JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 505-518 (2018)
DOI: 10.6688/JISE.201803_34(2).0012

# Arvand: A Method to Integrate Multidimensional Data Sources Into Big Data Analytic Structures

MOHAMMADHOSSEIN BARKHORDARI AND MAHDI NIAMANESH
*Advance Information System Research Group*
*Information and Communication Technology Research Centre*
*Tehran, 1599616313 Iran*
*E-mail: {Barkhordari; Niamanesh}@ictrc.ac.ir*

OLAP (online analytic processing) systems provide valuable insights into organizations; thus, it becomes necessary to integrate legacy OLAP systems into scalable and distributable architectures. This project comprises two important tasks: the first is transferring OLAP cubes to share nothing architectures. The second task is integrating OLAP information with other OLAP systems over distributable and scalable architectures. The main problem is to convert conceptual model OLAP data sources to shared nothing architectures. An additional problem is query execution time on the shared nothing architectures because by default, complete data locality is not considered in these environments.

In this paper, Arvand is proposed. This method can transfer multidimensional data sources into shared nothing architectures. Data are captured from multidimensional data sources and converted into a unified format. Through unification, multidimensional data sources can be easily distributed over homogeneous and heterogeneous nodes because the nodes will not need additional information from other nodes. As an added benefit, MapReduce methods can be used properly and with maximum performance for query retrieval. Arvand is implemented by adding some components to Hadoop. In this paper, architectures with different heterogeneous and homogenous nodes are proposed and evaluated using a TPC-DS benchmark.

***Keywords:*** multidimensional data source, MapReduce, data warehouse, big data, analytics

## 1. INTRODUCTION

The pace of information generation has accelerated, and a large volume of information is produced by a wide range of channels. Scientific instruments, sensors, social networks, and smartphones are some examples of big data. Information is generated rapidly and requires storage; therefore, the old methods for storing data must evolve. Just storing information is not the ultimate solution: prompt access to data and data analysis are important concerns. To solve data storing bottlenecks, distributed architecture is used, and this changes the software development model. One of the most prevalent methods is MapReduce, which helps divide big problems into smaller ones.

Another important issue is legacy systems; these contain information that is important and vital, but they are generally not compatible with new distributed structures. One type of legacy system is OLAP (online analytic processing) systems. These systems contain valuable information about different aspects of an organization but are not scalable and cannot be merged with other OLAP systems due to structural problems. There-

Received December 13, 2016; revised March 9, 2017; accepted March 31, 2017.
Communicated by Jan-Jan Wu.

fore, it is necessary to use a method to extract multidimensional system data that can be used in an analytic big data system. As mentioned in [1], one of the most important future trends for big data is merging it with existing multidimensional data.

The main problem with transferring multidimensional data sources to shared nothing structures is determining how to convert conceptual models of a multidimensional data source to NOSQL or a structure that is comprehensible for those systems run on shared nothing structures such as Hadoop. Another problem is that there are different types of multidimensional data sources depending on the value combinations that they store. There are various types of OLAP systems. For example, there is multidimensional OLAP or MOLAP. In this structure, all combinations of measures and dimensions are computed and stored. In contrast, there are some OLAP models where calculation is performed at run time rather than all combinations of dimensions and measures being calculated and stored. These groups of OLAP are called relational OLAP or ROLAP. There are various other OLAP model types such as hybrid OLAP or HOLAP. MOLAP executes queries faster but needs more space to store cubes, and the cube making process is more time consuming than it is for ROLAP. When investigating OLAP data source transformation, the OLAP model is one of the most important issues.

In this paper, Arvand is proposed. This method converts existing multidimensional data sources in legacy systems to uniform structures that can be deployed over scalable and distributable structures. In addition to being converted to scalable architecture, existing multidimensional data can be merged with big data and other OLAP and OLTP systems. One of the main advantages of the proposed method is that it offers structure unification for all types of OLAPs. This unified structure helps the architecture to easily distribute data on several nodes with various architectures. There are two types of transformation. The first is the transformation of a single multidimensional system to scalable structures. The second is the transformation of multiple multidimensional data systems with different structures and platforms to scalable structures.

In this paper, works relating to the proposed method are investigated in section 2; then in section 3, the proposed method and its various architectures are presented. In section 4, deployment is discussed, and in section 5, the method is evaluated. The final part, section 6, offers the conclusion.

## 2. RELATED STUDIES

This section investigates works relating to the proposed method. This section consists of two parts. First, related works that address the transfer of multidimensional data sources to shared nothing architectures are reviewed; second, methods to solve problems in shared nothing architectures that are based on MapReduce, such as Hadoop, are explained. All studies to date cover only the transfer of a multidimensional data source to shared nothing architectures and neglect data retrieval time in the destination architectures. Hence, share nothing architectures have problems, such as not meeting data locality in each node. Not meeting data locality occurs when a node does not have complete data to finish its process. This problem causes data dependency on the other nodes to create results. To be accessible from other nodes, data must be transferred over the network. Using the network to transfer data causes network congestion. One of the most

important methods used over shared nothing structures is MapReduce. Hadoop is based on Map-Reduce, and many open source products such as Hive, HBase and Spark-SQL work on Hadoop or directly use MapReduce. Therefore, problems with MapReduce are transferred with these products when they are used.

### 2.1 Transferring Data from a Multidimensional Data Source to NOSQL

There are two types of methods investigated in this section. The first type is methods that transfer logical models to NOSQL architectures, and the second type is methods that transfer conceptual models to NOSQL architectures.

The first group transfers relational data models to NOSQL. Some of these methods convert relational data models to HBase [3, 4]. Other methods convert relational data models to MongoDB [2, 5]. These methods try to convert a multidimensional data source to relational data models and support only star-schema models. A star-schema model is a model with one fact and many dimensions. If there is a hierarchy in dimensions, the model is called a snowflake, and if there is more than one fact, the model is called a constellation. The second group includes methods that transfer conceptual data models to NOSQL. In these methods, multidimensional data sources are transferred to NOSQL with some rules. In [6], multidimensional data sources are transferred to a document based on NOSQL, and in [7], multidimensional data sources are transferred to a column-based NOSQL. In these methods, the multidimensional data source models are star-schema models. In [8], three types of transformation are covered. In the first method, dimensions and measures are directly transferred to NOSQL (one table for each fact and dimension). In the second method, one table is transferred. Facts and dimension information are merged in that table. The last method is similar to the second method but with one difference: it uses a column family instead of a simple attribute. In [9], a rule-based method is used to convert a multidimensional data source to Hive. This method creates a Hive table based on each combination of dimensions and measures in addition to the star-schema model that supports a constellation model. In [19], three physical data warehouse designs were investigated to analyse the impact of attribute distribution among column-families in HBase based on OLAP query performance. The authors conclude that OLAP query performance in HBase can be improved by using a distinct set of attribute distributions among column-families. In [20], a method is proposed that transfers legacy data warehouses to Hive. In [21], data from legacy data warehouses are transferred to Hive by a rule-based method. NoAM [22] is an abstract model for NoSQl databases that extracts commonalities of various NoSQL systems.

### 2.2 MapReduce Optimizations

In this section, MapReduce problems related to multidimensional data sources are investigated. As mentioned above, Hadoop is based on the MapReduce method. Further, many open source products are either based on Hadoop or implement MapReduce directly. According to [10], MapReduce has nine main problems. Some of the problems that are related to our discussion are a high communication cost and a lack of support for join operations. Methods have been proposed to improve these problems. In some of these methods, data colocation is used [11], while in others [14-16], a data layout is used

to address performance issues. In these latter methods, the data file format is changed and optimized. Some important methods are Llama [12] and Cheetah [13]. There are also methods that try to add join to MapReduce by adding phases to MapReduce such as map-reduce-merge [17] or map-join-reduce [18].

In this paper, a method for transferring a multidimensional data source to shared nothing architectures is proposed. This method can transfer all models of a data warehouse to a distributive environment. In addition, this method can solve data locality problems in shared nothing architecture; therefore, the nodes do not need any data from other nodes. This method can be used with both homogeneous and heterogeneous nodes. According to the proposed data format, several allocation methods are possible. Additionally, the proposed method can combine data from different multidimensional data sources on various platforms.

## 3. PROPOSED METHOD

Each multidimensional data model has several dimensions and measures, and reports in OLAP are extracted by combining them. The proposed method requires that all dimensions and measures of the OLAP cube data source be put together in a report. For example, suppose that there is a cube in OLAP that has four dimensions and three measures; for Arvand input, a seven-field data source must be built.

The first table in the proposed method is Arvand_Cube, which stores cube information from different OLAP data sources. The next step is to build an Arvand_Dimension table. The columns of this table are equal to the number of cube dimensions plus two. If there are $k$ dimensions in the input data source, Arvand_Dimension will look like Table 2. Dimension_ID is the table key, Cube_ID is the Arvand_Cube key, and the other columns are the input dimension names. In each dimension name field, all used values of a dimension are stored. If there is more than one OLAP cube, then the number of Arvand_Dimension columns is equal to the maximum number of dimensions for the different cubes. To specify what dimension of data each column of Arvand_Dimension contains, Arvand_Dim_Map is used. In this table, the mapping between the Arvand_ Dimension columns and the cube dimensions is determined. Measure information is required, so Arvand_Measure is defined as in Table 4. To store the measure values, Arvand_Fact is defined as in Table 5. Arvand_Fact_ID is the table key, Measure_ID is the key of the Arvand_Measure, Dimension_ID is the key of Arvand_Dimension, and Measure_Value is the value of the measure in the input data source. The last Table is Arvand_Relation. This table is used to maintain the equivalent dimensions and measures of different cubes. A field (Rel_Type) indicates whether a relation is defined for dimensions or measures.

**Table. Arvand_Measure table.**

| Arvand_Cube |
|---|
| Cube_ID |
| Cube _Name |

**Table 2. Arvand_Dimension table.**

| Arvand_Dimension |
|---|
| Cube_ID |
| Dimension_ID |
| $Dimension_1$_Name |
| $Dimension_2$_Name |
| … |
| $Dimension_k$_Name |

**Table 3. Arvand_Dim_Map table.**

| Arvand_Dim_Map |
|---|
| Cube_ID |
| Dimension_ID |
| Dimension_Name |
| Column_Order |

**Table 4. Arvand_Measure table.**

| Arvand_Measure |
|---|
| Cube_ID |
| Measure_ID |
| Measure_ Name |

**Table 5. Arvand_Fact table.**

| Arvand_Fact |
|---|
| OLAP_Fact_ID |
| Dimension_ID |
| Measure_ID |
| Measure_Value |

**Table 6. Arvand_Relation table.**

| Arvand_Relation |
|---|
| DimMeas_ID1 |
| DimMeas_ID2 |
| Rel_Type |

For example, there is an OLAP cube with the following specifications:

**Table 7. Measures and dimensions definition.**

| Dimensions | Date(Year, Month, Full Date), Supplier, Place(Province, City) |
|---|---|
| Measures | Cost, Sell |

If the input table from the OLAP cube is Table 8.

**Table 8. Sample OLAP data source.**

| | Date | | | Place | | Supplier | Measures | |
|---|---|---|---|---|---|---|---|---|
| OLAP_Fact_ID | Full Date | Month | Year | Province | City | Supplier | Sell | Cost |
| 1 | 20120616 | 6 | 2012 | P1 | C1 | S1 | 2000000 | 16000 |
| 2 | 20120711 | 7 | 2012 | P1 | C2 | S2 | 2500000 | 18000 |
| 3 | 20130616 | 6 | 2013 | P1 | C1 | S1 | 1800000 | 20000 |

**Table 9. Arvand_Cube table sample data.**

| Cube_ID | Cube_Name |
|---|---|
| 1 | SampleCube |

Then, the Arvand_Cube tables can be filled as in Table 9.

The date dimension values equal “2013/06/16, 2012/07/11, 2012/06/16”, the supplier dimension values equal “S1, S2”, the province values equal “P1”, and it has two children “C1, C2”. To fill Arvand_Dimension, the table dimensions must be combined as in Fig. 1. In other words, all dimension values at any level of the hierarchy must be considered. The total states are the multiple of each dimension value, being one hundred and eight (9 * 3 * 4) in this example.

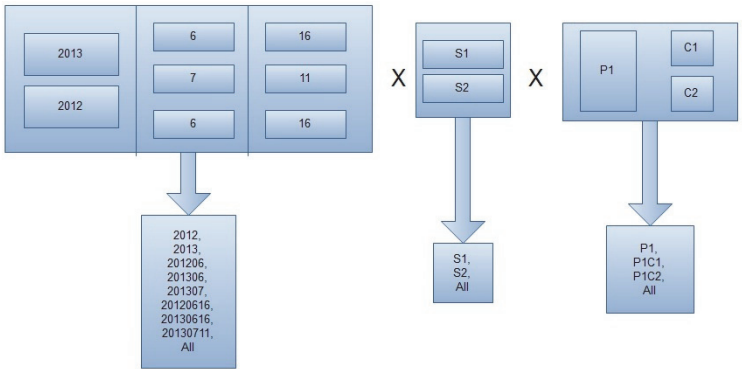


Fig. 1. Different combinations of dimensions.

According to the above descriptions, the Arvand_Dimension table is filled by the following values. Each level of the dimension hierarchy is separated by

**Table 10. Arvand_Dimension table sample data.**

| Cube_ID | Dimension_ID | Date | Place | Supplier |
|---|---|---|---|---|
| 1 | 1 | 2012 | S1 | P1 |
| 1 | 2 | 2012 | S1 | P1@C1 |
| … | … | … | … | … |
| 1 | 108 | All | All | All |

The measure definitions in the Arvand_Measure table are shown in Table 11.

**Table 11. Arvand_Dimension table sample data.**

| ID_Measure | Name_Measure |
|---|---|
| 1 | Cost |
| 2 | Count |

Because there is only a cube, there is no record in the Arvand_Relation table. Data items are entered in Arvand_Fact as Table 12. Because there are two measures, the dimension combinations count is multiplied by two (two hundred and sixteen).

**Table 12. Arvand_Fact table sample data.**

| Arvand_Fact_ID | Dimension_ID | Measure_ID | Measure_Value |
|---|---|---|---|
| 1 | 1 | 1 | 2000000 |
| 2 | 2 | 1 | 2500000 |
| … | … | … | … |
| 216 | 108 | 2 | 54000 |

If there is more than one multidimensional data source after they are all converted to the proposed format, then the dimensional relationship for each multi-dimension data source with other multi-dimension data sources should be determined if they have a relationship. These relations should also be determined for measures. The proposed method has a unified structure that has the following advantages over distributed nodes:

- Because all needed information is in the node, each node can execute its query independently.
- Data distribution over nodes with different architectures is possible. Some distribution architectures are
  - ✓ Dimension-based
  - ✓ Row-based
  - ✓ Query-based
  - ✓ Dimension and row-based
  - ✓ Business-based
  - ✓ Uniform-based
  - ✓ Hit ratio-based
  - ✓ Statistical and data mining algorithm-based

  - ✓ Measure-based
- Depending on the data structure of the proposed method, using MapReduce methods is possible, and parallel and scalable architectures can be used to address performance issues.
- The proposed method has no dependency on a particular hardware; therefore, homogeneous and heterogeneous nodes can be used in deployment architecture design.
- During query execution, due to the data structure of the proposed method, there is no need to complete the information for the dimensions.
- Due to the unification of the data structure, the calculation speed increases, and the variety of calculations decreases.

## 4. DEPLOYMENT ARCHITECTURES

In this section, various deployment architectures for the proposed method are investigated. The first considered architecture is dimension-based distribution, in which each node contains information of a particular dimension. Due to complete information in Arvand_Dimension, the fragmentation of data is easily possible.

Nodes can be homogeneous or heterogeneous. If nodes are homogeneous, the map phase speed is equal for all nodes; if they are heterogeneous, the mapping phase speed equals the processing time of the weakest node. For the proposed method, the query in this method is first changed based on the dimensions. Next, each query is sent to a related node. In this architecture, a group of nodes can be used for a dimension instead of a node. The number of nodes for a dimension varies based on the processing power, memory and dimension hit ratio. In heterogeneous nodes, the proposed method helps to create approximately homogeneous groups.

The second architecture is row-based distribution. In this architecture, a query is sent to all nodes, their results are sent to a Reducer, and the final result is then calculated. Nodes can be homogeneous or heterogeneous. If heterogeneous nodes are selected, the performance of the architecture is equal to that of the weakest node.

The third considered architecture is query-based distribution. In this method, data fragmentation and allocation is based on queries. First, it is determined which dimensions and measures are needed for each query. Then, data are allocated to each node based on the measures and dimensions. At run time, each query is sent to its related node using metadata. The proposed architecture is parallel, and multiple queries can be run on it simultaneously. In addition, it is very easy to execute a query on more than one node when needed. If there are sets of data needed by different nodes, the set can be saved on a node or replicated on each node. As a subset of query-based distribution, a row and dimension-based distribution can also be used. In this architecture, a query is not a criterion for data fragmentation and allocation. A combination of dimensions and rows are used for fragmentation and allocation, and a node or a group of nodes can be used for the combination. A business-based distribution is another subset of query-based distributions. In these architectures, data are fragmented according to business needs, and each node (or group of nodes) contains information of a part of a business. If there is common data between business parts, it can be located on a node (a group of nodes) or replicated over all related nodes.

Hit ratio-based distribution is the fourth considered architecture. In this architecture, data are categorized based on the hit ratio. High hit ratio data are allocated to powerful nodes, and low hit ratio data are allocated to medium and weak nodes. This architecture can properly use existing hardware with high performance and can execute multiple queries in parallel.

Statistical methods and data mining algorithm distribution are the fifth considered architecture. In this architecture, data mining algorithms and statistical methods are used for data fragmentation. Each fragmentation, or a set, can be allocated to a node or group of nodes.

Measure-based distribution architecture is the last proposed architecture. Because of the uniform structure of the proposed method, it is possible to fragment data based on measures. Measures can be allocated to a node or a group of nodes based on business concerns such as security and performance. In Arvand, Arvand_Allocator allocates data to nodes based on one of the above architectures. Arvand_Allocator is activated when ETL runs on different multidimensional data sources. In other words, when data are converted from the data source format to the Arvand format, the Arvand_Allocator allocates data rows to each node based on the selected allocation method.

## 5. DEPLOYMENT ARCHITECTURES

In this section, Arvand in compared with Hive [31] and Spark-SQL [30], these being two major data warehouse products for big data. Data from OLAP data sources are transferred to Hive and Spark-SQL with no change. Hadoop 2.7.3 [33] is used to implement Arvand, but some changes are made on the NameNode. In addition, Arvand allocates data to each DataNode using Arvand_Allocator in the ETL phase. As another difference, the proposed method uses PostgreSQL 9.6.1 [29] as a database on each DataNode, and Redis 3.2.5 [28] is used as the database on the NameNode with disk persistent =AOF. Redis is an in-Memory database, and Arvand uses it on the NameNode for performance issues. Fig. 2 shows Arvand implementation in Hadoop.

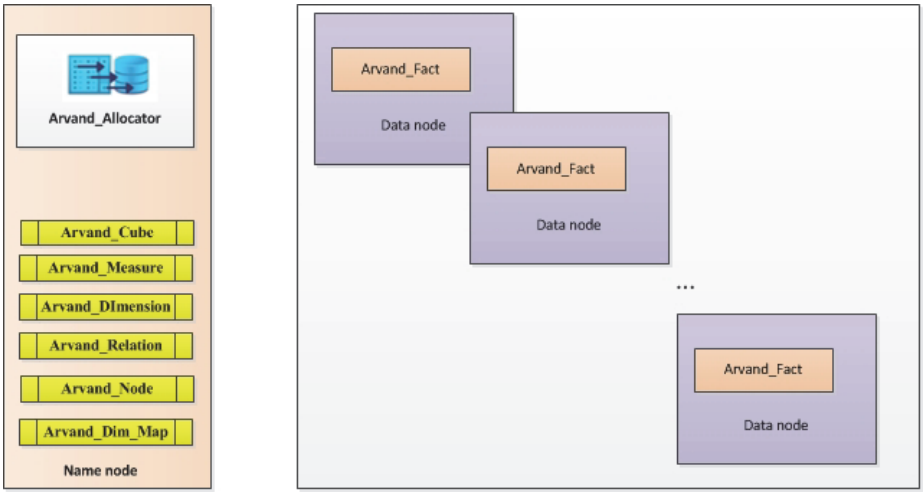


Fig. 2. Arvand implementation on Hadoop.

For the evaluation, TPC-DS 2.3.0 [34] queries are used. For data generation, TPC-DS_Tools_v2.3.0 [35] is used. Five fact tables and their related dimensions are used (store_sales, store_returns, catalog_sales, catalog_returns, web_sales, web_returns, inventory). All fact tables are allocated to servers with specifications as in Table 13. The scale factor is equal to 10 TB, and SF=10000. The first thirty queries of TPC-DS are selected for evaluation. To evaluate the proposed method, two groups of experiments are performed. First, each of the data sources is distributed over nodes separately, and different architectures are investigated in two modes, homogeneous and heterogeneous. Then, all data sources are combined and investigated in homogeneous and heterogeneous modes.

The first twenty nodes with specifications as shown in Table 13 are selected.

**Table 13. Analytic servers' specification.**

| Node | Count: 20 |
|---|---|
| CPU | Intel Core i5-2500K Quad-Core Processor 3.3 GHz |
| HDD | 5TB |
| RAM | 16 GB |

Each node has Ubuntu 16.1 [32] as the operating system and PostgreSQL 9.6.1 as the database. Each node can perform its process separately due to the Arvand data format. Then, Hive and Spark-SQL on Hadoop is used over twenty nodes. The following configuration is used for Hive and Spark-SQL.

**Table 14. Hadoop configuration.**

| | |
|---|---|
| dfs.replication | 3 |
| mapred.map.tasks | 19 |
| mapred.reduce.tasks | 1 |

**Table 15. Spark configuration.**

| | |
|---|---|
| SPARK_WORKER_CORES | 19 |

Table 17 shows the Arvand_Allocator's time to transform the multidimensional data source to Arvand format. To import data into Hive and Spark-SQL, the "load data" method is used, and each fact and dimension is a csv (comma-separated value) file. Arvand_Allocator uses different strategies to allocate data to nodes and takes longer than Hive or Spark-SQL. The best Arvand_Allocator load time is for row-based distribution, which requires the least computation and fragmentation. The maximum time is for the query-based method because twenty TPC-DS queries must be analysed and data fragmentation must be performed based on the results. In the heterogeneous mode, Hive and Spark-SQL have the same times, but Arvand_Allocator spends more time to achieve load balancing. Table 21 shows the load time for homogeneous and heterogeneous nodes. Fig. 4 shows results of the average execution time of queries on homogeneous nodes. In other methods for evaluating architecture types, nodes are considered heterogeneous. In this method, three types of nodes are considered; these are in Tables 16-18.

**Table 16. Node type 1.**

| Node type1 | Count : 5 |
|---|---|
| CPU | Intel Core i3-530 Processor 2.93 GHz |
| HDD | 1TB |
| RAM | 8 GB |

**Table 17. Node type 2.**

| Node type2 | Count : 10 |
|---|---|
| CPU | Intel Core i5-2500K Quad-Core Processor 3.3 GHz |
| HDD | 5TB |
| RAM | 16 GB |

**Table 18. Node type 3.**

| Node type3 | Count : 5 |
|---|---|
| CPU | Intel Core i7-3770 Quad-Core Processor 3.4 GHz |
| HDD | 10 TB |
| RAM | 32 GB |

The following configuration is used for Hive and Spark-SQL.

**Table 19. Hadoop configuration.**

| | |
|---|---|
| dfs.replication | 3 |
| mapred.map.tasks | 17 |
| mapred.reduce.tasks | 3 |

**Table 20. Spark configuration.**

| | |
|---|---|
| SPARK_WORKER_CORES | 19 |

**Table 21. Load data time (seconds).**

| Load time(seconds) | Hive | Spark-SQL | Dimension based | Row base distribution | Measure base distribution | Query base distribution |
|---|---|---|---|---|---|---|
| Homogeneous nodes | 900 | 900 | 2580 | 1398 | 2250 | 6300 |
| Heterogeneous nodes | 900 | 900 | 5520 | 2502 | 3363 | 9005 |

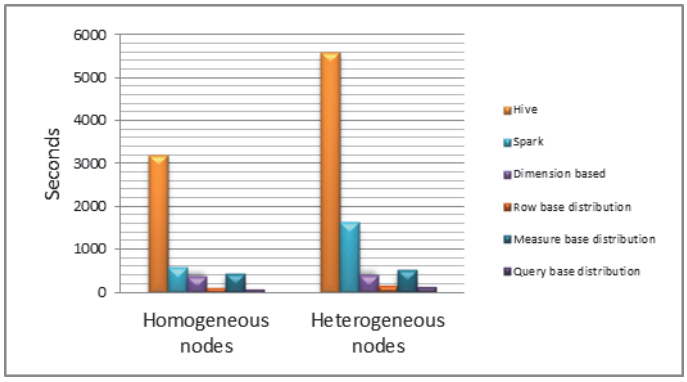


Fig. 3. Homogeneous/heterogeneous architecture types query execution times.

The results are shown in Fig. 3.

In this part of the evaluation, five multidimensional data sources with different scale factors and SF are combined together, and their performances with homogeneous nodes and heterogeneous nodes are compared. In homogeneous mode, 100 nodes with specifications shown in Table 13 are used. In the heterogeneous mode, 30 nodes with specifications from Table 16 and SF=10000 and Scale factor=10TB, 40 nodes with specifications from Table 17 and SF=100000 and Scale factor=100TB and 30 nodes with specifications from Table 18 and SF=100000 and Scale factor=100TB are used. For Hive and Spark, the following configuration is used.

**Table 22. Hadoop configuration.**

| | |
|---|---|
| dfs.replication | 3 |
| mapred.map.tasks | 90 |
| mapred.reduce.tasks | 10 |

**Table 23. Spark configuration.**

| | |
|---|---|
| SPARK_WORKER_CORES | 99 |

Load time is as shown in Table 24.

**Table 24. Load data time (Seconds) for multiple data sources.**

| Load time (seconds) | Hive | Spark-SQL | Dimension based | Row base distribution | Measure base distribution | Query base distribution |
|---|---|---|---|---|---|---|
| Homogeneous nodes | 2520 | 2520 | 8502 | 3608 | 9748 | 19885 |
| Heterogeneous nodes | 2520 | 2520 | 16205 | 9514 | 11258 | 29663 |

To help Hive and Spark know the relations between cubes, Arvand_Relation data are used for query processing. The results are shown in Table 25.

**Table 25. Query processing time for OLAP cubes combination.**

| Architecture type/Product | Average query processing time(s) in Homogenous node | nAverage query processing time(s) in Heterogeneous nodes |
|---|---|---|
| Hive | 4800 | 12580 |
| Spark | 2055 | 5205 |
| Dimension-based distribution | 415 | 450 |
| Row-based distribution | 136 | 176 |
| Measure-based distribution | 580 | 598 |
| Query-based distribution | 105 | 142 |

## 6. CONCLUSIONS

Organizations hold many multidimensional data sources with valuable information. To use the information, distributable architectures are required to lower the query execution time. However, data transfer to scalable and distributable environments does not completely solve the multidimensional execution time problem because each node needs data from other nodes to execute its query. If each node has all of the related data, network congestion decreases dramatically.

In this paper, Arvand is proposed as a method for data unification between different multidimensional data sources. Using this method, legacy multidimensional data sources can benefit from data distribution and scalability. Because the data format is unified in Arvand, each node can perform its process separately and does not need data items from other nodes. To transfer data from legacy multidimensional data sources to a scalable and distributable environment, data sources must send their data items with an Arvand defined format. Arvand_Allocator is used to allocate multidimensional data sources to different nodes. In some scenarios such as query-based or dimension-based allocation, it

takes longer to allocate data to nodes, but at query execution time, queries are executed more rapidly than they are with other methods. To implement the proposed method, Hadoop is used, and some changes in the NameNode and DataNodes are made. The proposed method was evaluated using several architectures with heterogeneous and homogeneous nodes. Its performance was also compared to that of Hive and Spark-SQL. The results show that the data unification in Arvand leads to a major improvement in execution time.

In the proposed method, dices of cubes with different granularities are transferred to a shared nothing environment. According to the data source OLAP model, if there is a cube in the data source where measures are computed for all dimension combinations (MOLAP), then the space to store data does not change in the data source and Arvand format. However, if measures are not computed for all dimension combinations (ROLAP or HOLAP) in the source cube, the size of data on the destination increases to be equal to the size of the uncalculated measures per dimension combinations.

Data format unification can be applied to problems in other fields. This method can be used in data warehouse [26, 27], graph processing [23], data mining [24] and specific problems like finding patient similarity [25]. For future works, this method can also be used for interactive query processing, online data mining and stream processing.

## REFERENCES

1. A. Cuzzocrea, L. Bellatreche, and I. Y. Song, "Data warehousing and OLAP over big data: current challenges and future research directions," in *Proceedings of the 16th ACM International Workshop on Data Warehousing and OLAP*, 2013, pp. 67-70.
2. E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance evaluation of a mongodb and hadoop platform for scientific data analysis," in *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*, 2013, pp. 13-20.
3. D. Han and E. Stroulia, "A three-dimensional data model in hbase for large time-series dataset analysis," in *Proceedings of the 6th IEEE International Workshop on Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*, 2012, pp. 47-56.
4. T. Vajk, P. Feher, K. Fekete, and H. Charaf, "Denormalizing data into schema-free databases," in *Proceedings of the 4th IEEE International Conference on Cognitive Infocommunications*, 2013, pp. 747-752.
5. E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance evaluation of a mongodb and hadoop platform for scientific data analysis," in *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*, 2013, pp. 13-20.
6. M. Chevalier, M. El Malki, A. Kopliku, O. Teste, and R. Tournier, "Implementation of multidimensional databases with document-oriented NoSQL," in *Proceedings of International Conference on Big Data Analytics and Knowledge Discovery*, 2015, pp. 379-390.
7. M. Chevalier, M. El Malki, A. Kopliku, O. Teste, and R. Tournier, "How can we implement a multidimensional data warehouse using NoSQL?" in *Proceedings of*

*International Conference on Enterprise Information Systems*, 2015, pp. 108-130.
8. K. Dehdouh, F. Bentayeb, O. Boussaid, and N. Kabachi, "Using the column oriented NoSQL model for implementing big data warehouses," in *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*, 2015, p. 469.
9. M. Y. Santos and C. Costa, "Data warehousing in big data: from multidimensional to tabular data models," in *Proceedings of the 9th International Conference on Computer Science and Software Engineering*, 2016, pp. 51-60.
10. C. Doulkeridis and K. Nørvåg, "A survey of large-scale analytical query processing in MapReduce," *The VLDB Journal*, Vol. 23, 2014, pp. 355-380.
11. M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson, "CoHadoop: flexible data placement and its exploitation in Hadoop," in *Proceedings of the VLDB Endowment*, Vol. 4, 2011, pp. 575-585.
12. Y. Lin, D. Agrawal, C. Chen, B. C. Ooi, and S. Wu, "Llama: leveraging columnar storage for scalable join processing in the MapReduce framework," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2011, pp. 961-972.
13. S. Chen, "Cheetah: a high performance, custom data warehouse on top of MapReduce," in *Proceedings of the VLDB Endowment*, Vol. 3, 2010, pp. 1459-1468.
14. Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu, "RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems," in *Proceedings of the 27th IEEE International Conference on Data Engineering*, 2011, pp. 1199-1208.
15. A. Floratou, J. M. Patel, E. J. Shekita, and S. Tata, "Column-oriented storage techniques for MapReduce," in *Proceedings of the VLDB Endowment*, Vol. 4, 2011, pp. 419-429.
16. A. Jindal, J. A. Quiané-Ruiz, and J. Dittrich, "Trojan data layouts: right shoes for a running elephant," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011, p. 21.
17. H. C. Yang, A. Dasdan, R. L. Hsiao, and D. S. Parker, "Map-reduce-merge: simplified relational data processing on large clusters," in *Proceedings of ACM SIGMOD international conference on Management of Data*, 2007, pp. 1029-1040.
18. D. Jiang, A. K. Tung, and G. Chen, "Map-join-reduce: Toward scalable and efficient data analysis on large clusters," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 23, 2011, pp. 1299-1311.
19. L. C. Scabora, J. J. Brito, R. R. Ciferri, and C. D. D. A. Ciferri, "Physical data warehouse design on NoSQL databases OLAP query processing over HBase," in *Proceedings of International Conference on Enterprise Information Systems*, *XVIII*, 2016, pp. p. 111-118.
20. B. Martinho and M. Y. Santos, "An architecture for data warehousing in big data environments," in *Proceedings of the 10th IFIP WG 8.9 Working Conference on Research and Practical Issues of Enterprise Information Systems*, 2016, pp. 237-250.
21. M. Y. Santos and C. Costa, "Data warehousing in big data: from multidimensional to tabular data models," in *Proceedings of the 9th ACM International Conference on Computer Science and Software Engineering*, 2016, pp. 51-60.
22. P. Atzeni, F. Bugiotti, L. Cabibbo, and R. Torlone, "Data modeling in the NoSQL

world," *Computer Standards and Interfaces*, 2016.
23. M. Barkhordari and M. Niamanesh, "ScaDiGraph: A mapreduce-based method for solving graph problems," *Journal of Information Science and Engineering*, Vol. 33, 2017, pp. 143-158.
24. M. Barkhordari and M. Niamanesh, "ScadiBino: An effective MapReduce-based association rule mining method," in *Proceedings of the 16th ACM International Conference on Electronic Commerce*, 2014, p. 1.
25. M. Barkhordari and M. Niamanesh, "ScaDiPaSi: an effective scalable and distributable MapReduce-based method to find patient similarity on huge healthcare networks," *Big Data Research*, Vol. 2, 2015, pp. 19-27.
26. M. Barkhordari, M. Niamanesh, "Aras: A method with uniform distributed dataset to solve data warehouse problems for big data," *International Journal of Distributed Systems and Technologies*, Vol. 8, 2017, pp. 47-60.
27. M. Barkhordari and M. Niamanesh, "Atrak: A Map-Reduce based warehouse for big data," *The Journal of Supercomputing*, 2017, DOI: 10.1007/s11227-017-2037-3.
28. https://redis.io/.
29. https://www.postgresql.org/.
30. http://Spark.apache.org/.
31. https://hive.apache.org/.
32. http://www.ubuntu.com/download/server.
33. http://hadoop.apache.org/.
34. http://www.tpc.org/tpcds/.
35. http://www.tpc.org/TPC_Documents_Current_Versions/download_programs/tools-download-request.asp?bm_type=TPC-DS&bm_vers=2.3.0&mode=CURRENT-ONLY

**Mohammadhossein Barkhordari** received the M.S. degree in Software Engineering from Amirkabir University, Iran. He is currently pursuing the Ph.D. degree in the Information and Communication Technology Research Center, Iran. His research interests include big data, business intelligence, data warehouse, data mining.

**Mahdi Niamanesh** received his Ph.D. degree in Computer Engineering in the Department of Computer Engineering, Sharif of University Technology in 2009. He is currently a Professor at the Information and Communication Technology Research Center, Iran. His research interests include algorithm pervasive computing, big data.