Issues in Information Systems
Volume 13, Issue 2, pp. 51-57, 2012

DISTRIBUTED MULTI-LEVEL QUERY CACHE: THE IMPACT ON DATA WAREHOUSING

Philip D. Noah Jr., Robert Morris University, philipdnoah@gmail.com
Sadry Abu Seman, Robert Morris University, muhdsadry@gmail.com

ABSTRACT

There has been an explosion in the amount of data companies are keeping in data warehouses and data-marts. The Internet and e-commerce are a large source of this data; however, up to 80% of the data generated by e-commerce never finds its way into the data warehouse. This means potentially valuable information is never analyzed. In this paper we propose a method called DMQC, Distributed Multilevel Query Cache, which supplements the traditional data warehouse and provides a less costly method for storage and analysis of data. To understand DMQC we first give a brief overview of how distributed and federated data warehouses systems work. Next we present an overview of the database systems DMQC is to be built on and how DMQC differs from existing distributed query cache systems. We conclude with the benefits of using DMQC and areas for further research.

Keywords: Data integration cache engine, Data warehousing, distributed computing, Federated search, Query cache, Big data.

INTRODUCTION

Ever since Bill Inmon coined the term "Data Warehouse" companies have been struggling to consolidate and manage vast amounts of data and to glean information that would give them a competitive advantage. Companies have the choice of building large all-encompassing data warehouses or implementing subject-oriented data marts. Both methods require extraction of data from production systems, transformation of the data, and loading it into the data warehouse or data mart. For the last 30 years the trend in data warehousing (DW) has been to use the "top-down" or "bottom-up" approach to create a data warehouse system separate from the transactional database. The top-down approach promotes the design of a large database system that extracts data from one or more transactional systems [9]. The data is Extracted, Transformed, and Loaded (ELT) into the DW. The user queries the DW using a front-end that converts the request to SQL and presents the data to the user. In the bottom-up approach supported by Kimball, smaller subject ordinated "data marts" are created. The data-marts are put to use as soon as they are built, the data-marts are connected by a "bus" that links the marts together based on integration points to create a DW [20]. Like the top-down approach, the bottom-up approach relies on a front end to convert the users request into SQL and to display the returned data. In both approaches, the DW is designed with the end in mind and as a large separate database system consisting of processing and storage separate from the transactional systems.

The purpose of this paper is to explore how the use of query caching and a federated approach can supplement existing data warehouses and data marts as well as being a low-cost alternative to them. Our approach replaces SQL and ELT with the eXtendable Markup Language (XML) and Distributed Multi-level Query Cache (DMQC) respectively. XML is an industry standard for quires and DMQC caches frequently used request for reuse.

BIG DATA

When Inmon and Kimball conceived of the data warehousing architectures in 1980s data warehousing like most data processing at the time, was done on large mainframe or mid-range computer systems. The data stored in the data warehouse was typically scalar numbers and text that was extracted from transaction processing systems. In the late 1990s there was a change in the type of data that businesses wanted to store and analyze. The Era of Big Data had started. Kimball defines Big Data as

> [S]tructured, semi-structured, unstructured, and raw data in many different formats… But most important, the data is a paradigm shift in how we think about data assets, where do we collect them, how do we analyze them, and how do we monetize the insights from the analysis [11, p. 2].

The shift to big data is a result of the Internet age and the richness of data that can be collected and extracted through e-commerce transactions. It is not only customer-purchasing transactions that are collected but also click throughs, sub-transactions, and social media data [11]. The mainframe systems on which data warehousing was built could not incorporate unstructured data and multimedia data, which pushed data warehouses size into the terabyte range. The architecture of the mainframe could not scale to meet the needs of modern computer age. A new highly scalable, highly available, and low latency, system was needed.

PARALLEL DATABASE SYSTEMS

To overcome the limitations of the mainframe and to handle Big Data parallel processing systems and techniques were developed starting in the mid-1990s. Parallel processing system are very scalable, can handle large data sets and are very efficient at processing transactional data loads, and creating auxiliary structures [7]. There are several types of parallel systems, each with its own benefits and drawbacks. The purpose of this paper is not to provide a detailed analysis of parallel system, but a brief over-view is needed to understand our proposed approach to data warehousing and how it differs from current parallel and distributed systems.

Parallel systems are defined by the components they share; the three major components are the CPU (processor), memory, and disk storage [7, 22]. The first parallel systems were the "shared everything" system, which is a system with multiple processors that have shared disk and memory [22]. In the shared everything system the CPU's can process data at the same time from a shard memory pool and disk array. The shared everything system is very similar to the mainframe and its expandability is limited by the architecture design of the hardware. The shared disk system is made up of several servers with their own CPU's and memory and a shared disk storage sub-system [7, 22]. Each CPU processes data in its own individual memory space, but the database resides on a shared disk system. Each of the servers is connected to the disk system by a dedicated interconnection network [7]. The advantage of the shared disk system is that it can easily be expanded by adding more servers or disk-subsystems with minimal software configuration changes to the existing systems.

GRID AND DISTRUSTED DATABASE SYSTEMS

The grid and distributed database systems are both types of the shared nothing architecture. In a shared nothing architecture each computer, which is called a node has its own processor, memory, and data [7, 22]. The grid and distributed database systems are interconnected through a network, unlike a parallel system. It is how this network is configured that distinguishes the grid from the distributed system.

Grid Systems

A grid system is typically a group of several commodity or special purpose built servers linked through a common network typically within the same server rack or data center. The interconnecting network can be as simple as a network switch or as complex as a high-capacity interconnected backplane. Variations of the grid system include systems, such as the EMC/Aciom data grid architecture that incorporates a shared data storage system [19]**.** The Cray XMT supercomputer is a grid-based system that uses a purpose built processor that supports 128 parallel processing threads with each node possessing up to 512 processors. The Cray XMT differs from other grid systems because the XMT has a global shared memory, which lowers the overhead of sending and receiving small messages as well as locking memory access. Locking memory access allows the cluster to do fetch and add type of operations on every word in the memory and thus increases performance when handling unstructured data [3].

Distributed system

A distributed system is an interconnected collections of heterogeneous computers typically distributed geographically and connected by a local area network, and a wide area network [7]. Some distributed systems are connected via the Internet, which allows more computers to be part of the distributed system however; it creates several problems, including speed of the Internet connection limiting performance and data not being available if a node or a link is down.

FEDERATED DATABASE SYSTEMS

A federated database system sends a search query to multiple databases, organizes, and returns the results to the end-user [4]. The federated database system is similar to the distributed system in, which data is spread across multiple servers using different database systems. In the federated architecture the query is converted to XML, which is a universal query language supported by most database vendors, such as Oracle, IBM, and Microsoft. The advantage of the federated database system is that it allows existing data warehouses and data marts to stay in place while supplementing them with data from legacy systems as well as the Internet [20]**.** According to Eckerson, "because of performance and data quality issues, most experts agree that Federated approaches work well to supplement data warehouse , not replace them" [5]. We will show that with the use of XML and query caching that federated database systems are viable alternative as well as a supplement to data warehouses and data marts.

DATA WAREHOUSE ARCHITECTURE

In a typical setup of a DW, a single database is dedicated to stores a copy of several transactional databases within the company and is optimized for query and analysis [17, 5, 16]. Specialized information is extracted from DW and shaped into data-mart, which defines a specific scope of a well bounded business problem. Data-marts are clustered into specific data areas pertaining to a department or unit of operation in the company for reporting and analysis of the company business. Thus, based on the logical architecture of a DW is categorized into four levels that consist of operational, DW, data-mart and user level.

This architecture is evolving because of the latest technologies available that allow DW to store data from various sources. For example, the virtual operational data store (VODS) allows answers to queries and unstructured data that combine corporate communications and transactions to create corporate information factory (CIF) [10].The core engine of DW is the ELT, which updates the DW with the new data copied from multiple sources. The extracting process extracts data from multiple transactional or operational databases; the transforming process allows data to be cleaned, transformed, and integrated into DW and the loading process ensure data is properly loaded into DW [2, 13, 18]. In the reporting process, On-Line Analytical Processing (OLAP) is required to arrange data in a cube for fast data analysis from multiples perspectives. The idea of OLAP is to overcome the limitation of data arrangement in relational databases and allow instant analysis on huge amount of data in the DW [17, 20,16].

PROPOSED DATA WAREHOUSE WITH DMQC IMPLEMENTATION

The utilization of DMQC in DW implementation has changed a few components that exist in a typical DW setup. The proposed DW with DMQC implementation is shown in Figure 1. The architecture of the DW remains the same but only frequently access cache queries are copied into DW [1, 12, 6]. These copied cache queries will become an agent to access frequent data from operational databases for user analysis and reporting. Thus, this would allow an instant query to the operational databases with the latest data. The idea of replacing the historical data with cache queries from operational databases is to eliminate replication time and duplication of denormalized data into DW. Moreover, a low-cost server can be used as DW rather than an expensive dedicated server for DW implementation. The conversion of data normally perform by ELT is replaced by XML, a standard data conversion widely used by various database systems. Furthermore, a specific subject or business scope is stored in several data marts to create a cube for multiple perspectives analysis. In a data mart, specific cache queries pertaining to the subject (e.g., accounting) is copied into dedicated data mart from DW and a low-cost server can be used as a data mart. At the

user level a simplified customize HTML-based query is used to perform data analysis and reporting. There is also a possibility of OLAP implementation for data analysis. The DMQC uses Java in order to ensure cross platform implementation throughout multiple servers. We believe that the performance increase from the multi-level cache and the flexibility of being platform independent will outweigh the overhead caused by using Java and the Java virtual machine.

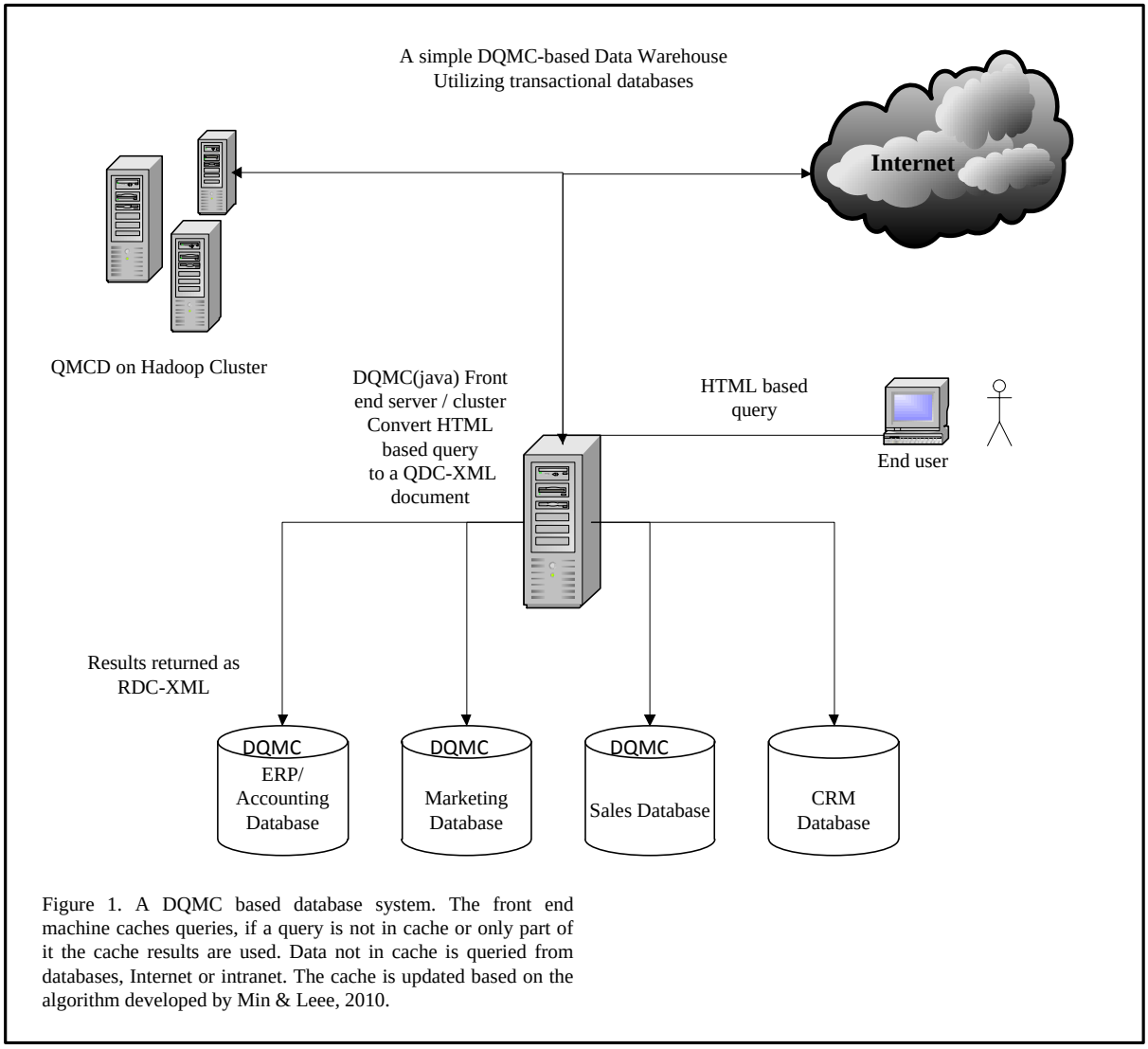

Figure 1. A DQMC based database system. The front end machine caches queries, if a query is not in cache or only part of it the cache results are used. Data not in cache is queried from databases, Internet or intranet. The cache is updated based on the algorithm developed by Min & Leee, 2010.

The benefit of adding DMQC to the database is the ability to reuse the caches of previous query as a current query to gather frequently accessed data. This allows instantaneous access for future queries [15]. DMQC adaptively selects the latest query based on monitored statistics to remove old queries that are no longer needed [15]. Thus, the use of DMQC in distributed systems does not required expensive hardware to implement, and it works on any existing databases because of the use of individual database cache. Furthermore, the implementation of DMQC does not disrupt scalability and reliability of the existing databases that exist in the distributed system [15]. The advantage of cache use in distributed systems is the high performance in data retrieval between servers and clients. It also reduces communication costs and off-loads shared data between servers [12, 6]. Therefore, the implementation of DMQC in

a DW eliminates extra hardware cost and makes use of the existing distributed system to perform similar storage of historic data through replication of frequently used query in a low-cost server. Typically, an ELT implementation in DW is expensive and requires a long learning curve [20]. With the implementation of DMQC in DW the middleware used in ELT is eliminated through the use of XML conversions. Moreover, it shortens the learning curve because XML is readily available and widely supported by database vendors.

RELATED WORK

BigInsight™/Hadoop

IBM's BigInsight™ is a distributed database system and data analytics tool for the Linux operating system. Like DMQC it is meant to be used as a low-cost supplement to an existing data warehouse or data marts. BigInsight™ runs on top of the open-source distributed database Hadoop, which like DMQC uses low-cost commodity servers [IBM]. BigInsight™ only partially eliminates the need for ELT, data not in the Hadoop database stills needs to be extracted and loaded into BigInsight™. DMQC completely eliminates the need for ELT by leaving the data on the native system and only caching recent queries on the front-end server. Both DMQC and BigInsight™ utilize "data locality" where the data is processed on the system it is stored on, thus reducing the amount of data transferred between systems when a query requires data from more than one source. DMQC and BigInsight™ differ in how they use data locality. BigInsight™ breaks queries down into small pieces and executes each piece on a different server, thus spreading the processing load and utilizing parallel processing when possible. DMQC uses a Java application to create a query cache on the local machine to keep recently used queries in memory, thus reducing processing time.

DICE – Distributed Interval Cache for a rEgion server/Glory-DB

DICE is a query cache application, developed by Min and Lee that runs on top of the Glory-DB distributed system [15]. Like Hadoop, Glory-DB is a column-based system however, with the use of DICE a cache is used instead of a secondary index for the columns [15]. With DICE new queries are compared with cached queries and cache results are used when there is a full match or a piratical match [15]. Because of the amount of data in large distributed databases DICE uses a statistical analysis to determine which queries to cache and a modified interleave skip algorithm to determine when to purge the cache [15]. Each of the servers in the cluster contains its own cache and a sub-set of the total data in the database system [15]. The data servers are called Region Servers, and a Master server oversees the query process. With DICE a client system sends a query to the master and the master sends a broadcast message to the Region servers to perform the lookup. If the Region server has the data in cache the results are sent directly back to the client. If the results are not cached the Region server looks up the data, updates the cache, and sends the result to the client. The region servers use an interval skip list algorithm to provide an index structure for the intervals, which are columns of data [15]. DICE differs from DMQC in the way the query caching occurs. In DMQC the front-end server stores cached results, in DICE the master server does not, only the region servers do. By having the host server store cached data results can be returned faster to the client, there is no need to send the query to the server that has the data stored on it. This is an important feature of DMQC because the database servers may not be on the same LAN or WAN as the client or front-end server. DICE is limited to running on top of Glory-DB, whereas DMQC is platform and database independent. With DICE data needs to be in a native Glory-DB, if it is not it has to be Extracted, Loaded, and Transformed into the Glory-DB. The use of ELT adds complexity and cost to the system, making it less flexible and harder to add data from new sources. Although it is true that Java and the conversion to and from XML in DMQC adds overhead we believe the multi-level cache and the elimination of ELT will make the over-all performance of DMQC equal to, or better than DICE.

BENEFITS OF DMQC

A data warehouse is crucial to business analysis and reporting but the implementation of a data warehouse is subject to design and cost. Simitsis and Theodoratos stated that an in-house DW project consumes up 70% of resources because of DW implementation is data intensive, complex and expensive in term of server and ETL selection [17].

However, with the introduction of DMQC, the cost of implementation of DW is lowered and can be implemented by using low-cost hardware and utilizing the existing feature of the available technologies. In the proposed DW implementation with DMQC, the replacement of an expensive ELT is done through the implementation of XML as data conversion and middleware tools are replaced with simple HTML-based query for the end user. . The SQL queries from multiple database servers are converted into XML and stored at DW. Liu and Krishnamurthy stated that the use of XML in database extraction allows DW to query and store structured, semi-structured and unstructured data [14]. The heart of DMQC implementation is the high performance of data access, which allows data to be accessed almost instantaneously through the use of frequently used cache queries. This eliminates the need for huge data duplication into DW and ensures the availability of fresh data. Min and Lee's experimental results show that "DICE [has] the best hit ratio over most of the [experimental] cases since, unlike other methods, the cache replacement policy of DICE considers the length of the query interval and the number of a query result" [15, p. 941]. Min and Lee found that "the hit ratio of DICE is twice of that of other techniques when the cache size is 40% for skewed disc/skewed query"[15, p. 941]. Their experimental results show that with the cache size of 60% the cache ratio is from 35% to 80% [10]. Earlier work on distributed cache for database queries by Triantafillou et. al. support the findings of Min and Lee. Triantafillou et.al. found that query caching achieves a 50% decrease in the user–perceived turnaround time for query and an average 40% decrease in network bandwidth [20]. In general, Garrod et. al. concluded that "the use of query cache eliminates the server load from the centralized infrastructure and reduced the number of requests send to the backend database server"[8 et. al. 2008].We expect to see similar performance increases in DMQC as in DICE since DMQC is built around a similar cache management system as DICE.

FURTHER RESEARCH

Despite the advantages of DMQC in the proposed architecture, there are a few considerations that require further investigation. The stored cache queries in the DW are based on queries made at the operational databases and specifically stored into an organized data mart. These queries are based on assumptions at the operational level not at the top level. Thus, any queries requested at top level or user level that are not available at the data mart or DW will force the DW to run a new query that will require extra time to retrieve before the cache query is stored in the DW. This is the proposed conceptual architecture, and even though it is possible to be implemented, it has yet to be applied in the current business scenario. The use of HTML-based query as user interface to data marts and DW is for simplification purposes. The use of advance middleware tools, such as OLAP, data mining, reporting tools, and data visualization tools can be explored further for implementation.

CONCLUSION

In this paper we have presented the two prevailing methods of storing and retrieving data for analytical process, the top-down data warehouse approach developed by Inmon and the bottom-up data mart approach developed by Kimball. We showed that as a result of the Internet and rich content that new approaches to the storage and retrieval of data is needed. We presented a brief overview of parallel computing systems and grid computing systems and how their ability to scale in terms of storage and computing power was needed to handle "Big Data". From the parallel and grid computing systems the distributed computing environment developed and led to federated systems with their geographically dispersed databases. We showed how the use of DMQC can be applied to either the data warehouse or the data mart as a way to supplement both with information drawn from legacy or other systems. We postulated that a well-designed system using DMQC could be a viable alternative to both the data warehouse and the data mart. The experimental results from Min and Lee along with the earlier research by Triantafillou et. al. showed that DICE does significantly improve the performance of database queries. We concluded this paper by presenting areas that need further experimental research before this proposed method can be accepted as a reliable alternative to the traditional methods of data warehousing.

REFERENCES

1. Adai, S., Candan, K., Papakonstantinou, Y., & Subrahmanian, V. (1996). Query caching and optimization in distributed mediator systems. *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, *25*(2), 137-148. doi:10.1145/235968.233327
2. Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and OLAP Technology. *ACM SIGMOD*, *26*(1), 65-74.
3. Cray Supercomputer Company. (2012) Meeting the Demands of Unstructured Data with an eXtreme MultiThreaded Machine. Retrieved from http://www.cray.com/Assets/PDF/products/xmt/CrayXMTOverviewWhitepaper.pdf
4. Curtis, A., & Dorner, D. G. (2005). Why federated search? *Knowledge Quest, 33*(3), 35-37.
5. Eckerson, W. (2005, April 1). Data warehouse builder's advocate for different architectures. *Application development trends.* Retrieved from http://adtmag.com/articles/2005/04/01/data-warehouse-builders-advocate-for-different-architectures.aspx
6. Folinas, N., Vassiliadis, P., Pitoura, E., Papapetrou, E., & Zarras, A. (2008). Context-aware query processing in ad-hoc environments of peers. *Journal of Electronic Commerce in Organizations*, 6(1), 38-62.
7. Furtado, P. (2009). A survey of parallel and distributed data warehouses. *International Journal of Data Warehousing and Mining*, *5*(2), 57-77.
8. Garrod, C., Manjhi, A., Ailamaki, A., Maggs, B., Mowry, T., Olston, C., & Tomasic, A. (2008). Scalable query result caching for web applications. *Proceeding of VLDB Endowment*, *1*(1), 550-561. doi:10.1145/1453856.1453917
9. Inmon, W. H. (2005b). *Building the data warehouse* (4th ed.). Indianapolis, IN: Wiley Publishing, Inc.
10. Inmon, W.H. (2005a). A brief history of architecture. *Information Management, 15*(4), 48-48.
11. Kimball, R. The Evolving Role of the Enterprise Data Warehouse in the Era of Big Data Analytics. Retrieved February 15, 2012, from http://www.informatica.com/downloads/1597_EDW_Big_Data_Analytics_Kimball.pdf
12. Kossmann, D., Franklin, M., & Drasch, G. (2000). Cache Investment: Integrating Query Optimization and Distributed Data Placement. *ACM Transactions on Database Systems*, *24*(4), 517-558.
13. Liao HM, Pei GS. Cache-based aggregate query shipping: An efficient scheme of distributed OLAP query processing. *Journal Of Computer Science And Technology* 23(6): 905-915 Nov. 2008.
14. Liu, Z. H., & Krishnamurthy, V. (2012). *Business intelligence – solution for business development*. M. Mircea (Ed.). Rijeka, Croatia: InTech.
15. Min, J., & Lee, M. (2010). DICE: An effective query result cache for distributed storage systems. *Journal of Computer Science and Technology*, *25*(5), 933-944. doi:10.1007/s11390-010-9378-1
16. Reddy, G., Srinivasu, R., Rao, M., & Rikkula, S. (2010). Data warehousing, data mining, olap and oltp technologies are essential elements to support decision-making process in industries. *International Journal On Computer Science & Engineering*, 2865-2873.
17. Samos, J., Saltor, F., Sistac, J., & Bardes, A. (1998). Database architecture for data warehousing: An evolutionary approach. Proceedings from DEXA '98: *The 9th International Conference on Database and Expert Systems Applications*. London, UK: Springer-Verlag.
18. Simitsis, A., & Theodoratos, D. (2006). *Encyclopedia of data warehousing and mining*. J. Wang (Ed.). Hersey, PA: Idea Group Reference.
19. Thompson, C. (2006, June). Towards a grid-based DBMS. *IEEE internet Computing*, *10*(3), 87-90.
20. Triantafillou, P., Ntarmos, N., N.Ntarmos., & Yannankopoulos, J. (2003, December). D.I.C.E. and Co.In.S: A Data Intergration Cache Engine for a Content intergration system. *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference*, 339 - 342. doi:10.1109/WISE.2003.1254507
21. Turban, E., Sharda, R., Delen, D., & King, D. (2011). *Business intelligence; A managerial approach* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.
22. Velicanu, M., Litan, D., & Mocanu (Vîrgolici), A. (2010). Some considerations about modern database machines. *Informatica Economica*, *14*(2), 37-44.