International Journal of Security and Its Applications
Vol. 11, No. 6 (2017), pp.53-68
http://dx.doi.org/10.14257/ijsia.2017.11.6.05

Data Warehouse Signature: High Performance Evaluation for Implementing Security Issues in Data Warehouses through a New Framework

Mayada J. AlMeghari

Information Systems Department, Faculty of Computers and Information, Cairo University, Egypt
mayada@teachers.org, malmeghari@ptcdb.edu.ps

Abstract

In this globalized world, the Internet services and uses are growing exponentially. Hence, the data stores contain a huge amount of business data used for decision-making and for financial analysis at sensitive organizations called Data Warehouses (DWs). DW is necessary for financial and business information making them an attractive purpose for hackers. The achievement of security issues in DW is very important for a proper and secure continuation of DW system work. This paper presents a new framework for implementing security issues in DWs named Data Warehouse Signature (DWS), which is distributed in two models: DWSend model and DWReceive model. The DWS framework solves one of the common security problems such as unavailability in network by using parallel computing through a middleware named View Manager Layer (VML). This framework ensures the security issues, such as Confidentiality, Integrity, and Availability (CIA) and it also reaches high performance in Average Execution Time (AET) evaluated in experimental studies. The execution of a large query result as blocks of a number of records in parallel computing saves more time than serial computing. The high performance has a limited increase in executor's numbers because there are time complexity factors, such as transmission time, separation and collection time. This paper presents a mathematical model used when the organization applies the DWS framework in DW systems to get the adequate number of executors joined in VML middleware to reach the high performance.

Keywords*: Data Warehouse, Security issues, Data Encryption, Data Integrity, Data Availability, Hash Function, Middleware, Performance, Information Security*

1. Introduction

A Data Warehouse (DW) is considered a core component of business intelligence environment just as decision-making and data mining. DW system is used for reporting and analyzing the huge amount of data without difficulty in accessing them for the users. The common DWs are used in sensitive organizations such as banking system, health care system, and financial marketing system. To make DW system more secure, the security issues such as Confidentiality, Integrity and Availability (CIA) must be ensured.

The flow process of transmission between a client and DWServer takes a lot of time since the query result has a huge amount of data. This makes it difficult to transfer the data securely and fast. The problem of security has appeared due to the longtime of encryption/decryption besides its transmission time. Moreover, the data integrity is needed as a security property to ensure data stability during the transmission process by checking the hash code. This is an additional load to the encryption/decryption process, which leads to a crisis in the server side. Thus, it can't continue to provide services to users as overhead in the network. In this case, sending/receiving process between DWServer and the client is a response timed-out.

ISSN: 1738-9976 IJSIA

In the distributed systems, the middleware can use the network resources such as CPUs as virtual supercomputer. This technique is able to perform encryption process and to compute hash code process in parallel to get the Average Execution Time (AET) in time shorter than that is needed in traditional approaches. As mentioned earlier, this paper presents a framework named Data Warehouse Signature (DWS), which includes two models: DWSend model and DWReceive model. This approach achieves data availability in DW service system and eliminates the query response time of DWServer. It can be developed by using a middleware called View Manager Layer (VML), which executes the security processes in parallel. Also, the DWS approach ensures data integrity in DW systems by using the hash code function.

The remainder of this paper is organized as follows: Section 2 presents the related work. Section 3 presents the architecture of DWS framework with its flow process structure. Section 4 presents the experimental studies environment. Section 5 presents how to determine the edge of query memory buffer for VML Middleware. Section 6 presents the high performance evaluation including the experimental results for the query load balance in the two models and determining the adequate number of executors through creating a mathematical model. Finally, Section 7 presents the main conclusion and future work.

2. Related Work

A number of researchers and those interested in DW security have provided many approaches to make frameworks and models solve the security issues in DWs. In essential orientations, these approaches are covered in two categories depending on applying availability or not [1]. The first category has offered some of the DW security approaches ensuring confidentiality and integrity of data in DW systems as presented in [2-6].

The second category has offered other DW security approaches that ensure confidentiality and availability of data without ensuring data integrity as presented in [7], [8-10]. These approaches used different types of middleware as cluster computing, grid computing and cloud computing for resisting overhead and query response timed-out.

Based on the DW security approaches of the two categories, this opens the gate for using digital signature as a new framework of DWS to ensure the three security issues CIA in DWs [11]. The DWS framework uses a middleware, VML that is an elastic network configuration for using any type of organizations.

3. DWS: Data Warehouse Signature Framework for Implementing Security Issues in DWs

The DWS framework is designed by using Server-Client architecture model, as shown in Figure 1. From the data flow between DWServer and the client, there are two process models in the DWS framework. These models are DWSend model and DWReceive model joined in the VML middleware [11]. The VML middleware is supported by .Net Microsoft Framework as Alchemi Desktop Enterprise [12]. This middleware consists of three elements of network nodes, which are the owner (DWServer or Client), the manager and the executors. In VML middleware, the owner separates the Query Result Table (QRT) or the Encrypted QRT (EQRT) into a small number of records as block objects. These blocks are sent to the manager to be distributed to a set of executors. The chosen executors perform encryption/decryption process and compute the hash code process in their blocks in a parallel execution.

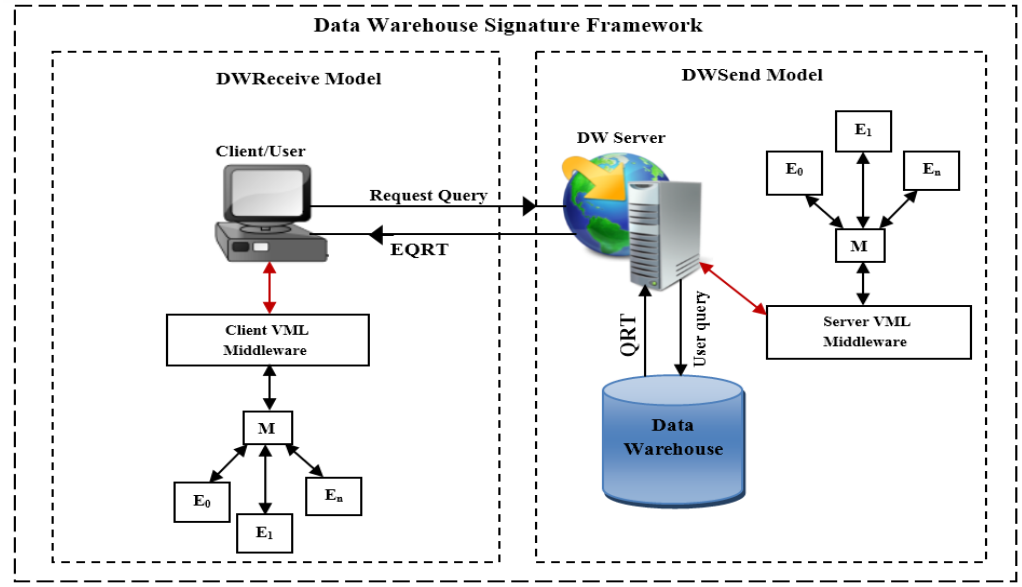

Figure 1. The DWS Framework Architecture

There are three types of network computing distributed through its network topology. The first type is the cluster computing, which connects the network nodes in LAN network. The second type is the grid computing, which connects the network nodes between LAN and LAN or LAN with WAN. Finally, the cloud computing connects the network nodes between WAN and WAN networks. The DWS framework contains two flow process structures based on its two models. These structures are presented as follows:

3.1. DWSend Flow Process

As shown in Figure 2, there are three main types of flow processes for DWSend structure depending on DWS architecture components. These types are described below:

A) In the first type, a set of processes are executed in the owner (DWServer). These processes are ordered as follows:

1. Get QRT from DW.
2. Choose a number of records as a block object.
3. Read QRT as data block objects.
4. Assign a thread for every block object.
5. Start the thread.
6. Wait for the thread to finish.
7. Make XOR result to other to generate the computed hash for all encrypted block objects to get the Hash of Encrypted Block Objects (HEBOs).
8. Encrypt the HEBOs by using RSA algorithm [13][14] with the private key (K_{Pr}) of DWServer to get the Encrypted Hash of Encrypted Block Objects (EHEBOs).
9. Write EQRT || EHEBOs as a view table.
10. Calculate execution ttime.

A) In the second type, a set of processes are executed in the manger and ordered as follows:

1. Get the thread from the owner.
2. Assign the thread to an executor.
3. Pool finish thread.
4. Send the results to the owner. These results are:

- The hash value for each Encrypted Block Object (EBO).
- The EBOs are recollected to become one block object as EQRT.

B) In the third type, a set of processes are executed by each executor and ordered as follows:

1. Get a thread from the manager.
2. Encrypt the data of block object as number of records by using AES algorithm[15] with the share key(K_{SK}), which is distributed by using Diffie-Hellman (D-H) algorithm [16].
3. Execute the hash code to compute the hash function for each EBO using SHA-1 algorithm [17-18].
4. Return the results to the manager.

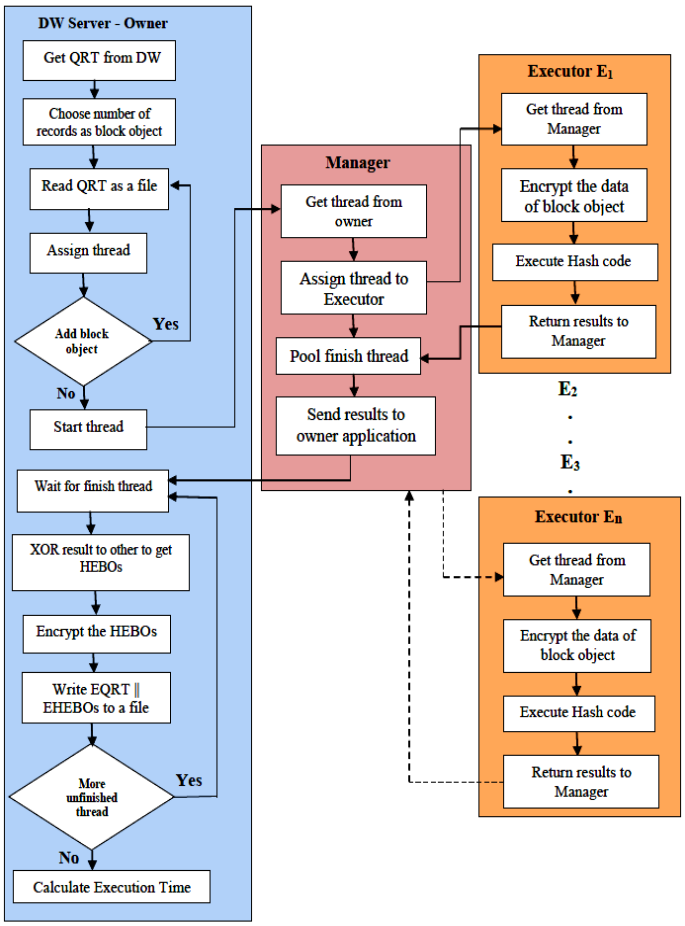

Figure 2. The Flow Process Structure for Dwsend Model in DWS Framework

3.2. DWReceive Flow Process

The structure of DWReceive model is also separated into three types of flow processes, as shown in Figure 3. These types are described below:

A) In the first type, a set of processes are executed in the owner (Client) and ordered as follows:

1. Separate the EQRT from the EHEBOs, which are received from the DWServer.
2. Decrypt the EHEBOs by using RSA algorithm with the public key (K_{Pu}) of DWServer [13][14].
3. Read the chosen number of records as block objects in the DWSend model.
4. Read the EQRT as data block objects.
5. Assign a thread for each block object of number of records.
6. Start the thread.
7. Wait for the thread to finsih.

8. Make XOR result to other to generate the computed hash for all encrypted block objects to get the Hash of Encrypted Block Objects (HEBOs).
9. Compare the calculated HEBOs with the received HEBOs.
10. Calculate the execution time.

B) In the second type, a set of processes are executed in the manger and ordered as follows:

1. Get the thread from application.
2. Assign the thread to an executor.
3. Pool finish thread.
4. Send the results to the owner. These results are:

- The hash value for each EBO .
- The decrypted block objects are recollected to get QRT.

C) In the third type, a set of processes are executed in each executor and ordered as follows:

1. Get the thread from the manager.
2. Execute the hash code to compute the hash function for each encrypted block object using SHA-1 algorithm [17][18].
3. Decrypt the data of each EBO by using AES algorithm [15] with the K_{SK} distributed by using D-H algorithm [16].
4. Return the results to the manager.

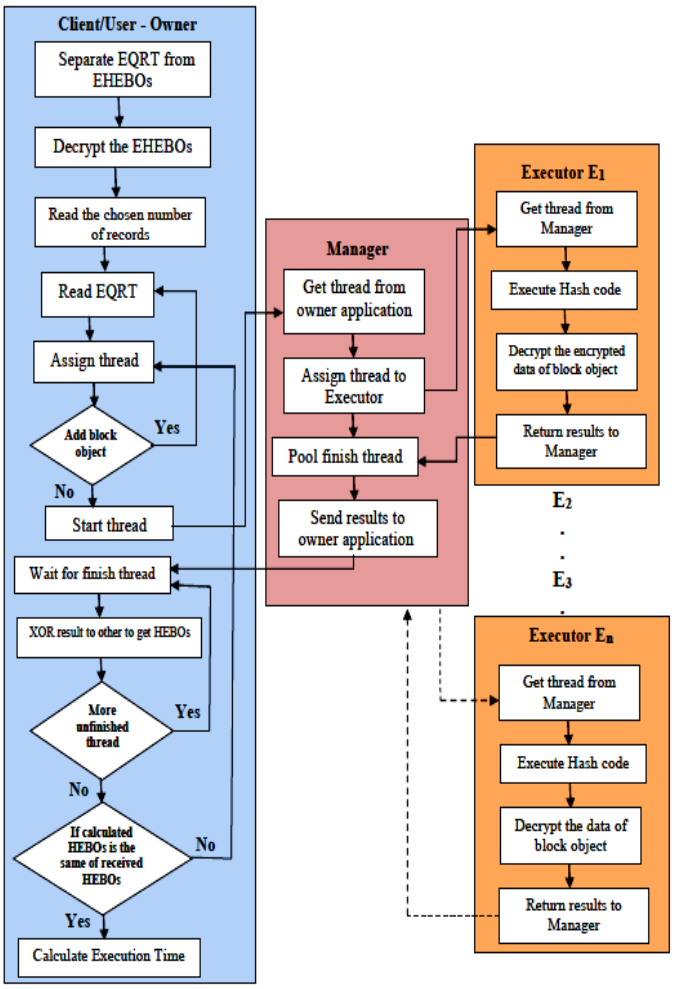

Figure 3. The Flow Process Structure for DWReceive Model in DWS Framework

4. Experimental Environment

The DWS framework used the VML middleware as the open source code of Alchemi grid computing Version 1.0.4 .Net Framework 2. We used a set of environment tools such as operating system, RAM size and CPU. The operating system is Microsoft Windows 7.0 (2009), Service Pack 1(SP1), System type 32-bits. The RAM size is 4GB at the manager node, and it is 2 GB at the owner and executors nodes. The CPU type is Intel Core™ 2 Duo for all nodes with different speeds, where the CPU speed at the manager is 3.10 GHz and is 2.80 GHz at the other nodes.

The network environment is CISCO TP-LINK Corporation with bandwidth estimated to 100 Mbit/s, where network topology is a star and cluster computing system. The database that managed the data flow in the VML middleware is Microsoft SQL Server 2000 Enterprise Edition. We used the data set of sales DW from an exported database file.

The programming language used for implementing DWS framework is C# language, which is edited by Microsoft Visual C# 2008 Express Edition. The classes and methods are included from Microsoft Visual Studio as Benchmark for all nodes in the DWS framework. Different algorithms can be used for executing a set of processes. The encryption/decryption process is applied by using AES algorithm of 128-bit block size with the K_{SK}, which is distributed by using D-H algorithm. The hash code function is computed by using SHA-1 algorithm with 160-bit hash code size. This hash code is encrypted/decrypted by using RSA algorithm with K_{Pr}/K_{Pu} of DWServer.

5. Determining the Edge of Query Memory Buffer for VML Middleware

We need to detect the memory buffer used for one job of thread of block object in the executors. First, we start the DWS experimental studies with different set of QRT sizes measured in Record unit (R) and executed on one executor. The QRT sizes are 10R (3 KB), 50R (11 KB), 100R (21 KB), 500R (115 KB), 1KR (220 KB), 5KR (1.037 MB), 10KR (2.043 MB), 50KR (10.672 MB), 100KR (20.387 MB), 500KR (102.336 MB), 1MR (204.406 MB). The results of executing these QRTs on one executor are shown in Table 1 and Figure 4. The AET is the average of ten execution times for executing each of these QRTs in a single machine and it is measured in a Second time unit.

The results show that the AETs take the same time when using QRT sizes of 10R, 50R, 100R, 500R, 1KR. This means that the AETs for these QRTs are consuming the same time and are constant. Depending on these results, the AET began to increase at QRT size of 5K records. This proves that Edge of Query Memory Buffer ($EQMB_f$) in the VML middleware is centered on 5KR (5000 records) with byte size 1.037 MB. Thus, the QRT sizes of 10R, 50R, 100R, 500R, 1KR will be ignored in our experimental studies.

In Alchemi V.1.0.4 .Net 2.0 Windows Framework, the Query Memory Buffer (QMB_f) size is equal to 20 MB and the $EQMB_f$ size is equal to 1 MB for each job of thread [19]. The VML middleware uses the same structure of Alchemi with converting the size in bytes to the size in records. According to the above results, the $EQMB_f$ in the VML middleware is centered on 5KR with byte size 1.037 MB. Thus, we find that the $EQMB_f$ (1MB) is equal to 4822 records (5000 / 1.037), and the QMB_f (20MB) is equal to 96440 records (4822*20) in the VML middleware. By using the VML query memory buffer, we can explain when the AET can reach the high performance as presented in next section.

Table 1. The AETs for Executing a Set of QRTs $_{\text{Record Size}}$ to Determine the $EQMB_f$

QRT (Record)	10 R	50 R	100 R	500 R	1 KR	5 KR	10 KR	50 KR	100 KR	500 KR	1 MR
AET (Sec.)	1.1676	1.1823	1.2062	1.4493	1.7282	5.9177	11.638	57.237	109.246	551.086	1066.78

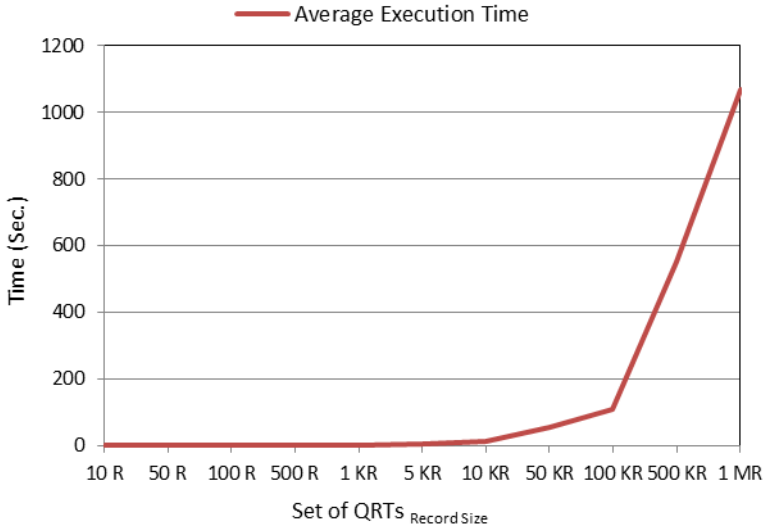

Figure 4. The AETs for Executing a Set of QRTs $_{\text{Record Size}}$ to Determine the EQMB$_f$

6. High Performance Evaluation

The parallel computing in DWS approach has two types of methods for executing QRT/EQRT. The first type of method is the query load balance. The other one is determining the adequate number of executors to reach the high performance.

6.1. Query Load Balance

The QRT/ EQRT is loaded by the owner and separated in balance to a set of executors. This method is applied to reach the high performance of AET as hypothesis needed to prove in practical environment. The parallel execution relies on the idea of saving the time of executing QRT. The experimental results of DWS approach are distributed based on the two models (DWSend and DWReceive) with different sets of QRT or EQRT $_{\text{record sizes}}$ as 5KR, 10KR, 50KR, 100KR, 500KR, 1MR.

6.1.1. Experimental Results for DWSend Model

When applying the QRT in the DWSend model with a set of record sizes in the VML middleware of DWS framework. The high performance will be reached by using parallel computing to save more time than using serial computing. We use a different set of QRT sizes distributed as blocks in balance to a set of executors (from one executor to seven executors). The AETs for QRTs load balance in DWSend model are shown in Table 2 and are represented in Figures 5.a, 5.b, 5.c, 5.d, 5.e, and 5.f. Also, the Percentage of the High Performance (%HPs) for the AETs is shown in Table 3.

Table 2. The AETs for All Experimental Results in DWSend Model

Number of Executors	QRT $_{\text{Record Sizes}}$ 5 KR	10 KR	50 KR	100 KR	500 KR	1 MR
1	5.9177	11.6375	57.2371	109.246	551.087	1066.79
2	5.2795	9.5038	48.1203	89.0884	480.876	880.784
3	5.2872	9.6643	42.5234	80.4705	424.732	833.529
4	5.286	9.686	42.7966	80.5721	398.467	810.428
5	5.2934	9.6955	42.4723	80.3939	398.820	778.946
6	5.2942	9.6942	43.0031	80.7962	398.422	778.982
7	5.3005	9.7061	43.0752	80.6631	399.339	779.000

Table 3. %HPs for All Experimental Results in DWSend Model

Number of Executors	QRT Record Sizes					
	5 KR	10 KR	50 KR	100 KR	500 KR	1 MR
1	0	0	0	0	0	0
2	10.784	18.3347	15.9281	18.4515	12.7403	17.4358
3	10.654	16.9555	25.7066	26.3401	22.9282	21.8654
4	10.675	16.7690	25.2293	26.2471	27.6942	24.0309
5	10.55	16.6874	25.7959	26.4102	27.6302	26.9820
6	10.536	16.6986	24.8685	26.0420	27.7026	26.9787
7	10.429	16.5963	24.7425	26.1638	27.5362	26.8831

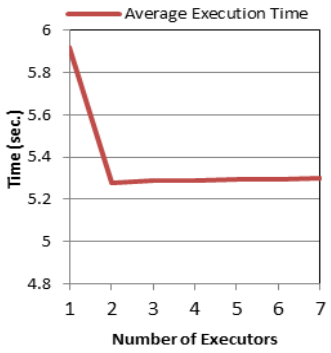

Figure 5. a The AETs for Executing QRT Record Size of 5KR on a Number of Executors

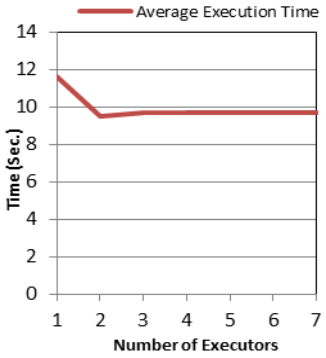

Figure 5. b The AETs for Executing QRT Record Size of 10KR on a Number of Executors

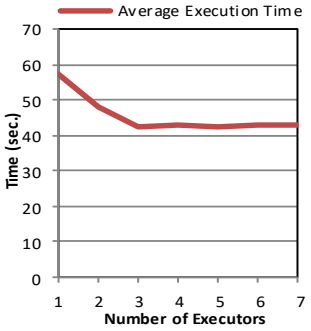

Figure 5. c The AETs for Executing QRT Record Size of 50KR on a Number of Executors

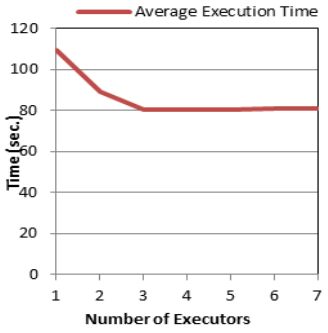

Figure 5. d The AETs for Executing QRT Record Size of 100KR on a Number of Executors

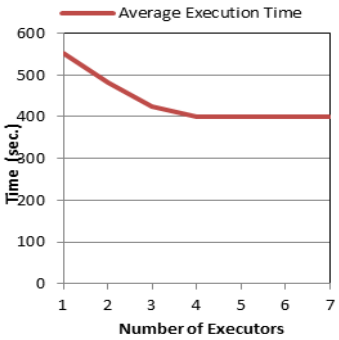

Figure 5. e The AETs for Executing QRT Record Size of 500KR on a Number of Executors

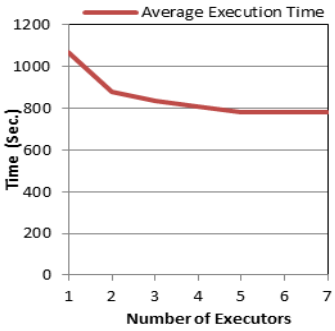

Figure 5. f The AETs for Executing QRT Record Size of 1MR on a Number of Executors

When using a parallel computing through the VML middleware of DWS framework, the result of executing the QRT of 5KR shows that the %HP for the AETs is estimated to 10% as shown in Tables 2,3 and represented in Figure 5.a. The result of executing the QRT of 50KR shows that the %HP for the AETs is estimated from 15% to 25% as shown in Tables 2,3 and represented in Figure 5.c. The result of executing the QRT of 500KR shows that %HP for the AETs is estimated from 12% to 27% as shown in Tables 2,3 and represented in Figure 5.e. The result of executing the QRT of 1MR shows that %HP for the AETs is estimated from 17% to 26% as shown in Tables 2,3 and represented in Figure 5.f.

6.1.2. Experimental Results for DWReceive Model

When the EQRT is received from DWServer, the client applies the DWReceive model that computes the hash values for the EQRT as HEBOs. This model decrypts the EBOs (Encrypted Block Objects) and compares the calculated HEBOs with the received HEBOs to display the QRT to the client. The EQRTs that are the outputs of DWSend model are used as inputs for the DWReceive model. The EQRTs are distributed in balance as blocks to the selected executors (from one executor to seven executors). This reaches the high performance by using parallel computing that saves time more than using serial computing as %HP results shown in Table 5. The AETs for the EQRTs load balance are shown in Table 4 and Figures 6.a, 6.b, 6.c, 6.d, 6.e, and 6.f.

Table 4. The AETs for All Experimental Results in DWReceive Model

Number of Executors	EQRT Record Sizes					
	5 KR	10 KR	50 KR	100 KR	500 KR	1 MR
1	9.924	19.3588	95.1705	180.937	916.413	1656.296
2	9.2433	16.5825	77.8879	142.126	752.32	1302.880
3	9.2509	16.2599	72.4065	136.595	698.727	1289.103
4	9.2541	16.2609	72.7736	131.086	685.772	1259.840
5	9.2591	16.2617	72.5459	132.514	655.109	1232.589
6	9.2534	16.2631	72.9051	132.908	655.987	1167.304
7	9.2558	16.2616	72.9422	132.216	656.044	1167.521

Table 5. %HPs for All Experimental Results in DWReceive Model

Number of Executors	EQRT Record Sizes					
	5 KR	10 KR	50 KR	100 KR	500 KR	1 MR
1	0	0	0	0	0	0
2	6.8591	14.3412	18.1596	21.4501	17.9060	21.3377
3	6.78254	16.0077	23.9191	24.5067	23.7541	22.1695
4	6.7503	16.0025	23.5334	27.5516	25.1678	23.9362
5	6.69991	15.9984	23.7727	26.7624	28.5138	25.5815
6	6.75735	15.9911	23.3952	26.5445	28.4179	29.5232
7	6.73317	15.9989	23.3562	26.9268	28.4117	29.5101

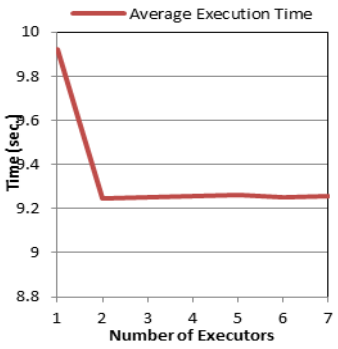

Figure 6.a The AETs for Executing $EQRT_{Record\ Size}$ of 5KR on a Number of Executors

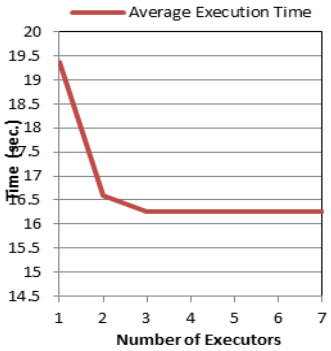

Figure 6. b The AETs for Executing $EQRT_{Record\ Size}$ of 10KR on a Number of Executors

Figure 6. c The AETs for Executing $EQRT_{Record\ Size}$ of 50KR on a Number of Executors

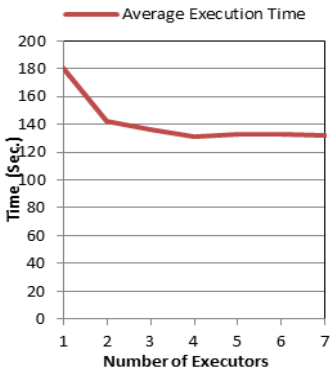

Figure 6. d The AETs for Executing EQRT Record Size of 100KR on a Number of Executors

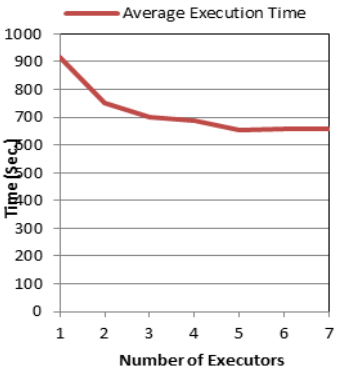

Figure 6. e The AETs for Executing EQRT Record Size of 500KR on a Number of Executors

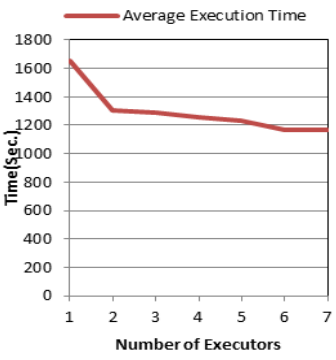

Figure 6. f The AETs for Executing EQRT Record Size of 1MR on a Number of Executors

The result of executing the EQRT of 5KR shows that the %HP for the AETs is estimated to 6% when using parallel computing in the VML middleware as shown in Tables 4,5 and represented in Figure 6.a. The result of executing the EQRT of 50KR shows that the %HP for the AETs is estimated from 18% to 23% as shown in Tables 4,5 and represented in Figure 6.c. The result of executing the EQRT of 500KR shows that %HP for AETs is estimated from 17% to 28% as shown in Tables 4,5 and represented in Figure 6.e. The result of executing the EQRT of 1MR shows that %HP for the AETs is estimated from 21% to 29% as shown in Tables 4,5 and represented in Figure 6.f.

6.2. Determining the Maximum Number of Executors for High Performance

The high performance time is the main idea to achieve availability in a DW system by eliminating overhead and query response timed-out. In DWS framework, the VML middleware is used to reach the high performance by using parallel computing. The small unit for parallel execution is a thread. This thread is scheduled as a job for an executor. The main method for scheduling threads is First-Come-First Served (FCFS) as ordered from the owner to the manager and from the manager to the executors [20].

In DWS framework, there are two factors, which are the transmission time and the collection/separation time. These factors lead to two challenges through choosing the number of records as block object. First, dividing the QRT to small blocks takes a lot of time in the collection, although the transmission time for these blocks is a small. Second, dividing the QRT to large blocks takes a little time in the collection, although the transmission time is a lot. Therefore, the load balance for the number of executors solves these challenges of time factors.

In the DWSend model, the AETs are decreased when using parallel computing. This decrease is not continuous because of the cost of using parallel computing. The cost of using parallel computing is defined in DWS approach by two factors resisted to reach the high performance completely. As the number of executors increases, the AET taken to execute the QRT does not decrease. Where the AET of 500KR is equal to 398.467 Sec. when using four executors, but when the number of executors increases to five executors, the AET is equal to 398.820 Sec., as shown in Table 2 and Figure 5.e. Thus, the high performance has a limit to the used number of executors.

In DWReceive model, the increasing number of executors is the way to reach the high performance in the AET as a first idea. However, the performance doesn't reach to a complete save of time in the parallel computing. As shown in the results in Table 4 and Figure 6.e, the AET for the EQRT of 500KR is equal to 655.109 Sec. when using five

executors, but when the number of executors increases to six, the AET is equal to 655.987 Sec. In this case, the increasing number of executors does not change in the AET; hence, the high performance has a limit.

The maximum number of executors is limited with the high performance in the AETs by determining two AETs: the AET_{Exe1} is the average of 10 execution times for loading the QRT/EQRT as one block object of records executed on one executor (serial computing). The AET_{Exe2} is the average of 10 execution times for loading the QRT/EQRT as two block objects of records distributed to two executors (parallel computing). The Maximum Query Memory Buffer ($MaxQMB_f$) is calculated by dividing the $QRT/EQRT_{Record\ Size}$ on the Query Memory Buffer $_{Record\ Size}$ ($QMB_{f\ Record\ Size}$), as shown in Equation (1).

$$\mathrm{MaxQMB}_f = \left\lceil \frac{QRT\,or\,EQRT_{Record\ Size}}{QMB_{f\ Record\ Size}} \right\rceil \quad (1)$$

The $_{Record\ Size}$ is the basic unit used in the DWS framework, because the QRT/EQRT is a number of records of data.

The needed number of executors is the ratio between executing the QRT in parallel computing by using the VML middleware and executing the QRT on a single machine in serial computing. Only the number that is enough used to reach the high performance is the Maximum Number of Executors for DWSend model ($MaxNoEs_{DWSend}$) computed in the Equation (2).

$$\mathrm{MaxNoEs}_{DWSend} = \frac{AET_{Exe1} + ((\frac{QRT_{RecordSize}}{MaxQMB_f * EQMB_f}) * (AET_{Exe1} - AET_{Exe2}))}{AET_{Exe1}} \quad (2)$$

For example, execute a QRT with size (120 KR "120,000 records ", 24.621 MB) as an experimental study in the VML middleware of the DWS framework. We can summarize it to find the $MaxNoEs_{DWSend}$ through applying the Equation (2) as presented below:

1. Calculate the $EQMB_f = \frac{QRT_{Record\ Size}}{QRT_{MB\ Size}}$
$= \frac{120\ KR}{24.621}$
= 4874 Records per 1MB
2. Calculate the $QMB_{f\ Record\ size} = QMB_{f\ MB\ Size} * EQMB_f$
= 20 * 4874
= 97480 Records
3. Calculate the $MaxQMB_f$ by using the Equation (1).
$= \left\lceil \frac{120\ KR}{97480} \right\rceil$
$= \lceil 1.23 \rceil$
= 2 QMB_f s

4. Measure the AET on single machine and measure the AET on two executors joined in the VML middleware for the DWS framework.

AET_{Exe1} = 131.429 Sec. , AET_{Exe2} = 105.124 Sec.

5. Apply the Equation (2) to find the $MaxNoEs_{DWSend}$ as follows:

$$\mathrm{MaxNoEs}_{DWSend} = \left\lceil \frac{131.429 + ((\frac{120\ KR}{2 * 4874}) * (131.429 - 105.124))}{131.429} \right\rceil$$
$$= \lceil 3.463842 \rceil$$
= 4 Executors

To prove the theoretical concept and to reach the high performance by finding the MaxNoEs in the DWSend model as practical results, we apply the Equation (2) on the QRT sizes used in the experimental studies (see Section 6.1). We note that the $EQMB_f$ is equal to 4822 records and the QMB_f is equal to 96440 records in the VML middleware. The results are shown in Table 6.

Table 6. The Maximum Number of Executors for Experimental Studies in Dwsend Model

QRT Sizes (Record)	$MaxQMB_f$	AET_{Exe1} (Sec.)	AET_{Exe2} (Sec.)	MaxNoEs (Executor)	
5KR	1	5.9177	5.2795	1.111826995	≈ 2
10KR	1	11.6375	9.5038	1.380230068	≈ 2
50KR	1	57.2371	48.1203	2.651610383	≈ 3
100KR	2	109.246	89.0884	2.913269185	≈ 3
500KR	6	551.087	480.8769	3.201762312	≈ 4
1MR	11	1066.79	880.7844	4.287174494	≈ 5

As the DWSend model, the MaxNoEs in the DWReceive model is the ratio between executing the EQRT in parallel through the VML middleware at the client and executing the EQRT in serial on a single machine. The MaxNoEs $_{DWReceive}$ is computed by using the Equation (3) in order to reach the high performance in the DWReceive model for DWS framework as practical results. The MaxNoEs for each experimental study in the DWReceive model is shown in Table 7.

$$\text{MaxNoEs}_{\text{DWReceive}} = \left\lceil \frac{AET_{Exe1} + \left(\left(\frac{EQRT_{Record\ Size}}{MaxQMB_f * EQMB_f}\right) * (AET_{Exe1} - AET_{Exe2})\right)}{AET_{Exe1}} \right\rceil \quad (3)$$

Table 7. The Maximum Number of Executors for Experimental Studies In Dwreceive Model

EQRT Sizes (Record)	$MaxQMB_f$	AET_{Exe1} (Sec.)	AET_{Exe2} (Sec.)	MaxNoEs (Executor)	
5KR	1	9.924	9.2433	1.07112328	≈ 2
10KR	1	19.3588	16.5825	1.29741355	≈ 2
50KR	1	95.1705	77.8879	2.88299655	≈ 3
100KR	2	180.937	142.126	3.22419388	≈ 4
500KR	6	916.413	752.32	4.09451303	≈ 5
1MR	11	1656.3	1302.88	5.02279692	≈ 6

6.2.1. Mathematical Model Proof

This paper presents the mathematical model proof for the two DWS models shown in Equations (2) and (3). Depending on the grid signature approach, the time complexity analysis shows that the execution time function O (T) is equal to one over number of executors when increasing the number of executors [21] as presented below:

Assume N is the number of executors.

$$O(T) = \frac{1}{N}$$

In the DWS framework, the maximum number of executors that reaches high performance is the ratio between executing the QRT/EQRT in parallel computing, Parallel Time ($T_{Parallel}$), and executing the QRT/EQRT in serial computing, Serial Time (T_{Serial}). T(N) is equal to one over number of executors $\frac{1}{N}$ based on the previous math proof. By substitution, we find the $T_{Parallel}$ for T_{Serial}, as shown in Equation (4):

$$T_{Serial}(N) = \frac{1}{N} T_{Parallel}$$

$$T_{Parallel} = N * T_{Serial}$$

$T_{Serial} = AET_{Exe1}$, the AET_{Exe1} is acquired when using one executor in serial computing.

$$\text{Then, } T_{Parallel} = N * AET_{Exe1} \quad (4)$$

The speed up in GridCryptoGraphy approach is defined as the ratio for the load in one executor with different executors in parallel [22]. Based on the analysis of the result graphs shown in figures (see Section 6.1), we find the math function $\boldsymbol{f}(x)$ to get the number of executors for the high performance in the practical experiments. There is an exponential growth relation between the increase of the number of executors and the high performance. Also, there is an exponential growth relation between the QRT size and the high performance. So, the math function becomes as follows:

$$\boldsymbol{f}(x) = \boldsymbol{x} + \boldsymbol{adx} \quad (5)$$

By using the modelization, we substitute the math function value $\boldsymbol{f}(x)$ in Equation (5) with the maximum number of executors given from the mathematical proof in Equation (4). The unit of execution function is distributed in one executor at minimum, so, we substitute the x value with the AET_{Exe1}. Because the QRT size is related to the constant number of executors, we substitute the a value with the number of blocks passed through the VML middleware. Thus, the a value is equal to $\frac{QRT_{Record\ Size}}{MaxQMB_f * EQMB_f}$.

The change of the AET in the middleware is reached from the serial execution to the maximum transfer process in the parallel execution as AET_{Exe2}, because it is the maximum time acquired when sending the data of the QRT as two blocks in parallel($\frac{1}{2} > \frac{1}{3} > \frac{1}{4} \ldots > \frac{1}{n}$). Then, this change creates the time difference between using one block in serial and using more blocks in parallel. So, we substitute dx with (AET_{Exe1} - AET_{Exe2}) to make the sides of Equations (4) and (5) as:

$$N * AET_{Exe1} = x + a\,dx$$

$$N * AET_{Exe1} = AET_{Exe1} + \left(\left(\frac{QRT\ or\ EQRT_{Record\ Size}}{MaxQMB_f * EQMB_f}\right) * (AET_{Exe1} - AET_{Exe2})\right)$$

$$N = \frac{AET_{Exe1} + \left(\left(\frac{QRT\ or\ EQRT_{Record\ Size}}{MaxQMB_f * EQMB_f}\right) * (AET_{Exe1} - AET_{Exe2})\right)}{AET_{Exe1}}$$

Then, the mathematical model for the MaxNoEs (DWSend or DWReceive) in the DWS is equal to the found N based on the experimental results graphs and the mathematical proof for the time complexity analysis.

6.2.2. Special Cases

To prove the stabling of Equations (2), (3) in a mathematical formula, we test the change in equation parameters as presented in two cases:

- **Case 1:**

If there is no difference in time between AET_{Exe1} and AET_{Exe2} (*i.e.*, the value of AET_{Exe2} is equal to AET_{Exe1}), as shown below:

$$MaxNoEs_{DWSend\ or\ DWReceive} = \left\lceil \frac{AET_{Exe1} + \left(\left(\frac{QRT\ or\ EQRT_{Record\ Size}}{MaxQMB_f * EQMB_f}\right) * (AET_{Exe1} - AET_{Exe2})\right)}{AET_{Exe1}} \right\rceil$$

$$= \left\lceil \frac{AET_{Exe1} + \left(\left(\frac{QRT\ or\ EQRT_{Record\ Size}}{MaxQMB_f * EQMB_f}\right) * (0)\right)}{AET_{Exe1}} \right\rceil$$

$$= \left\lceil \frac{AET_{Exe1}}{AET_{Exe1}} \right\rceil$$

= 1 executor as maximum

- **Case 2:**

If the QRT $_{\text{Record Size}}$ is less than the EQMB$_f$, (*i.e*. If (QRT $_{\text{Record Size}}$ < EQMB$_f$)

$$\text{MaxNoEs}_{\ DWSend\ or\ DWReceive} = \left\lceil \frac{AET_{Exe1} + \left(\left(\frac{QRT\ or\ EQRT_{Record\ Size}}{MaxQMB_f\ *\ EQMB_f} \right) * (AET_{Exe1} - AET_{Exe2}) \right)}{AET_{Exe1}} \right\rceil$$

$$= \left\lceil \frac{AET_{Exe1}}{AET_{Exe1}} + \frac{\left(\left(\frac{QRT\ or\ EQRT_{Record\ Size}}{MaxQMB_f\ *\ EQMB_f} \right) * (AET_{Exe1} - AET_{Exe2}) \right)}{AET_{Exe1}} \right\rceil$$

Then QRT$_{\text{Record Size}}$ < QMB$_f$, and MaxQMB$_f$ =1.

$$= \left\lceil 1 + \frac{\left(\left(\frac{QRT\ or\ EQRT_{Record\ Size}}{1\ *\ EQMB_f} \right) * (AET_{Exe1} - AET_{Exe2}) \right)}{AET_{Exe1}} \right\rceil$$

Where AET $_{\text{Exe2}}$ < AET $_{\text{Exe1}}$

$$= \left\lceil 1 + \frac{\left(\left(\frac{QRT\ or\ EQRT_{Record\ Size}}{EQMB_f} \right) * (AET_{Exe1} - AET_{Exe2}) \right)}{AET_{Exe1}} \right\rceil$$

Where $\frac{QRT\ or\ EQRT_{Record\ Size}}{EQMB_f} < 1$, and $\frac{AET_{Exe1} - AET_{Exe2}}{AET_{Exe1}} < 1$

$$= \lceil 1 + (< 1) \rceil$$

= 2 executors as maximum

In general, when the organization applies the DWS framework and needs to reach the high performance in the AET, it should choose the maximum of the MaxNoEs computed in the two models (MaxNoEs $_{\text{DWSend}}$ and MaxNoEs $_{\text{DWReceive}}$). As shown in Table 8, the MaxNoEs for the DWS framework is found by using Equation (6).

$$\text{MaxNoEs}_{\text{DWS}} = max(MaxNoEs_{DWSend}, MaxNoEs_{DWReceive}) \quad (6)$$

Table 8. The Maximum Number of Executors for the DWS Framework

QRT/EQRT Sizes(Record)	MaxNoEs $_{\text{DWSend}}$	MaxNoEs $_{\text{DWReceive}}$	MaxNoEs $_{\text{DWS}}$
5KR	2	2	2
10KR	2	2	2
50KR	3	3	3
100KR	3	4	4
500KR	4	5	5
1MR	5	6	6

7. Conclusion and Future Work

In distributed systems, the harnessing of the CPU of PCs and the data sources are the main priority to be created in the DW field. The data of DW contains financial and business information, such as credit and debit cards data in banking or e-commerce systems. So, achieving all security issues CIA is very important for DW organizations. One of the security issues is the availability that enables services to clients from the DWServer at any request. This paper has presented two sets of experimental studies distributed into the two models (DWSend and DWReceive). First, determining the edge of query memory buffer for the VML Middleware to use it for measuring performance as a theoretical calculation. The other experiments are to evaluate the high performance that

is divided into two parts: the Query load balance execution and the maximum number of executors for the high performance. The DWS framework has reached the high performance by using the query load balance. The QRT has been separated as blocks in balance and then distributed to a set of executors to be executed in parallel. The performance of the AET has increased with the increase of the number of executors to limit value, where the AET value becomes a constant as proved in the experimental studies. Based on the performance evaluation of the experimental studies and its mathematical analysis, this paper has presented a mathematical model for the DWS framework. The mathematical model has been built to detect the maximum number of executors for using the DWS framework. This model allows DW organizations to reach the high performance and to resist the overhead of query response time as a server services in network when executing the QRT/EQRT in parallel.

In future work, the DWS approach will be evaluated in different threat models to modify the authentication process in the VML middleware with establishing a group key to increase security in DW systems. In addition, we will experiment the DWS framework in other middleware tools such as Globus, SETI@Home, XtermWeb to be compared with the current experimental results used the VML middleware. Finally, we will apply the VML middleware in decision analysis processes in DWs to obtain the performance in execution time of decision-making by using parallel computing.

References

[1] M. A. Meghari, "Survey on Security Issues Techniques Used in Data Warehouses", International Journal of Computer Science and Network Security (IJCSNS), vol. 17, no. 3, **(2017)**, pp. 236-243.
[2] M. S. Reddy, M. R. Reddy, R. Viswanath, G. V. Chalam, R. Laxmi, and Md. A. Rizwan, "A Schematic Technique Using Data type Preserving Encryption to Boost Data Warehouse Security", International Journal of Computer Science Issues(IJCSI), Vol. 8, Issue 1, **(2011)**, pp. 460-465.
[3] T. Ge and S. Zdonik, "Fast, Secure Encryption for Indexing in a Column-Oriented DBMS", Proceedings of IEEE 23rd International Conference on Data Engineering (ICDE), Istanbul, **(2007)**, pp. 676-685.
[4] H. Kadhem, T. Amagasa and H. Kitagawa, "A Novel Framework for Database Security based on Mixed Cryptography", Proceedings of IEEE 4th International Conference on Internet and Web Applications and Services, Venice/Mestre, **(2009)**, pp. 163-170.
[5] A. Deshmukh and R. Qureshi, "Transparent Data Encryption- Solution for Security of Database Contents", International Journal of Advanced Computer Science and Applications (IJACSA), vol. 2, no. 3, **(2011)**, pp. 25-28.
[6] V. Attasena, N. Harbi. and J. Darmont, "A Novel Multi-Secret Sharing Approach for Secure Data Warehousing and On-Line Analysis Processing in The Cloud", International Journal of Data Warehousing and Mining (IJDWM), vol. 11, no. 2, **(2015)**, pp. 22-43.
[7] R. J. Santos, D. Rasteiro, J. Bernardino and M. Vieira, "A Specific Encryption Solution for Data Warehouses", Proceedings of International Conference on Database Systems for Advanced Applications, Springer Berlin Heidelberg, pp. 84-98, **(2013)**.
[8] R. J. Santos, J. Bernardino and M. Vieira, "A data masking Technique for Data Warehouses," Proceedings of the 15th International Database Engineering and Applications Symposium (IDEAS'11), ACM, pp. 61-69, **(2011)**.
[9] R. Chowdhury, P. Chatterjee, P. Mitra and O. Roy, "Design and Implementation of Security Mechanism for Data Warehouse Performance Enhancement Using Two Tier User Authentication Techniques", International Journal of Innovative Research in Science, Engineering and Technology, vol. 3, no. 6, **(2014)**, pp. 165-172.
[10] Preeti and K. Khatka, "Enhancing Data Security And Privacy On WebOS Using TSFS," International Journal of Computer Engineering And Electronics Technology (IJCEET), vol. 1, no. 1, **(2015)**, pp. 1-5.
[11] M. AlMeghari, "Data Warehouse Signature: A Framework for Implementing Security Issues in Data Warehouses", Journal of Computer Sciences and Applications, Science and Education Publishing(SciEP), DOI: 10.12691/jcsa-5-1-3, vol. 5, no. 1, **(2017)**, pp. 17-24.
[12] A. Luther, R. Buyya, R. Ranjan and S. Venugopal, "Alchemi: A .NET-based Enterprise Grid Computing System", Internet Computing, cloudbus.cis.unimelb.edu.au, **(2005)**, pp. 5-9.
[13] R. Bhanot, and R. Hans, "A Review and Comparative Analysis of Various Encryption Algorithms", International Journal of Security and Its Applications (IJSIA), SERSC, vol. 9, no. 4, **(2015)**, pp. 289-306.
[14] A. Kak, "Lecture 12: Public-Key Cryptography and the RSA Algorithm", Lecture Notes on Computer and Network Security, Purdue University, **(2015)**.

[15] Federal Information Processing Standards-FIPS Publication 197, "Advanced Encryption Standard (AES)," **(2001)**.
[16] W. Stallings, "Cryptography and network Security Principles and Practice", Sixth Edition, Pearson Education, Inc, Printed in USA, ISBN 13: 978-0-13-335469-0, **(2014)**.
[17] Federal Information Processing Standards-FIPS Publication 180-4, "Secure Hash Standard (SHS)," **(2012)**.
[18] A. Kak, "Lecture 15: Hashing for Message Authentication," Lecture Notes on Computer and Network Security, Purdue University, **(2015)**.
[19] R. Griffith, G. Valetto and G. E. Kaiser, "Effecting Runtime Reconfiguration in Managed Execution Environments", CRC Book, IBM Thomas J. Watson Research Center, Columbia University, **(2005)**.
[20] M. Haque, "An Evaluation of the State of Affairs of Grid Computing: Current and Future Projections," Master of Science in Engineering and Management, Massachusetts Institute of Technology, **(2005)**.
[21] A. Hegazy, B. Hasan and I. Shaheen, "Grid Signature: High Performance Digital Signature Through Using Alchemi Grid Computing", Canadian Center of Science and Education (CCSE), Journal of Computer and Information Science, vol. 3, no. 3, **(2010)**, pp. 56-65.
[22] M. Awadallah and A. Youssef, "Scalable Symmetric Key Cryptography Using Asynchronous Data Exchange Enterprise Grid", International Journal of Computer Science Issues (IJCSI), vol. 8, no. 3, **(2011)**, pp. 107-115.

Authors

Mayada J. ALMeghari, Ph.D student of Information Systems at Faculty of Computers & Information, Cairo University, Egypt. She received M.Sc. degree of Computer Information Systems from the Faculty of Information Systems & Technology at The Arab Academy for Banking & Financial Sciences University, Jordan 2006. She is now an academic lecturer in Computer department at Palestine Technical College-Deir El-Balah, Gaza, Palestine. Her research interest includes information security, software engineering, data warehouses and distributed systems.

