Using Intermediate Data of Map Reduce for Faster Execution

Shah Pratik Prakash, Pattabiraman V.
School of Computing Science and Engineering VIT University – Chennai Campus
Chennai, India

Abstract—Data of any kind structured, unstructured or semistructured is generated in large quantity around the globe in various domains. These datasets are stored on multiple nodes in a cluster. MapReduce framework has emerged as the most efficient technique and easy to use for parallel processing of distributed data. This paper proposes a new methodology for mapreduce framework workflow. The proposed methodology provides a way to process raw data in such a way that it requires less processing time to generate the required result. The methodology stores intermediate data which is generated between map and reduce phase and re-used as input to mapreduce. The paper presents methodology which focuses on improving the data reusability, scalability and efficiency of the mapreduce framework for large data analysis. MongoDB 2.4.2 is used to demonstrate the experimental work to show how we can store and reuse intermediate data as a part of mapreduce to improve the processing of large datasets.

Keywords—MapReduce, Intermediate data management, MongoDB, Architecture aware mining.

I. Introduction

COMPANIES across various industries have been archiving data in their own required format and sizes. Retrieving Statistical, Analytical and other various forms information require high amount of processing of large size raw data, which is a costly task in terms of resource and time [7]. Present cloud based product having data distributed across several nodes uses MapReduce [4] framework for the processing of data with a certain level of optimization.

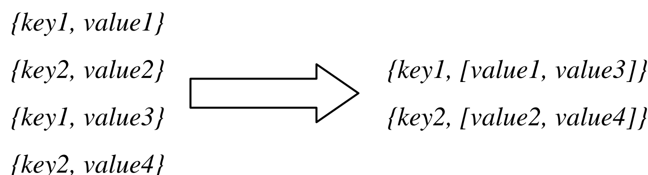
MapReduce is a parallel programming model which is very efficient for processing large data set across multiple core machine or nodes of a cluster. It is customizable so that one can implement their own logic for information retrieval. The workflow developed upon MapReduce achieves higher performance than traditional solutions. MapReduce model is based on two primary process map (Mapper) and reduce (Reducer), accompanied by combiner and finalize for performing reduction at map phase and operation on the result of reduce phase respectively. Analysis of raw data is carried out by workflow, which consists of several mapreduce jobs. The map task/phase is performed on either the raw data or on the result of previous mapreduce jobs. This phase performs calculations defined by the user and generates a key value pair as an intermediate data. The key and value represent a relation similar to a hash map in a programming construct. The map phase can emit zero or more times and same key with different values.

Map:

Raw data / Result of map reduce job ⟶ *{key, value}*

The result of mapping phases is the intermediate data which are then passed to reduce phase. The intermediate data is a list of values corresponding to unique key.

Intermediate data:



On reduce phase the user defined reduce function is performed on intermediate data. This can be ranging from sum, average, standard deviation to predictive analysis of values corresponding to the unique key. The result of reduce phase generates the same or different type of key with the result of the processed value list.

Reduce:

{key, [value1, vaue2]} ⟶ *{key3, value}*

In mapreduce framework the intermediate data generated is being deleted currently, so any mapreduce has to perform operation on raw data frequently. This paper focuses on storing and re-using the intermediate data and performing mapreduce more efficiently to generate the required output.

This was an introductory section to mapreduce framework describing its general execution and elements. Section II introduces how intermediate data are handled in MongoDB. Section III presents the proposed methodology. Section IV shows results of existing and proposed methodology. Section V is the analysis of experiment comparing existing system and proposed methodology. Section VI concludes this paper and present brief introduction to future work to be carried out. Lastly, we provide the list of references.

DOI: 10.46300/91013.2022.16.4 Volume 16, 2022

II. INTERMEDIATE DATA IN MAPREDUCE FRAMEWORK

Large dataset requires multilevel and multipath mechanism for computation. Workflow is formed composed of number of map reduce task. In a workflow the data generated by one mapreduce job is input to another and so on. It is found that several mapreduce jobs of different workflow are repeated over the raw data. This kind of execution is found in analysis, logging, transaction, image processing and other different types of complex task. As the mapreduce jobs are repeated, same intermediate data is generated which are either deleted or stored but not used. The intermediate data is a set for key value pair where key is unique and the value corresponding to the key is a list of values emitted by the map phase. The intermediate data is lost if its size is limited to temporary storage else is written to disk [2]. This intermediate data is then passed to reduce phase. Further, in this section follows MongoDB architecture and intermediate data in the mapreduce framework in MongoDB.

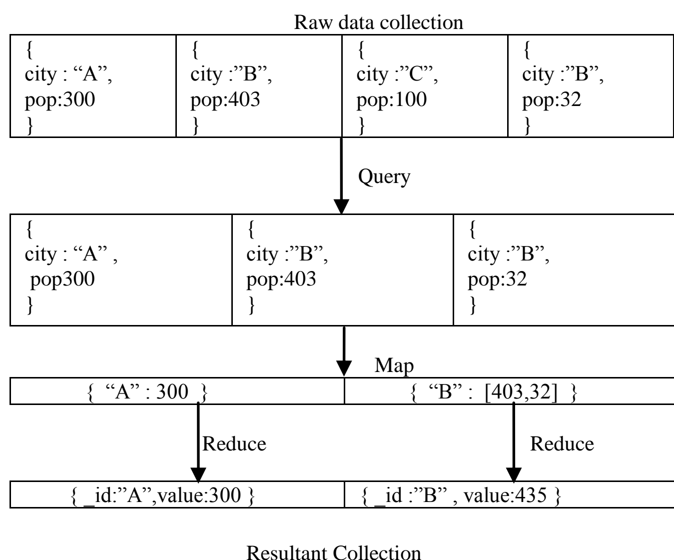


Fig 1. MapReduce operation in MongoDB.

A. MongoDB and MapReduce Framework

MongoDB is a NoSQL document-oriented database system [9]. MongoDB uses a memory map file that directly map disk data file to a memory byte array where data accesses and processing is carried out. In MongoDB "*database*" are represented in a similar pattern as RDBMS, incorporating various other objects like tables, indexes, triggers etc. Tables in MongoDB are referred as "*collection*", each can have zero or more record which is referred as "*document*". Columns in RDBMS have "*field*" as their counterpart in MongoDB.

MapReduce in MongoDB increases performance and utilization of resource [8]. Figure 1 shows a complete mapreduce operation in mongodb. It is a two-step process, map and reduce. On submission of user defined functions map and reduce to mapReduce, firstly map task is performed followed by reduce task. The map task processes either every document of the given collection or set of documents based on the specified query result. The map function emits key and its related value obtained during processing, this is at document level so there will be bunch of key-value pair, with the possibilities of pair having same named key with different values. These emitted pairs are stored in in-memory as temporal data and on memory size overflow they are stored on local disk. Figure 2 shows the existing mechanism of map phase. In reduce phase the reduce function performs a remote procedure call to mapper nodes to gather the intermediate data which is a pair of key and corresponding list of values. After the values are transferred from mapper node the user defined code processes these values for aggregation, statistical or analysis and the result is stored on a specified output collection. Figure 3 shows the existing mechanism of reduce phase.

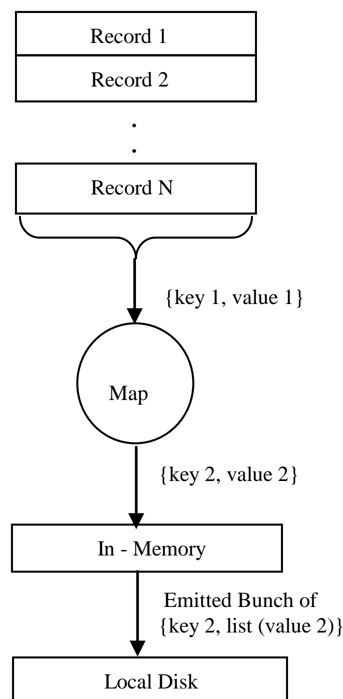


Fig 2. Existing Map phase

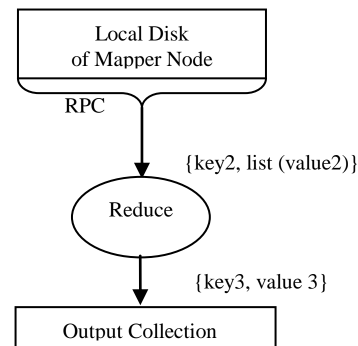


Fig 3. Existing Reduce Phase

B. Intermediate Data in MongoDB

The intermediate data are generated between the map and reduce phase. In mapreduce the reduce function is performed

only after all map tasks of job are performed. At this point the data emitted by map is sent to the reducer. During this transfer the data from different nodes are collected and values of identical key are appended to list corresponding to the key, having the form:

{key, list(value)}

This pairs are stored in-memory and on overflow are placed on disk. They are then sorted based on key and sent to corresponding reducers [3].

C. Limitations

The mapreduce framework defines a job, a complex analytical task is performed as a workflow which is composed of number of mapreduce jobs. On daily basis several workflows are initiated in a company. Some of the jobs in are common among workflows. This leads to processing of same raw data again among different workflow.

The proposed methodology focuses on following areas of improvement in mapreduce framework:

1) *Input Size:* Processing large number of records.

2) *Processing Time:* Complex processing of data for emitting key value pair in map phase increases the overall time of workflow.

3) *Intermediate Data:* Generated intermediate data left unused on disk.

III. PROPOSED METHODOLOGY

The proposed methodology addresses the limitations described in section II. Performing complex calculations on raw data requires some mapreduce jobs to be repeated at some point in workflows affecting overall execution time, the proposed methodology uses a comparison of stored required pattern [6] with the minimal set of records of the stored intermediate data collection, which on satisfying the condition alters the map phase for the actual reduce phase to reduce the processing as well as the size of inputs. Thus a decision mechanism is established at the start of mapreduce job to decide which set of data and map task to use as an input to the job [2].

Figure 4 and 5 shows the conceptual view of the proposed methodology. Using this methodology requires less processing with raw dataset, only on first computation and updating of the dataset we perform mapreduce directly on raw dataset. On the direct computation on the raw dataset we store [3] or merge the intermediate dataset in a separate collection which are likely to be used frequently in the future as an input pattern to reduce phase and also compute the reduce task.

This methodology is useful for repetitive jobs. The map task and collection on which it is to be performed jobs are decided based on processing to be carried out over the sample records of the raw data. This is like a pattern recognition process [5]. This is carried out by applying mapreduce with reduce phase just returning the emitted elements of the map phase. The output is compared with the required pattern. On success we alter the map function implantation and collection (collection having intermediate data) which provides same data as we would have found using mapreduce on raw data collection. Thus the input changes to intermediate collection and implementation of map phase just becomes re emitting the key and values from list corresponding to the key which requires much less processing and the reduce task receives the same input so output as required receives the same input so output as required.

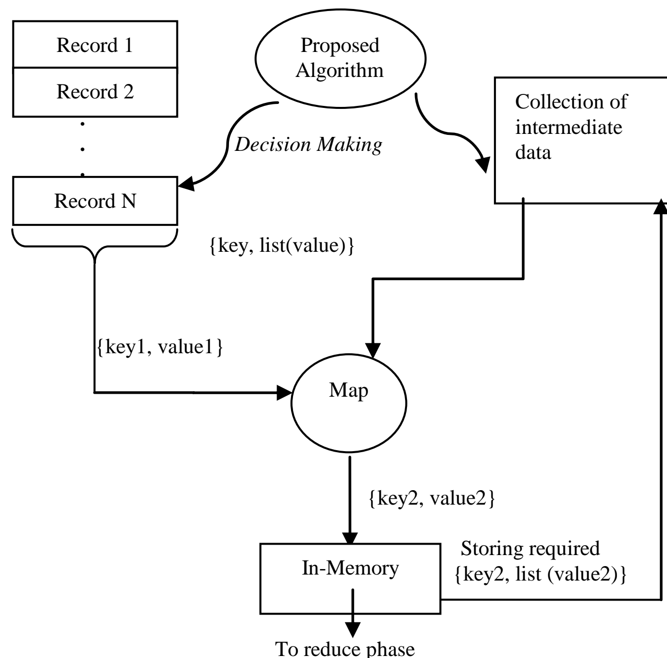


Fig 4. Proposed methodology for map phase

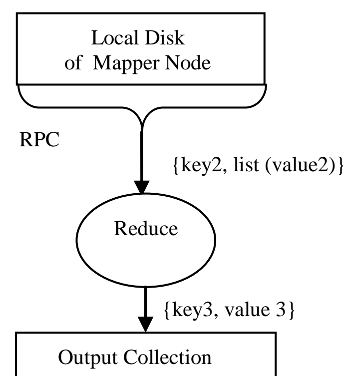


Fig 5. Reduce Phase of proposed system.

Algorithm 1 Store intermediate data

```
/* map function*/
m <= function () {
/*
  In – between calculation
*/
      emit (key, this.value);
}
/* reduce function*/
r <= function (c, p) {
   return p;
}
db.rawdatacollection.mapReduce (m, r
  {
    out: intermediate data collection
  }
);
```

Algorithm 1 shows the method how to store intermediate data using mapreduce. The method comprises of map and reduces functions where the map function can have implementation focusing on that results that are commonly generated in the form of intermediate data in mapreduce workflow. The reduce function here simply returns the list of values captured as input for a key. MapReduce is performed on collection, storing raw data and the result is stored in a separate collection for intermediate dataset.

Algorithm 2 Decision making on raw data

```
m <= user defined map function
r   <= user defined reduce function
db.rawdatacollection.mapReduce (m, r,
   {
          out: scratch collection,
          query: select sample records
   }
);
If (output: result satisfies the required pattern)
        /* alter map function and perform mapreduce on
        intermediate data collection*/
           newM <= (function processing intermediate data
           emit key value pair)
           db.intermediatecollection.mapReduce (newM, r
        {
      out: resultant collection
        }
           ) ;
else
/* perform mapreduce on raw data collection directly*/
db.rawdatacollection.mapReduce (m, r,
      {
   out: resultant collection
      }
);
```

mapReduce syntax in MongoDB:

```
db.collectionName.mapReduce (
mapFunction,
 reduceFunction,
   {
       out: {
                      <action>: <collectionName>
        [, db: <dbName>]
        [, sharded: <boolean>]
        [, nonAtomic: <boolean>]
      },
       query: // clause to restrict records,
       jsMode: <boolean>,
       ...
           }
);
```

Algorithm 2 is the proposed method used for deciding which collection and mapping task to be applied for a mapreduce job. Initially map and reduce functions are defined as per user requirement, and then tested on very small number of documents of collection having the raw data. To restrict the sample size query parameter of mapReduce function can be used. This concept is similar to scratch database or temporary table in RDBMS. The output of this job is compared with the pattern which is the way how documents of intermediate data collection would be stored. If the pattern matches the document result of scratch collection we alter the collection and map function on which the mapreduce is to be performed. The mapreduce will now be executed on intermediate data collection and the implementation of map is changed such that every value of the list corresponding to the key is emitted as a key value pair. The reduce function is not changed so the output result and the collection where the result is to be stored remains the same. This effect of this is a very small operation is required to process the data at map phase which was taking comparatively more time for complex processing. Moreover the number of inputs document for job is less while intermediate collection is considered compared to raw data collection. New intermediate documents also merge or update with documents collection periodically for consistency. This is how the overall processing is improved leading to effective workflow.

IV. RESULT OF EXPERIMENT

This section addresses the experimental setup and the experiment performed using the existing and proposed methodology.

A. Experimental Setup

The experiment was performed using MongoDB 2.4.2 on 2 GB RAM node. NetBeans IDE 7.1.2 was used for executing java application. The raw data was collection of documents providing city wise population for different states. Figure 6 shows sample documents from the collection. 20 lakh documents were initially used for evaluation. The job is to find

total state wise population. Further are the results of the experiments carried out.

{"city": "LOWER KALSKAG", "loc": [-160.359966, 61.51377], "pop": 291, "state": "AK", "_id": "99626"}
{"city": "MC GRATH", "loc": [-155.585153, 62.967153], "pop": 618, "state": "AK", "_id": "99627"}
{"city": "MANOKOTAK", "loc": [-158.989699, 59.009559], "pop": 385, "state": "AK", "_id": "99628"}
{"city": "FLAGSTAFF", "loc": [-111.574109, 35.225736], "pop": 26878, "state": "AZ", "_id": "86004"}
{"city": "EUFAULA", "loc": [-85.165605, 31.905063], "pop": 14189, "state": "AL", "_id": "36027"}
{"city": "DOZIER", "loc": [-86.366315, 31.506614], "pop": 741, "state": "AL", "_id": "36028"}
{"city": "PERRY", "loc": [-92.787976, 35.042732], "pop": 648, "state": "AR", "_id": "72125"}
{"city": "PERRYVILLE", "loc": [-92.847171, 34.970096], "pop": 3851, "state": "AR", "_id": "72126"}
{"city": "PLUMERVILLE", "loc": [-92.620435, 35.157466], "pop": 1940, "state": "AR", "_id": "72127"}

Fig 6. Raw data documents (records) from the collection

Firstly, the mapreduce was executed on the raw data collection directly. The execution input size and time size calculated is presented in Figure 7. Same required job was performed using the proposed methodology where first the intermediate data is stored in separate collection and then before performing the required job check is made that if the pattern of intermediate data exists if true, we modify the map implementation with reduce implementation same as that used in existing systems and calculate the input size and time taken. Figure 8 shows the proposed methodology result.

The time taken for retrieving sample results from the intermediate data collection and comparing is 250ms. Thus, combining processing time of mapreduce using intermediate data and time for pattern check is 10969ms which is less than processing time of existing methodology. Input size is half of the size used in existing methodology.


```
{
    "result" : "result",
    "timeMillis" : 39056,
    "counts" : {
        "input" : 2000000,
        "emit" : 2000000,
        "reduce" : 100000,
        "output" : 100000
    },
    "ok" : 1,
}
```


Fig 7. Input size and processing time in existing system.


```
{
    "result" : "Intresult",
    "timeMillis" : 10719,
    "counts" : {
        "input" : 100000,
        "emit" : 2000000,
        "reduce" : 100000,
        "output" : 100000
    },
    "ok" : 1,
}
```


Fig 8. Input size and processing time in proposed methodology.

V. ANALYSIS OF EXISTING SYSTEM AND PROPOSED METHODOLOGY.

The evaluation of proposed methodology was performed taking the aggregation experiment. This experiment aim is to compare the performance of existing system and proposed model, calculating output with new intermediate dataset and reducing the size of input documents for generating the same output as using the raw data documents.

A. Intermediate Data and Result in Existing System

Firstly mapreduce was carried out on the documents of raw data collection directly and output was stored on resultant collection. Repeatedly performing the task for same map and different reduce implementation such as count resulted in same intermediate data but was used for internally between two phases. Thus the intermediate remained unused and processing was carried out on periodically increasing raw data affecting the processing time.

B. Decision making based on pattern matching

The decision making phase gets a single document from raw data and performs map only operation and stores the result in scratch database. The resultant output’s keys are compared with the key of intermediate collection.

Scratch collection: [_id, value]

Intermediate collection: [_id, value]

On matching the mapreduce operation of the workflow is performed on the intermediate collection where original map function is altered to logic given below:

```
for(var index in this.value) {
        emit (this.key ,this.value[index]);
    }
```

The reduce phase is kept same as the original one. This overall reduces the processing incurred that would be large when executed on raw data collection.

C. Intermediate Data and Result in Proposed Methodology

Experiment with proposed system at only first time the map function of map reduce is executed on raw data collection, the successive mapreduce jobs were first sent to decision making phase where Algorithm 1 is applied to the minimal number of document using query parameter and storing the resultant output in scratch collection. This result is compared with the required pattern, on success map function was altered and output was stored in intermediate collection. The collection was also altered on which the mapreduce to be performed. If the pattern did not match the normal execution was performed. In effect of this the input size decreased for same calculation, intermediate data got reused and processing time was also reduced compared to existing mapreduce workflow. Figure 9 shows the comparison of input size and processing time of existing and proposed methodology.

The result of analysis:

P=Number of documents in raw data collection.

I=Number of documents of intermediate data.

Size=Input size for map phase

1. Time(I) < Time(P)
2. Size(I) < Size(P)

TABLE I

INPUT SIZE FOR EXPERIMENT AMONG SYSTEMS.

Sr no.	Existing System		Proposed Methodology	
	Number of documents	Processing time (ms)	Number of documents	Processing time (ms)
1	500000	9960	25000	3174
2	1000000	20305	50000	5639
3	1500000	30314	75000	8511
4	2000000	39056	100000	10969

Table I. shows the number of documents and corresponding processing time in mapreduce for existing and proposed methodology. We can see that there is a drop in the number of documents to be processed which would decrease the execution time. As the number of documents to be processed decreases there is a relative decrease in processing time this is shown in Figure 9 which shows the execution time between the systems.

VI. CONCLUSION

The proposed methodology and the experiment performed results that for large datasets the intermediate data storage and re-usage of can be highly efficient for mapreduce framework. The limitations described in section II(c) are overcome using proposed mechanism. It was shown how changes in the map side affect the processing. As per general implementation large calculations are carried out on map phase, so diverting the implementation on collection having intermediate data bring down the complex processing to minimal processing leading to reuse of intermediate data and managing of input size with a reduction in overall processing time. The challenge with the proposed model is that the intermediate data produced by the mapper has some extra documents internally generated which exceed the size of the collection of intermediate dataset. To overcome this jsMode was enabled but it's limited to 5, 00,000 key of result set. This is due the current version of mongodb 2.4 that does not support for multiple valued in mapreduce.

Future work for this is to perform same implementation combining Hadoop and MongoDB. Text search on Hadoop is the next challenge as there is a lot of potential areas to be worked upon to optimize and get accuracy in search in big data.

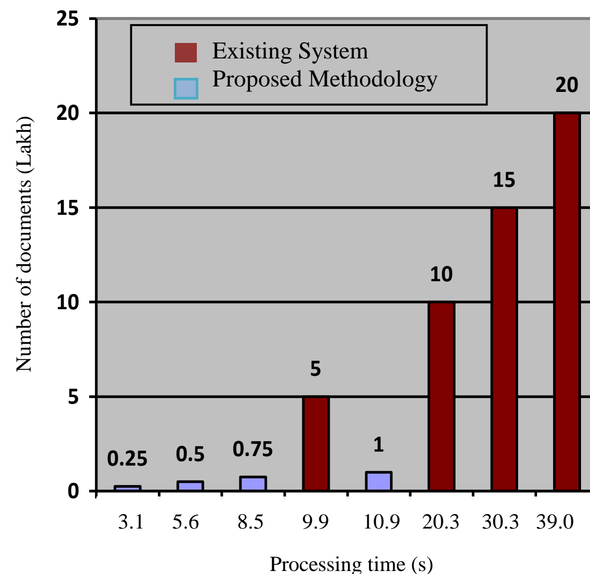


Fig 9. Number of documents (lakh) being processed and processing time (ms) in existing system and proposed methodology.

REFERENCES

[1] A. Espinosa, P. Hernandez, J.C. Moure, J. Protasio and A. Ripoll . “Analysis and improvement of map-reduce data distribution in mapping applications”. Published online: 8 June 2012 JSupercomput (2012) 62:1305-1317.

[2] Diana Moise, Thi-Thu-Lan Trieu, Gabriel Antoniu and Luc Bougé. “Optimizing Intermediate Data Management in MapReduce Computations”. CloudCP ‘11 April 10, 2011 Salzburg, Austria ACM 978-1-4503-0727-7/11/04.

[3] Iman Elghandour and Ashraf Aboulnaga. “ReStore: Reusing Results of MapReduce jobs”. The 38th International Conference on Very Large Data Bases, August 27th 31st 2012, Istanbul, Turkey. Proceedings of the VLDB Endowment, Vol. 5, No. 6.

[4] J. Dean and S. Ghemawat. “MapReduce: Simplified data processing on large clusters”. In Proc. OSDI, pages 137-150. 2004.

[5] Qiang Liu, Tim Todman, Wayne Luk and George A. Constantinides. “Automatic Optimisation of MapReduce Designs by Geometric Programming”. FPT 2009 IEEE.

[6] Rabi Prasad Padhy. “Big Data Processing with Hadoop-MapReduce in Cloud Systems”. International Journal of Cloud Computing and Services Science (IJ-CLOSER) Vol.2, No.1, February 2013, pp. 16~27 ISSN: 2089-3337.

[7] Raman Grover and Michael J. Carey. “Extending Map-Reduce for Efficient Predicate-Based Sampling”. 2012 IEEE 28th International Conference on Data Engineering.

[8] Ruxandra Burtica, Eleonora Maria Mocanu, Mugurel Ionut, Andreica, and , Nicolae Ţăpuş. “Practical application and evaluation of no-SQL databases in Cloud Computing”. 2012 IEEE. This paper is under research grants PD_240/2010 (AATOMMS - contract no. 33/28.07.2010) from the PN II - RU program and ID_1679/2008 (contract no. 736/2009) from the PN II - IDEI program.

Creative Commons Attribution License 4.0 (Attribution 4.0 International, CC BY 4.0)

This article is published under the terms of the Creative Commons Attribution License 4.0
https://creativecommons.org/licenses/by/4.0/deed.en_US