index is generated and their offsets are written out to disk as indices.

A typical read operation always looks up for data first in the in-memory data structure. If found the data is returned to the application since the in-memory data structure contains the latest data for each key. If it is not found then we perform disk I/O against all the data files on disk in reverse time order. This scenario with read operation in-memory data structure we consider with *PHP*/*MySQL* in ALDF approach. By applying Cassandra system we have all that at one place in the most efficient way.

The *Cassandra* used development by some of the biggest properties on the Web, including *Facebook*, *Twitter*, *Cisco*, *Rackspace*, *Digg*, *Cloudkick*, *Reddit*, and more. The huge difference between *MySQL* and *Cassandra* is in writes/reads (R/W) time frames, and it will be discussed in section 4.

SQL is the standard language used for query and analysis of data in relational *DBMS* [3]. Unfortunately, *SQL* has no

*For correspondence

vector and matrix computation capabilities that are essential in multidimensional (multivariate) statistics, machine learning and data mining. The *SQL* constructions and the *SQL* primitives for data mining are proposed in [4, 5], but such constructions do not offer adequate and flexible matrix manipulation capabilities. Implementation of certain vector and matrix operators based on programming User-Defined Functions (*UDFs*) are further studied in [6]. *UDFs* represent a *C* programming interface that allows the definition of scalar and aggregate functions which can be used in *SQL*. *UDFs* have several advantages and limitations. An *UDF* allows fast evaluation of arithmetic expressions, memory manipulation, using multidimensional arrays and exploiting all *C* language control statements.

In this paper we presented *UDF* defined in [1] in object-oriented approach. The object-oriented programming (*OOP*) is a kind of programming added to *php*5 which makes building complex, modular and reusable web applications much easier. With the release of *php*5, *PHP* programmers finally had the power to code. Like *Java* and *C#*, *PHP* finally had a complete *OOP* infrastructure. *PHP* provides the basics to support object-oriented programming. Among other things, the *OOP* syntax in *PHP* allows for programmer-defined classes with member variables and member data and offers single inheritance, constructor functions, object serialization, and functions for introspection. The *OOP* extension is usable, fairly mature, pretty stable, and widely used. It provides an extra layer of organization that can be helpful when maintaining complex code and offers a nice way to package code for distribution and reuse.

The matrix operations underlie a lot of linear equations and statistics. They are used extensively in games programming, and bitmap manipulation. We have implemented various matrix operations by using *SQL*. We created a table to store two-dimensional arrays containing the real number. We can also create any number of matrices in this array, which cuts down on the chore of creating tables. It also means that functions and procedures can be hardwired to a common matrix table. Interesting code for matrix *SQL* operations can be found in [7]. A graphical *SQL* query generator and query operators are disclosed in [8]. The query tool embeds matrix objects that are used for building and transforming *SQL* based queries, views, data cubes and other relations including "virtual" or calculated relations.

The structure of the paper is as follows. The paper proposes two new approaches. The object-oriented approach of the matrix computation is described in the second section, with classes, variables and methods. In section 3 is presented OOP and MySQL approach of weighted MP inverse, while in section 4 is presented OOP and NoSQL Cassandra approach of weighted MP inverse. In the third section we presented different *string* data types in relational-*MySQL* database. The power of the *PHP* to develop a free web based application is used in the pseudo-inverses computation. The *Cassandra* data model is presented in section 4, with *Keyspace*, column families and appropriate procedures. A few illustrative examples and comparative studies are presented in the last section followed with discussion.

2. The object-oriented programming approach

The use of object-oriented (*OO*) technology remains a challenge. Systems with embedded *OO* technology require high execution speed and significant high memory constraints. Object-oriented model severely conflicts with the highly static model that is required in high integrity systems to meet the goals of time- and memory-boundedness assurance, and of determinism in data transformation due to code operation.

Because of demand for improved *OOP* support, the entire object model was completely redesigned for *PHP*5, adding a large amount of features and changing the behavior of the base "object" itself. *OOP* turns out to be a very simple idea, which leads to all sorts of more complicated elaborations. *OOP* approach rather than creating data structures on the one hand and code on the other, suppose we reorganize everything so that associated pieces of code and data are bundled together [9].

In order to store functions in our data, *OOP* approach lets us define these combinations as genuinely new datatypes that the language supports like any other type. In our case we defined the class named matrix (which specifies the different kinds of data and functions that every self-respecting matrix should have). The object (or instance) refers to any individual instance of the type. After we define class matrix, we can make a number of matrix objects.

In this section we cover the *PHP* matrix computation, for *OOP* approach, from the ground up, with the example of weighted Moore–Penrose inverse. In the Appendix part we presented the classes that we used in the application (see Class 1, 2 and 3 in Appendix). The next procedure, method of the class Matrix, present *OOP* approach of the weighted Moore–Penrose inverse. During the execution of an object's method, a special variable called $*this* is automatically defined, which denotes a reference to the object itself. See Procedure 1 in Appendix.

In general, the way to refer to a member variable from an object is to follow a variable containing the object with → and then the name of the member. After we have a class definition, the default way to make an instance of that class is by using the *new* operator. When creating an object (using the new keyword), the returned value is a handle to an object or, in other words, the id number of the object. See Procedure 2 in Appendix. Arguments, in Procedure 2, $*matrix*1, $*matrix*2 and $*matrix*3 are presented in the view of two-dimensional array, as given in [1].

3. *MySQL* data model of weighted MP inverse

Nowadays, relational databases are the most frequently used type of database. *SQL* database is based on the very simple logical structure. Each database contains a number of tables, made up of carefully defined columns, and every entry can be thought of as an added record or row.

Four data manipulation statements are supported by every *SQL* server: *select*, *insert*, *update*, and *delete*. They constitute an extremely high percentage of all operations within a relational database. These four *SQL* statements manipulate only with database values, not with the structure of the database itself.

The actual *PHP* functions used to create *MySQL* databases are trivial compared to the *MySQL* data structure statements that are passed in those functions. The general rule is to use data type that will adequately meet the needs of particular column in database. *MySQL* is known for having compact types that are good for things like 0/1 values or very large types that can store up to 4GB of data in one field [9].

There are three buckets of *MySQL* data types: numeric types, date and time types, and string (or character) types. In our application we compare string data types: *blob*, *varchar* and *text*. *Blob* is the *string* data type used for binary data. Any queries on a *blob* type result in a case-sensitive return. This is opposite to the behavior of the *text* type. *Text* is the *string* data type used for character data. It works much like the *blob* data type, so queries upon the *text* type will return case insensitive values. The *varchar* data type works like the *char* type except for its data-storage method. The *varchar* type removes all trailing whitespace from inserted data. Designated by *varchar*(*M*) where *M* designates the field length, the maximum length of the *varchar* type is 255 characters [9]. For some systems, in considering space requirements, the *varchar* type should be used as opposed to the *char* type.

Table 1 shows the three enumerated *MySQL* data types and their possible values. M stands for the maximum number of digits displayed, and it is optional. It should be noted that the CPU time for insert and searching data is much shorter when matrix stored in the database in the *R* format [1], because we use it in that form.

The description of different ways of accessing data stored in *MySQL* database is further presented. Procedure 3 and Procedure 5 are showing the insert and select of data when using *blob MySQL* data type, while Procedure 4 and Procedure 6 are related to *char* and *text MySQL* types of data. Analysis of the execution speed of insert and read for different *MySQL* data types is presented in detail in section 5 (see Procedures 3, 4, 5 and 6 in Appendix). Aside from testing different types of *MySQL* data, our research is also considering $OOP + MySQL$ and $OOP + Cassandra$ approach of Weighted Moore–Penrose inverse computation.

We provided certain procedures, applicable to dense matrices, that are used in our implementation. Testing results together with the best possible combination of matrix presentation (procedural or object oriented), on the middle tier, as well as the storage system (*MySQL* or *Cassandra*), on the database tier, are presented in section 5. Implementation of an arbitrary matrix operation using $OOP + MySQL$ approach can be described by the following algorithm, while algorithm for $OOP + Cassandra$ approach is presented in the next section.

Algorithm 1 *OOP* and *MySQL* approach of weighted Moore - Penrose inverse.

Require: Input matrices.

```
$file = new UploadFile(), Class 8.1
$dataManipulation = new DataManipulation(), Class 8.2
$matrix_input = $dataManipulation → search_input(),Class 8.2
if $matrix_input == TRUE then
    $result = $dataManipulation → search_result(),Class 8.2
    if $result == TRUE then
        goto Step 16
    else
        goto Step 14
    end if
else
    $database → enter_matrix_in(), Class 8.2 goto Step 14
end if
$matrix = new matrix; $matrix → WeightedInverse(), Class 8.3
$dataManipulation → enter_weighted(), Class 8.2
return $dataManipulation → display_weighted(), Class 8.2
```

4. *NoSql Cassandra* model of weighted MP inverse

Apache Cassandra is a free, open source, distributed data storage system that differs sharply from relational database management systems. Science and web affine companies were engaged in the research database which has led to the

Table 1. MySQL data types.

MySQL data types		
Name	Storage size	Using
varchar(*M*)	Up to M bytes	Variable in length. M must be ≤ 255
blob	Up to 64KB	blob is case-sensitive for sorting and comparison
text	Up to 64KB	text is case-insensitive
longblob	Up to 4GB	longblob is case-sensitive for sorting and comparison
longtext	Up to 4GB	longtext is case-insensitive

emergence of a great variety of alternative databases. That leads phenomenon new datastores known as *NoSQL* databases (key-value-stores, document databases, column-oriented databases), "used to describe the increasing usage of non-relational databases among Web developers" [10]. The emergence of key - value stores has been heavily influenced by Amazon's Dynamo, although key-value-stores have existed for a long time.

Model of relation database, such as *MySQL*, was difficult at first for all but the most advanced computer scientists to understand, until broader adoption helped to make the concepts much clearer. Using the database built around this model required learning new terms and thinking about data storage in a different way. The massive use by businesses and government agencies, is the result of fast capability for processing thousands of operations per second. The new model for database was threatening, chiefly for two reasons. First, the new model was very different from the old one, which it pointedly controverted. Also, it was threatening because it can be hard to understand something different and new.

Cassandra is very powerful system in Facebook's inbox search. It has become "the hands down winner for transaction processing performance," with a deserved reputation for reliability and performance at scale [11]. The *Cassandr*a is the cluster or the ring that assigns data to nodes in the cluster by arranging them in a ring. Typically, an instance of *Cassandra* presents distributed multidimensional map indexed by a key [12]. A table is structured by the following dimensions:

- Rows which are identified by a string-key of arbitrary length.
- Column Families which can occur in arbitrary number per row. As in *Bigtable*, column-families have to be defined in advance. The number of column-families per table is not limited; however, it is expected that only a few of them are specified. A column family consists of columns and supercolumns which can be added dynamically to column-families and are not restricted in number.
- Columns have a name and store a number of values per row that are identified by a timestamp (like in *Bigtable*). A table cannot be thought of as a rectangle, because each row in a table can have a different number of columns.
- Supercolumns have a name and an arbitrary number of columns associated with them. Also, the number of columns per super-column may differ per row.

A cluster is a container for keyspaces. A keyspace is the outermost container for data in *Cassandra*, corresponding closely to a relational database. Like a relational database, a keyspace has a name and a set of attributes that define keyspace-wide behavior. A good idea is to create a single keyspace per application. It is certainly an acceptable practice, but it is perfectly fine to create as many keyspaces.

A column family is a container for an ordered collection of rows, each of which is itself an ordered collection of columns. In the relational databases we have physically creating database from a model. Database has the name of the database (keyspace), the names of the tables (similar to column families), and then we define the names of the columns that will be in each table. For several good reasons a column family is not like a relational table. First of all, the *Cassandra* is considered schema-free because although the column families are defined, while the columns are not. Also, we can freely add any column to any column family at any time. Second, a column family has two attributes: a name and a comparator. The comparator value indicates how columns will be sorted: byte, UTF8, or other ordering.

The *Cassandra*, *API*, exposed to client-applications, consists of the three simple methods: insert(table; key; *rowMutation*); get(table; key; *columnName*) and delete(table; key; *columnName*) where the *columnName* argument of the get and delete operation identifies either a column family, or a *supercolumn* within a column family, or column family as a whole. Connection between *PHP* and *API* of *Cassandra* is exposed via *API Thrift*. The programming language libraries for *PHP*, *Java*, *Ruby*, *Pyhton*, *C#*... are available for interaction with Cassandra. Implementation of an arbitrary matrix operation using *OOP* approach and *Cassandra* database can be described by the following algorithm.

The basic unit of data structure in the *Cassandra* data model is a column. A column is a triplet that contains a name, a value, and a clock. First of all, when designing a relational database, you specify the structure of the tables up front by assigning all of the columns in the table a name; later, when you write data, you are simply supplying values for the predefined structure [11].

Algorithm 2 *OOP* and *Cassandra* approach of weighted Moore - Penrose inverse.
Require: Input matrices.
1: $file = new UploadFile(), Class 8.1
2: getConnection(), Procedure 8.8
3: $exist_in = getColumnMatrixIn(), Procedure 8.10
4: **if** $exist_in == TRUE **then**
5: $exist_out = getColumnMatrixOut() Procedure 8.12
6: **if** $exist_out == TRUE **then**
7: **goto** *Step 16*
8: **else**
9: **goto** *Step 14*
10: **end if**
11: **else**
12: insertColumnMatrixIn(), Procedure 8.9 **goto** *Step 14*
13: **end if**
14: $matrix = new matrix; $matrix → WeightedInverse(), Class 8.3
15: insertColumnMatrixOut(), Procedure 8.11
16: **return** displayColumnMatrixOut(), Procedure 8.13

5. Experimental results

In order to compare two different programming for matrices computation (procedural and *OOP* approach), we were using the precise implementation of the weighted Moore–Penrose algorithm. Details concerning the implementation of the partitioning algorithm corresponding to the weighted Moore–

Penrose inverse in the view procedural programming can be found in [1], while *OOP* approach is presented in section 2.

Testing was performed on the local machine and from a client in a wireless network. We had an access to the web server using the infrastructure mode wireless networking with an access point. Testing was executed on the server machine with the following specification: *Windows edition*: *Windows Vista(TM) Ultimate*; Processor: *Intel(R) Pentium(R) Dual CPU T3200 @ 2 : 00GHz*; *Memory (RAM)*: 2940MB; System type: 32-bit Operating System; Free Softwares: *WampServer* version 2.2 which contains *PHP* 5.3.8, *MySQL* 5.5.16 and *phpMyAdmin* 3.4.5, *Apache Cassandra* 1.0.6.

5.1 *Object-oriented vs. procedural approach of weighted MP inverse*

The paper reports the specific approach of matrix calculating in the form of classes and objects. In the next table, we compare the CPU executing times for computing weighted Moore–Penrose inverse using procedural and object oriented programming approach. The efficiency comparison on the set of randomly generated test matrices is given in table 2.

This testing is performed for dense matrices with randomly chosen elements, and Test matrices (*A*_30_3, *A*_50_4, *F*_15_2, *F*_30_3, *F*_50_4, *S*_50_4, *S*_80_5) from [13]. In our application the CPU executing time, for AL computation, is slightly shorter when we have *OOP* approach (figure 1).

5.2 *Compare the time for different MySQL data types*

We obtain positive results compared to the results presented in [1], because we create instance of *class Matrix* - object,

Table 2. CPU time for computation of the weighted MP inverse using procedural and *OO* programming.

Arithmetic - Logic unit		
$m \times n$	PP approach	OOP approach
*A*_30_3	0.897 s	0.790 s
*A*_50_4	6.199 s	5.140 s
*F*_15_2	0.166 s	0.120 s
*F*_30_3	0.880 s	0.750 s
*F*_50_4	6.077 s	5.100 s
*S*_50_4	6.070 s	5.058 s
*S*_80_5	40.75 s	34.459 s
50×50	6.203 s	5.870 s
15×15	0.160 s	0.116 s
50×35	1.997 s	1.602 s
45×70	19.920 s	17.549 s
60×60	12.627 s	10.722 s

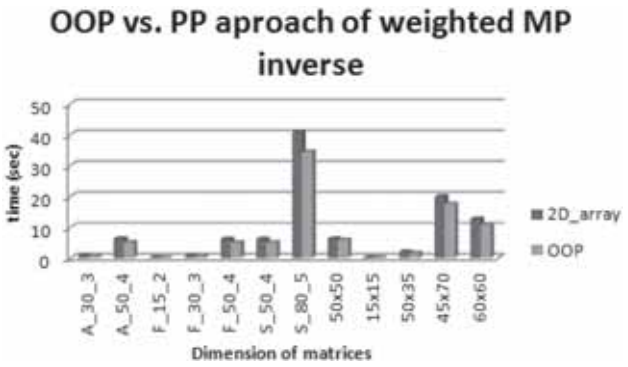

Figure 1. Computation of the weighted MP inverse using procedural and *OO* programming.

Table 3. The CPU time for insert matrices in the *MySQL* database with different data types.

MySQL 2D array (R format): INSERT			
$m \times n$	longtext	blob	varchar
30×30	0.151 s	0.078 s	0.106 s
50×50	0.162 s	0.079 s	0.117 s
60×60	0.156 s	0.075 s	0.119 s
70×70	0.162 s	0.079 s	0.122 s
80×80	0.165 s	0.068 s	0.140 s

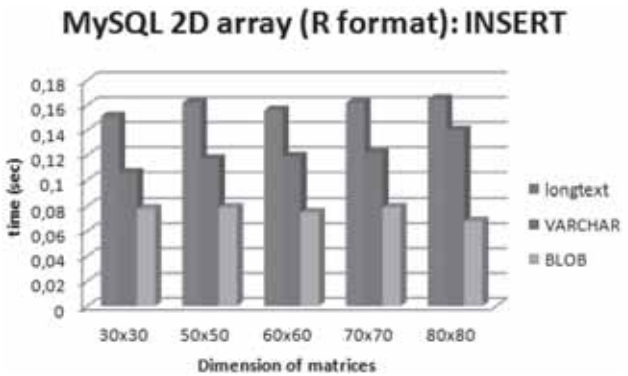

Figure 2. Insert data in different *MySQL* data types.

only once, and we read all the necessary information without additional calculations. In the next table, we compare the time for insert of data when we have different *MySQL* data types (table 3; figure 2).

We compare searching of data time in the *MySQL* with different data types (table 4; figure 3).

This testing is performed for dense matrices with randomly chosen elements. The best executing CPU time we have achieved during the insert and searching data operations in *MySQL* database with *blob* data type. We note that matrices are in the database in the *R* format, such as presented in [1].

Table 4. The CPU time for read data from *MySQL* database when we have different data types.

MySQL 2D array (R format): SELECT			
$m \times n$	longtext	blob	varchar
30 × 30	0.0089 s	0.0010 s	0.0085 s
50 × 50	0.0104 s	0.0017 s	0.0087 s
60 × 60	0.0145 s	0.0024 s	0.0104 s
70 × 70	0.0293 s	0.0025 s	0.0204 s
80 × 80	0.0444 s	0.0026 s	0.0324 s

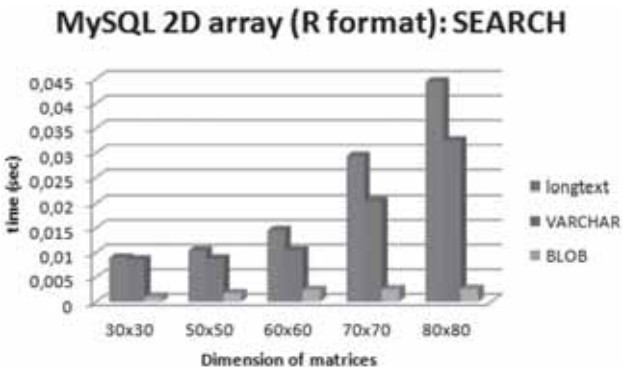

Figure 3. Read data of different *MySQL* data types.

Table 5. The CPU time for insert matrices in *MySQL* and *Cassandra*, when they are stored in the *R* format.

INSERT data			
$m \times n$	Number of records	DB MySQL	CASSANDRA
80 × 80	50	4.517 s	0.422 s
80 × 80	100	8.018 s	0.849 s
80 × 80	500	41.081 s	4.291 s
80 × 80	1000	84.889 s	8.126 s
80 × 80	5000	496.038 s	40.053 s

5.3 *MySql vs. NoSqlCassandra data model of weighted MP inverse*

We compare the time durations for insert matrices when they are stored in the database with two completely different approaches: relation database *MySQL* and key-value storing system, *Apache Cassandra* (table 5).

This testing is performed for dense matrices with randomly chosen elements, presented in the *R* format. The CPU executing time, in table 6, indicates the much better results in the case key value store system Cassandra.

Table 6. The CPU time for searching matrices, dimension 80 × 80 in the *R* format, from structural different type of databases.

Search of stored matrices (SELECT data)		
Number of matrices	R format+MySQL	R format+CASSANDRA
50	0.102 s	0.0293 s
100	0.166 s	0.0308 s
500	0.791 s	0.0276 s
1000	1.901 s	0.0267 s

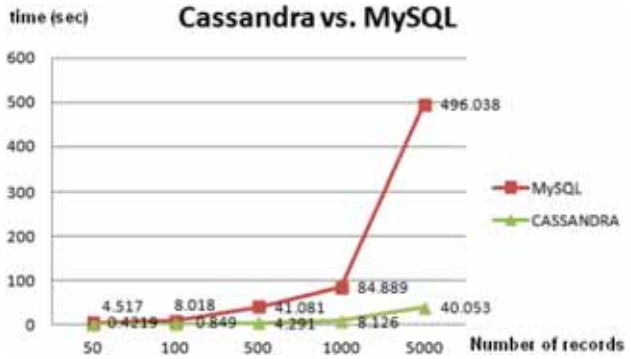

Figure 4. Insert matrices in *Cassandra* and *MySQL*.

In table 6 comparison of the searching time duration when matrices are stored in the relation database, *MySQL* [1], and *noSQL* database, *Cassandra* is shown.

Now we can define Keyspace named "MatrixComputation" and in Keyspace we define a Column Families named "Matrix" and define secondary index "dimidx" on field "dimension", and "opidx" on field "operation" (see Procedure 7, 8, 9, 10, 11, 12, and 13 in Appendix).

This testing is performed for dense matrices with randomly chosen elements and matrices from [13]. The CPU time for searching is much shorter when a matrix is stored in the key value storage system Cassandra (figures 4, 5).

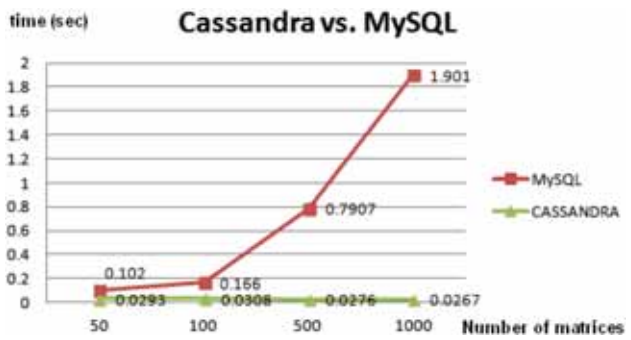

Figure 5. Reading matrices from *Cassandra* and *MySQL*.

6. Discussion and conclusion

Many studies from other fields of applied mathematics and computer sciences have shown that the OOP approach for effective implementation of similar algorithms is better. In our previous article we presented the implementation of the weighted Moore–Penrose inverse by procedural programming approach in PHP and MySQL database [1]. Previously, we have also stated the conditions which characterize the weighted Moore–Penrose inverse in our article [2].

Our main motivation was to provide better access compared to our previous work, as well as other studies, primarily in the field of matrix calculations. In this study, we propose better way to matrix computation on the middle tier in the three layer web architecture, along with the implementation and experimental support. Specifically, we provide a new approach for calculating generalized weighted Moore–Penrose matrix based on object-oriented programming (OOP) with the technique of storing with *CassandraNoSQL* database approach.

We have tested and compared the speed of insert and search data operations in relation database and key value store and draw conclusions that much better results were obtained with key value store *Cassandra*. We have built, implemented, and operated a storage system providing scalability, high performance, and wide applicability.

This segment of experimental results proved that the implementation of the OOP approach gives significantly better results compared to the results of previous work where the PP approach was applied. We have presented the several test matrix for the process of entering and searching data with experimental results and comparative analysis.

Overall, the work provides new approaches dealing with matrix computations that are widely used. Emphasis is placed on finding the best approach in the implementation of such problems in order to achieve maximum execution speed. In addition, the research is performed on issues for storing the results and the most efficient ways for storing applying popular database systems are determined and described in detail. Our research is aiming to propose better solution for calculating matrix operation that could be applied in various domains, both in mathematics and in other areas (Game theory, Economics, Data mining and Text mining).

For our future work we plan to develop mechanisms to decrease memory management, the enhancement of matrix operations, matrix type and minimal speed of the search storage systems, thereby enhancing the *DBMS* data mining functionality. Also, realization of the matrix library will be based on the principle of *Mobile Programming*. Future work also involves adding compression, ability to support atomicity across keys and secondary index support.

Appendix: Classes and procedures for algorithms of weighted Moore–Penrose inverse

Class 1. The class *UploadFile*(), the constructor and list all the methods and properties for upload txt file which contains matrix.

```
class UploadFile{
    private $files_type;
    private $max_size;
    public $userfile;
    public $filename;
    public $upload_dir;
    public $row;
    public $column;
    public function __construct(){
        $this->upload_dir = "./fajlovi";
        $this->files_type = explode(",",$ext_str);
    }
    public function upload_file($file_name="",$userfile="");
    public function row();
    public function column();
    public function filename();}
```

Class 2. The class *DataManipulation*(), the constructor and list all the methods and properties for database connect and data manipulating (search, insert, read...)

```
class DataManipulation{
    var $dbConnectionID;
    var $queryID;
    var $record;
    var $host;
    var $database;
    var $user;
    var $password;
    var $uploadfile;
    function DataManipulation($host="localhost",$db="matrices",
    $user="root",$pwd=""){
        $this->host = $host;
        $this->database = $db;
        $this->user = $user;
        $this->password = $pwd;
        $this->connect();
    }

    }
```

Class 3. Central class in our client/server application is a *class Matrix*() with variables declaration, the constructor and the list of all the methods and properties.

```
class Matrix {
    var $numbers=array();
    var $numColumns=0;
    var $numRows=0;
    function matrix(){
        $numberColumns=0;
        $numberRows=0;
        if(empty($numbers)==false){
            foreach($numbers as $i=>$rows){
                foreach($rows as $j=>$number){
                    if($number!=0){
                        $this->numbers[$i][$j]=$number;}
                    if($j>=$numberColumns){
                        $numberColumns=$j;}}
                if($i>=$numberRows){
                    $numberRows=$i;}}
            $numberRows++;
            $numberColumns++;}
        if($numberRows>$this->numRows){
            $this->numRows=$numberRows;}
        if($numberColumns>$this->numColumns){
            $this->numColumns=$numberColumns;}}
}
```

Procedure 1. *OOP* approach of the weighted Moore–Penrose inverse.

```
function WeightedInverse($MatrixM,$MatrixN){
    $aa=$this->ithCol(1);
    if($aa->isZero()==0)
        $ar=$aa->transpose();
    else{
        $ta=$aa->transpose();
        $alb=$ta->times($MatrixM);
        $ali=$alb->times($aa);
        $inv=$ali->Inverse();
        $ar=$inv->times($alb);}
    for($i=2;$i<=$this->get_num_columns();$i++){
        $ii=$this->ithCol($i);
        $di=$ar->times($ii);
        $fc=$this->frstiCol($i-1);
        $mm=$fc->times($di);
        $ci=$ii->minus($mm);
        $nim1=$MatrixN->mMinus($i);
        $nim1g=$nim1->Inverse();
        $li=$MatrixN->mColumn($i);
        if($ci->isZero()==0){
            $nii=$MatrixN->mscalar($i);
            $dit=$di->transpose();
            $dtn=$dit->times($nim1);
            $dtnd=$dtn->times($di);
            $dtnd=$dtnd->matNum();
            $ditl=$dit->times($li);
            $lit=$li->transpose();
            $litdi=$lit->times($di);
            $ditdi=$ditl->plus($litdi);
            $ditdi=$ditdi->matNum();
            $nim1li=$nim1g->times($li);
            $litnli=$lit->times($nim1li);
            $litnli=$litnli->matNum();
            $lktar=$lit->times($ar);
            $arnklk=$fc->times($nim1li);
            $novo=$lktar->times($arnklk);
            $novo=$novo->matNum();
            $si=$nii+$dtnd-$ditdi-$litnli+$novo;
            $dtnar=$dtn->times($ar);
            $dilit=$dtnar->minus($lktar);
            $bit=$dilit->s_times(1/$si);}
        else{
            $cit=$ci->transpose();
            $citm=$cit->times($MatrixM);
            $pom=$citm->times($ci);
            $pom=$pom->matNum();
            $bit=$citm->s_times(1/$pom);}
        $nim1li=$nim1g->times($li);
        $arai=$ar->times($fc);
        $arnim1=$arai->times($nim1li);
        $pi=$nim1li->minus($arnim1);
        $k1=$di->times($bit);
        $ark1=$ar->minus($k1);
        $pib=$pi->times($bit);
        $ar=$ark1->minus($pib);
        $ar=$ar->appRow($bit);}
    return $ar;}
```

Procedure 2. A new instance of the Matrix class using:

```
function computeWeighted($matrix1,$matrix2,$matrix3){
    $oop_matrix1 = new matrix($matrix1,$m,$n);
    $oop_matrix2 = new matrix($matrix2,$m,$m);
    $oop_matrix3 = new matrix($matrix3,$n,$n);
    $oop_matrix1->WeightedInverse($oop_matrix2,$oop_matrix3);
    $oop_matrix1->numbers;
    }
```

Procedure 3. Insert matrix file in *MySQL* database with *blob* data type.

```
function insertBlob($file,$db){
    $data=file_get_contents($file);
    $data=mysql_real_escape_string($data);
    $query="insert into testblob(data) values ('$data')";
    $result=$db->query($query,$db->db_link());
    $row=$db->nextRecord($result);}
```

Procedure 4. Function for insert the file, witch contains matrix, on *MySQL varchar* and *longtext* data type.

```
function insertText($matrix,$db){
    $dimension=$matrix->numRows()."x".$matrix->numColumns();
    $query="insert into matrices_in(elements_in,dimension)
    values('$matrix->$numbers','$dimension')";
    $result=$db->query($query,$db->db_link());
    $row=$this->nextRecord($result);}
```

Procedure 5. Function for the search data from MySQL database with *blob* data type.

```
function searchBlob($file,$db){
    $handle=fopen($file->filename(),"rb");
    $binarydata=addslashes(fread($handle, 65535));
    $result=$db->query(
    "select * from testblob where data='$binarydata'",$db->db_link());
    $row=$db->nextRecord($result);
    return $row[1];}
```

Procedure 6. Function for the search data from *MySQL* database with *varchar* and *longtext* data type.

```
function searchText($matrix,$db){
    $dimension = $matrix->numRows()."x".
    $matrix->numColumns();
    $query = "select * from matrices_
    in where elements_in='$matrix->$number' and
    DIMENSION='$dimension'";
    $result=$db->query($query,$this->db_link());
    $row=$db->nextRecord($result);
    return $row[1];}
```

Procedure 7. Code for create the Keyspace in Cassandra.

```
function createKeyspace(){
    $servers=array(
        array(
            'host'=>'127.0.0.1',
            'port'=>9160,
            'use-framed-transport'=>true,
            'send-timeout-ms'=>1000,
            'receive-timeout-ms'=>1000));
    $cassandra=Cassandra::createInstance($servers);
    $cassandra2=Cassandra::getInstance();
    $cassandra->createKeyspace('MatrixComputation');
    $cassandra->useKeyspace('MatrixComputation');
    $cassandra->setMaxCallRetries(5);
    $cassandra->createStandardColumnFamily(
        'MatricesComputation',
        'matrices_in',
        array(
            array(
                'name'=>'id',
                'type'=>Cassandra::TYPE_UTF8,
                'index-type'=>Cassandra::INDEX_KEYS,
                'index-name'=>'idIdx'),
            array(
                'name'=>'dimension',
                'type'=>Cassandra::TYPE_UTF8,
                'index-type'=>Cassandra::INDEX_KEYS,
                'index-name'=>'dimIdx'),
            array(
                'name'=>'test',
                'type'=>Cassandra::TYPE_UTF8),
            array(
                'name'=>'sparse',
                'type'=>Cassandra::TYPE_UTF8)));
    $cassandra->createStandardColumnFamily(
        'MatricesComputation',
        'matrices_out',
        array(
            array(
                'name'=>'matrix_I',
                'type'=>Cassandra::TYPE_INTEGER,
                'index-type'=>Cassandra::INDEX_KEYS,
                'index-name'=>'NameIdx'),
            array(
                'name'=>'matrix_II',
                'type'=>Cassandra::TYPE_INTEGER),
            array(
                'name'=>'matrix_III',
                'type'=>Cassandra::TYPE_INTEGER),
            array(
                'name'=>'operation',
                'type'=>Cassandra::TYPE_UTF8,
                'index-type'=>Cassandra::INDEX_KEYS,
                'index-name'=>'opIdx'),
            array(
                'name'=>'r',
                'type'=>Cassandra::TYPE_INTEGER),
            array(
                'name'=>'s',
                'type'=>Cassandra::TYPE_INTEGER),
            array(
                'name'=>'p',
                'type'=>Cassandra::TYPE_INTEGER),
            array(
                'name'=>'q',
                'type'=>Cassandra::TYPE_INTEGER)));
    return $cassandra;
}
```

Procedure 8. Function *getConnection*(), connecting to localhost test server.

```
function getConnection(){
    $servers=array(
        array(
            'host'=>'127.0.0.1',
            'port'=>9160,
            'use-framed-transport'=>true,
            'send-timeout-ms'=>1000,
            'receive-timeout-ms'=>1000
        )
    );
    $cassandra=Cassandra::createInstance($servers);
    $cassandra2=Cassandra::getInstance();
    $cassandra->useKeyspace('MatrixComputation');
    $cassandra->setMaxCallRetries(5);
    return $cassandra;}
```

We use this new keyspace with:

Procedure 9. Insert data in the *Cassandra* column family *matrices_in*.

```
function
insertColumnMatrixIn($cassandra,$array,$dimension,$id,$test,$sparse){
    $cassandra->cf('matrices_in')->set(
    $array,
    array(
        'id'=>$array,
        'dimension'=>$dimension,
        'test'=>$test,
        'sparse'=>$sparse),
    Cassandra::CONSISTENCY_QUORUM);
}
```

Procedure 10. Searching for data from the *Cassandra* column family *matrices_in*.

```
function
getColumnMatrixIn($cassandra,$array,$dimension,$id,$test,$sparse){
    $search=$cassandra->cf($array)->getWhere(array('id'=>
    $id,'dimension'=>$dimension,'test'=>$test,'sparse'=>$sparse));
    return $search->getAll();}
```

Procedure 11. Insert data in the *Cassandra* column family *matrices_out*.

```
function
insertColumnMatrixOut($cassandra,$matrix1,$matrix2,
$matrix3,$operation,$r,$s,$p,$q){
    $cassandra->cf('matrices_out')->set(
    $array,
    array(
        'matrix_I'=>$matrix1,
        'matrix_II'=>$matrix2,
        'matrix_III'=>$matrix3,
        'operation'=>$operation,
        'r'=>$r,
        's'=>$s,
        'p'=>$p,
        'q'=>$q),
    Cassandra::CONSISTENCY_QUORUM);
}
```

Procedure 12. Searching for data from the *Cassandra* column family *matrices_out*.

```
function
getColumnMatrixOut($cassandra,$matrix1,$matrix2,$matrix3,$operation,$r,$s,$p,$q){
    $search=$cassandra->cf('matrices_out')->getWhere(array('matrix_I'=>$
    matrix1,'matrix_II'=>matrix2,'matrix_III'=>matrix3,'operation'=>
    $operation,'p'=>$p,'q'=>$q,'r'=>$r,'s'=>$s));
    return $search->getAll();
    }
```

Procedure 13. Display result from the *Cassandra* column family *matrices_out*.

```
function
displayColumnMatrixOut($cassandra,$matrix1,$matrix2,$matrix3,$operation,$r,$s,$p,$q){
    $search=$cassandra->cf('matrices_out')->getWhere(array('matrix_I'=>$
    matrix1,'matrix_II'=>matrix2,'matrix_III'=>matrix3,'operation'=>
    $operation,'p'=>$ p,'q'=>$q,'r'=>$r,'s'=>$s));$
    result=array_keys($search->getAll());
    return $result[0];
    }
```

References

[1] Tasić M, Stanimirović P and Pepić S 2011 Computation of generalized inverses using Php/MySql environment. *Int. J. Comput. Math.* 88(11): 2429–2446

[2] Tasić M, Stanimirović P and Pepić S 2010 About the generalized *LM*-inverse and the Weighted Moore–Penrose inverse. *Appl. Math. Comput.* 216: 114–124

[3] Elmasri R and Navathe SB 2003 *Fundamentals of database systems: 4th edition.* Addison-Wesley

[4] Clear J, Dunn D, Harvey B, Heytens L, and Lohman P 1999 Non-stop SQL/MX primitives for knowledge discovery. In: *ACM KDD Conference*, pp. 425–429

[5] Sattler K and Dunemann O 2001 SQL database primitives for decision tree classifiers. In: *ACM CIKM Conference*, pp. 379–386

[6] Ordonez C and Garca J 2006 Vector and matrix operations programmed with UDFs in a relational DBMS. In: *Proceedings of the 15th ACM international conference on information and knowledge management*, Arlington, Virginia, USA, 503–512

[7] Page R and Factor P 2008 SQL Server Matrix Workbench. In: http://www.simple-talk.com/sql/t-sql-programming/sql-server-matrix-workbench/, visited: September 2015

[8] Egilsson S, Gudbjartsson H and Sigurjnsson S 2003 *SQL query generator utilizing matrix structures*. U.S. Patent 6,578,028

[9] Suehring S 2002 *MySQL bible*. Wiley Publishing, USA

[10] Obasanjo D 2009 Building scalable databases: Denormalization, the NoSQL movement and Digg. In: http://www.25hoursaday.com/weblog/2009/09/10/BuildingScalableDatabases-DenormalizationTheNoSQLMovementAndDigg.aspx, visited: sept 2015

[11] Hewit E 2011 *Cassandra: The definitive guide*. O'Reilly Media, USA

[12] Lakshman A and Malik P 2010 Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.* 44: 35–40

[13] Zielke G 1986 Report on test matrices for generalized inverses. *Computing* 36: 105–162