Asian Journal of Information Technology 15 (11): 1691-1705, 2016
ISSN: 1682-3915
© Medwell Journals, 2016

# An Enhanced Query Optimization Approach for Cloud Data Management

Eman A. Maghawry, Rasha M. Ismail, Nagwa L. Badr and M.F. Tolba
Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt

**Abstract:** Cloud computing is a promising computing model that provides a combination of parallel and distributed computing paradigms. It has the characteristics of on demand provisioning of a shared pool of configurable computing resources as a service. It provides a cost effective paradigm of computational, storage and database resources to users over the internet. Cloud storage is an important service that is provided by the cloud system, it provides the data owners with high accessibility, availability and scalability in respect to increasing the amount of their data in cloud repositories. The increasing number of users query data from deployed virtual instances can lead to increased loads on the cloud data management system. Multiple queries compete for hardware resources causing resources contention within a rapidly changing in environment computational properties. So efficient concurrent queries execution on especially structured data in such environment has become an important challenge.

**Key words:** Query optimization, query processing, cloud computing, cloud storage, distributed resources

## INTRODUCTION

Cloud computing is becoming an emerging powerful model for hosting computing services. These services are delivered to clients over the internet with their different expectations on the quality of the service (Maghawry *et al*., 2014a). Cloud computing unifies computing components to provide software, platforms and infrastructure as a service. Software-as-a-Service (SaaS) is considered the top level component that presents the model of deploying applications to end users on demand. The next level is Platform-as-a-Service (PaaS) that provides development tools to build applications based on the service provider's resources. On the lowest level, Infrastructure as a Service (IaaS) offers resources such as processing power or storage to the end users. These services are offered and maintained by various cloud computing providers over the internet.

Service providers offer on-demand resources provisioning to the clients in a pay-only-for-what-you-use pricing model (Kossmann and Kraska, 2010). Some of these major cloud service providers are Amazon (Amazon Web Services http://aws.amazon.com/), Google Apps (Google Apps:www.google.com/Apps/Work), Microsoft (Microsoft Azure: http://azure.microsoft.com/en-us/) and Sales-force (Sales-force:http://www.salesforce.com/). They prevent the clients from operational costs such as purchasing, maintaining hardware and set up costs. Also cloud systems offer Service Level Agreements (SLA) that describe the Quality of Service (QoS) and service pricing to cloud's users. In addition, it applies penalties to the service provider in the case of user agreement violations (Brandic and Dustdar, 2011). Cloud data storage is one of the services offered by the cloud computing providers. As the continuous data is growing, cloud provider enables the remote clients to store their data to the cloud storage environment by hosting their data on the cloud resources instead of their own servers, these resources are virtual machines with previously installed and configured database systems. A query processing on data stored within the cloud clusters is considered the major challenge over cloud data storage service. Therefore cloud storage has resulted in an increasing demand to handle the high amount of concurrent queries submitted by users that accessing the resources (Maghawry *et al*., 2014b).

Most of the commercial cloud providers are supported by collections of physical instances within distributed data centers over large geographical regions. New challenging research issues are raised with such environment controlling these cloud resources efficiently offer services to the clients. As cloud computing grows in popularity, querying distributed data sources becomes a challenge because a data source is needed to handle any failures and deliver high availability. Cloud users and providers are interested in achieving performance objectives such as; minimizing requests response time and maximizing the utilization of cloud resources. In this study, a collection of techniques are proposed in an integrated methodology in order to enhance the performance of query processing over cloud resources.

**Corresponding Author:** Eman A. Maghawry, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt

Cloud environments consist of heterogeneous commodity resources and they process workloads and tasks in parallel. When a client submits a query, master nodes dispatch the query into worker nodes for concurrent queries processing and the collected results are returned from the distributed nodes. The queries execution may cause low performance because queries are executed under rapidly changing computational properties in the cloud environment such as; loads. Some of the nodes may execute faster while some may be slower because of the system nodes contention. This node contention can slow the query response time and effect on the system performance. Therefore, heterogeneous resources can result in a load imbalance during execution, therefore load balancing technique is essentially required to handle the high amount of concurrent queries that access the resource (Maghawry *et al*., 2014a, b). So an optimized query processing technique is required with taking into consideration continuous monitor assessment and fast response according to the progress of resources execution.

Previous researches on query processing over a cloud focused only on issues on query optimization, query resource allocation or query scheduling topics. Therefore in this study, we focus on presenting integrated techniques for optimizing, scheduling and allocating queries in addition to load management techniques that are presented and implemented in an efficient arrangement for enhancing the overall query processing performance. The approach enhances the overall performance of query executions over cloud environments. The main aim of the proposed approach is minimizing the response time and maximizing the utilization of cloud resources. The presented approach is evaluated in terms of the query processing performance. This study improves the query processing efficiency by contributing with integrating the following techniques:

- Query optimization techniques to exploit the shared data among the submitted queries through optimizing and merging the related queries to improve query processing efficiency over cloud resources. It also considers different delay times of submitted queries in order to apply the queries merging step in case of a positive impact on the query execution performance
- Scheduling technique to determine the efficient ordering of the queries execution to reduce their response time. It also determines the scheduling decisions by taking into account saving the waiting time of the queries within the execution queue
- Assigning and allocating the queries across the cloud virtual instances in an efficient manner
- Workload Management technique to recover the failure or imbalance that may occur during the queries execution. This is achieved by exploiting the database replicas to enhance the execution performance and handling instances contention.

This study focuses on presenting the following main contributions:

- Enhancing the query processing performance over the cloud by combining the previous presented techniques (query optimizing, scheduling, allocating and workload management) in a working architecture
- Using a real world cloud (Amazon EC2 infrastructure provisioning service) instead of the simulated environment that's used in our previous research and in most other researches
- Different query types and machine capabilities were used to evaluate the proposed architecture. Examples of the used measurements are the query response time, the query throughput and query optimization time

**Literature review:** Previous researches on query processing have focused on issues on only query scheduling, query resource allocation or query optimization. Unfortunately, most of the research in this area has focused on processing a single query and does not consider multi-users and concurrent queries with multi-resource issues such as loads. Although, previous research addresses several issues in queries processing, our proposed architecture combines the query optimization and query resource allocation techniques with monitoring the concurrent queries execution over the running resources. Furthermore, it responds to any load imbalance by applying the workload management technique on the cloud environment. In the query optimization research area, a Merge-Partition (MP) query reconstruction algorithm was presented by Chen *et al*. (2011).

Their algorithm is able to exploit data sharing within the submitted concurrent sub-queries to reduce the average communication overheads. Their research is related to the IGNITE system that was proposed by Lee *et al*. (2007) which was developed based on the PostgreSQL database (PostgreSQL homepage: http://www.postgresql.org). Also, Liu and Karimi (2008) they proposed a resource selection module to select the appropriate sub-set of resources to execute the query by applying a ranking function on the available resources to improve the query execution performance and optimization time. In the mentioned techniques

(Chen *et al*., 2011) they focused only on exploiting and optimizing the shared data among the submitted queries. On the other hand, Liu and Karimi (2008) they focused on scheduling and assigning the queries to suitable resources. our previous research by Maghawry *et al.* (2012) combined and enhanced the previous techniques by Liu and Karimi (2008) and Chen *et al.* (2011) to optimize the shared data among the queries then assign the queries to suitable resources based on a ranking function to improve the query processing performance.

On the other hand, many approaches discussed scheduling techniques some of them relied on machine learning and prediction techniques to estimate the performance metrics of database queries before starting the execution as proposed by Luo *et al.* (2006), Gupta *et al.* (2008), Ganapathi *et al.* (2009), Akdere *et al.* (2012) and Li *et al.* (2012). Their goal was to create an accurate prediction tool to predict the performance of new queries for making scheduling decisions. In addition, the Contender framework was presented by Duggan *et al.* (2014) for concurrent query performance prediction; they used an integration of empirical evaluation and semantic information to create models for the query template. These models describe the resource contention for each query template at different concurrency levels. Query interaction was also considered by Ahmad *et al.* (2011a) and Sheikh *et al.* (2011), they presented an approach for calculating workload completion times by taking into consideration the samples of the space of possible query mixes by choosing the most suitable models, they then built performance models by observing the performance of these samples.

A Service Level Agreement (SLA) tree which is a data structure was proposed by Jarke *et al.* (2014). Authors constructed the SLA tree for each query to combine a sequence of buffered submitted queries together with their SLA requirements. They built this tree to support profit oriented decisions in many components within cloud systems such as scheduling. The main disadvantage of machine learning techniques is the accuracy of the query running times prediction which is a difficult problem because it is a complex function of the query itself including any run-time parameters and dynamic factors. Therefore, this is not practical in most real world database applications. Recently, there have been many efforts in research areas (Ganapathi *et al*., 2009; Chi *et al*., 2011) to reduce the errors of the query running times prediction (Tozer *et al*., 2010). The Shepherd technique stands for (scheduling under probabilistic histogram based query time distributions scheduling technique) was proposed by Maghawry *et al.* (2014). It used the probability distributions of query execution times for scheduling queries in the presence of SLA requirements. Therefore the Shepherd scheduler technique (Chi *et al*., 2013) was preferred to be used in our proposed system because it considers all possible values of the queries execution time rather than relying on just a single point of execution time estimation. On the other hand, Shepherd didn't consider the related queries running in the system. Therefore, in our previous research (Maghawry *et al*., 2014a, b) an enhancement on the Shepherd technique was proposed by considering the related queries running in the system which depends on how much I\O and CPU are needed to finish the queries execution.

In the field of workload management, several techniques have been proposed to achieve a workload balance during the tasks execution. Many approaches (Krompass *et al*., 2006; Schroeder *et al*., 2006) were introduced workload management for managing resource allocation for database queries to handle different resource requirements and capabilities in workload environments. Also the research by Paton *et al.* (2009a) proposed several techniques for dynamically re-distributing assignments of processor loads with consideration to the varying resource capabilities.

These techniques were designed for using in unpredictable environments conditions; they proposed an adaptive load balancing approach depending on incremental replication of a query operator state. Furthermore, the workload manager presented by Subramanian *et al.* (2000) used feedback control to adjust and govern the resources execution. It receives information about data workload performance from performance monitoring that's created by the application provider then their workload manager uses a controller to specify the suitable resources for allocating a workload, on the other hand some techniques consider the impact of concurrency and query properties on managing the workload. Their technique focuses on the query behavior analysis so they proposed sampling techniques in order to predict resource contention as Liu and Karimi (2008) and Duggan *et al.* (2011). The authors as Avnur and Hellerstein (2000) and Tian and DeWitt (2003) continually monitor the speed of query operators and use this information to modify the query plan by improving the query performance. Furthermore, they used a queuing network to describe performance metrics for response time and system throughputs for a distributed stream management system to control workload imbalance. Moreover, many approaches rely on utility functions such as the researchess by Paton *et al.* (2009, 2012) they use utility functions integrated with optimization algorithms in order to maximize utilities for a given workload for certain resources. For instance, they present an autonomic workload mapper that adaptively assigns tasks to execution sites.

Furthermore, they check the resources assignment by revising the feedback of the submitted requests progress during the workload executions. The goal of their approach is to check the alternative mapping space to achieve the maximum utilities by applying and computing the utility function. Other techniques proposed a workload management system which includes a dynamic execution organizer that leverages fuzzy logic as Krompass *et al*. (2007) also other researches suggest machine learning techniques that use monitored data to sample the system. They have developed a query scheduler that considers the query interaction within workloads as in the research that was presented by Ahmad *et al.* (2011b).

As presented in previous workload management topic researches, there are different techniques for handling the load during queries execution though focusing on one technique such as machine learning to predict the resource contention, analyzing the query behavior or revising the feedback on the resources assignments or exploiting the replicas in re-distributing the load. In our previous research (Maghawry *et al*., 2012), many features on workload management are presented to recover the resources load such as checking the resources contention, revising feedback loop about resources status after allocating the resources and exploiting the replicas to handle any load that may occur during the execution.

A SQL query processing algorithm called (ESQP stands for Efficient SQL Query Processing) is proposed by Zhao *et al*. (2010). They used data replicas in cloud storage for processing the SQL queries and they presented global and random scheduling in query procedures to reduce the response time for each query. In this study, the proposed query processing approach was compared against the ESQP approach which was proposed by Zhao *et al.* (2010). However ESQP, relied on just their scheduling technique to get the load balance and didn't revising the resources feedback loop after resource allocation process.

To overcome the shortage of previous researchess, this study presents an enhancement to the query processing performance over cloud environment. It proposes an architecture that illustrates the main techniques of our research. The next section explains this proposed architecture and its components.

## MATERIALS AND METHODS

Cloud computing platforms contain many of the heterogeneous hardware that are responsible for data storage. As a result of the popularity of traditional Relational Database Management System (RDBMS), the used data by the majority of enterprisers deal in decision support and business planning is structured data. However, most cloud platforms do not support SQL queries because they are not designed for structured data management (Zhao *et al*., 2010). On the other hand some platforms support SQL queries that based on traditional relational databases. For cloud deployment, the data and workload characteristics of typical data management applications are well suited within it (Zhao *et al*., 2010).

There are two types of nodes in the proposed system: Master nodes and worker nodes. Master nodes store performance and meta data about the total worker nodes (such as RAM amount, CPU speed) while worker nodes store the data records and their replicas. The proposed architecture for query processing over cloud environments is represented in Fig. 1. The key innovation of the proposed architecture is the way of arranging the processes of the combined modules in an optimized manner which highly improves the queries processing performance over cloud environments. The main contribution of the proposed architecture is to minimize the queries response time and maximize the utilization of cloud resources.

When the master node receives the clients' queries, the optimizer module exploits the shared requested data among the submitted queries. Then the queries are queued by the scheduler to be ready to start distributing them on the available and most suitable worker instances. During the queries execution, the workload manager module monitors, diagnoses and recovers any failure or load imbalance that may occur during the execution through exploiting the existing replicas. Finally, results are collected from the distributed virtualized resources and returned to the clients. The proposed architecture involves three key components of the query processing.

**Query optimization module:** This module is responsible for optimizing and executing the submitted queries in an efficient manner. It exploits data sharing among the queries then determines the order of the queries execution and finally, it allocates the queries to the appropriate worker nodes.

The output of this module is the list of the worker nodes that are responsible for executing the queries to send to the workload manager module and involves the following processes:

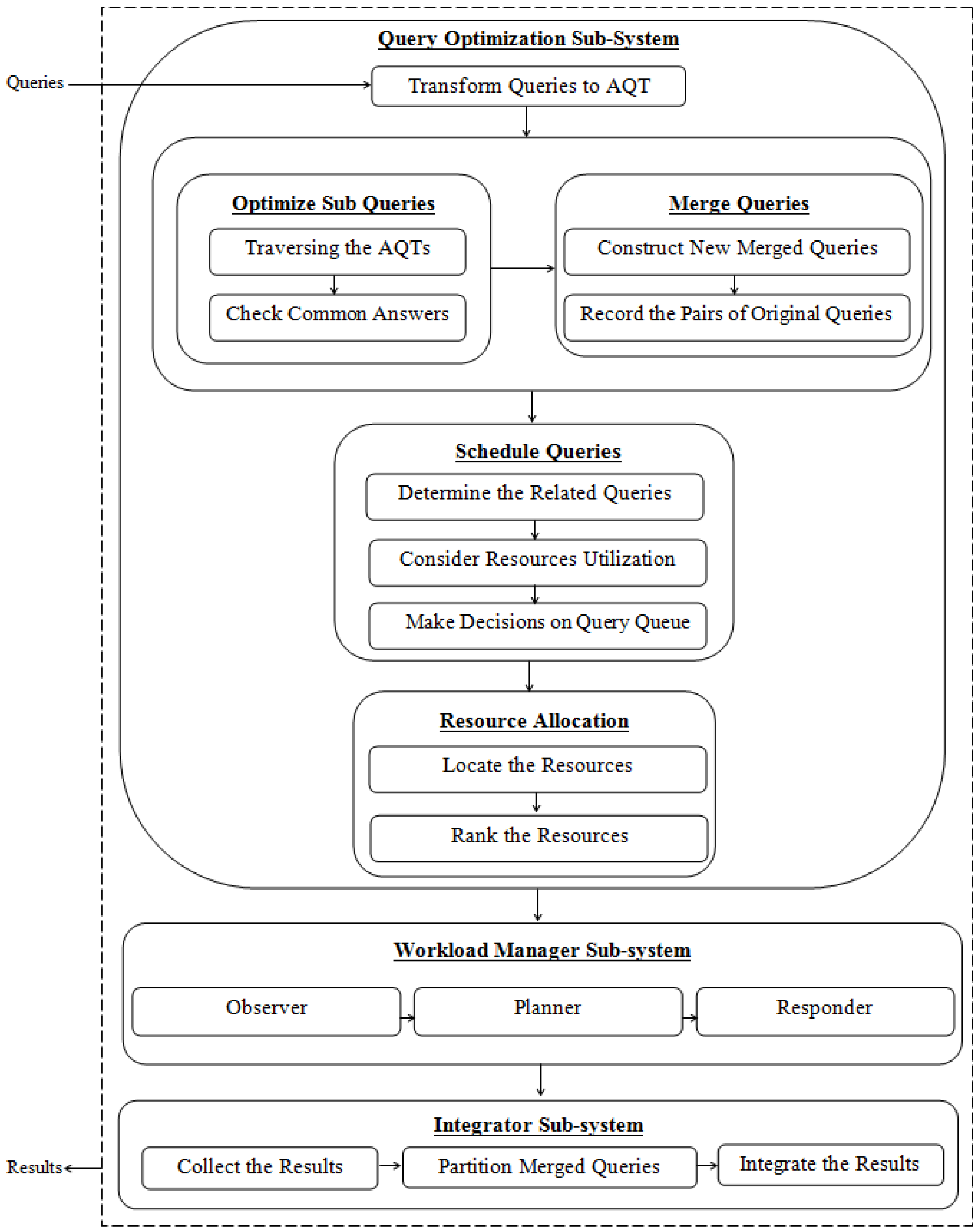

Fig. 1: Proposed query processing archite

**Transform queries to the Abstract Query Tree (AQT) process:** Each query is transformed to AQT by a parser which generates a query execution plan as a tree. The tree consists of query operations on its internal nodes such as; select operation and the relations included in the query on the tree leaves.

**Optimize sub-queries process:** Traversing the AQTs to determine which sub-queries will have common answers. It checks if there are sub-queries selecting data from the same table and it also checks specific conditions that must be satisfied then it specifies the list of related sub-queries to be sent to the merge process.

**Example:** For a given submitted query q: $\prod_L (\sigma_p (R))$ where L is the output attributes list, P is the selection predicate and R are queried relations. Let us assume the following assumptions:

- $L_o(q_i)$ is the list of output attributes
- $P(q_i)$ is the selection predicates of $q_i$
- $L_c(q_i)$ is the list of attributes that appear in $P(q_i)$

For two queries $q_i$ and $q_j$, let $R(q_i \cap q_j)$ be their common result. Sub queries can be merged in the case of three conditions that should all be satisfied to make sure that the two queries can be merged:

- $L_c (q_i) \subseteq L_o (q_i)$
- $L_c(q_j) \subseteq L_o(q_j)$
- $L_o(q_i) = L_o(q_j)$

We also consider different delay times of submitted queries to only merge the queries that have a positive impact on the query execution performance. For example, a short running query and a long running query can be merged if they satisfy the previous afore mentioned merging conditions; but in this case, the waiting time of the short running query will increase because of the merging step with the long running query, therefore it can lead to decreasing the performance. Therefore, we enhanced the merging process (Maghawry *et al*., 2010) through adding another condition to improve the overall query processing performance.

Our enhancement technique groups the related queries based on the number of rows that will be scanned. The queries can be merged only if their scan ratio is fallen within the same range of specific thresholds that are specified experimentally in our previous research (Maghawry *et al*., 2010) by applying this enhancement in the merging process, we can ensure that the merged queries will have a positive impact on their execution performance. Therefore, it can lead to improving the queries execution performance and response time.

**Merge queries process:** The new merged queries set are reconstructed by the query reconstruction mechanism (Chen *et al*., 2011) from the group of optimized sub-queries that have shared data to eliminate data redundancy among them and minimize communication overheads. The answer of the new merged query is used to compute the answers of the original queries before the merging step. For example, $q_3$ is the new merged query of $q_1$ and $q_2$ by satisfying the previous mentioned merging conditions, the answer to $q_3$ can get all the rows required by $q_1$and $q_2$ and also the answers to $q_3$ can get all the columns required by $q_1$ and $q_2$ and in addition the answers to $q_1$ and $q_2$ can be computed since the attributes required by their predicates are provided in the answer to $q_3$ because the predicates of the two queries are merged in the new merged query $q_3$. Therefore, the answers to $q_1$ and $q_2$ can be computed from the answers to $q_3$ Also in this process, we record all the pairs of queries that can be merged and the following information is recorded: <New merged query, condition original query > where the condition is the predicate used to compute the answer to the original query from the answer to the new merged query. All the pairs of original queries are recorded to compute their answers in the partitioning queries process within the Integrator module that will be introduced in this study. The previous processes were evaluated in our previous research (Maghawry *et al*., 2012).

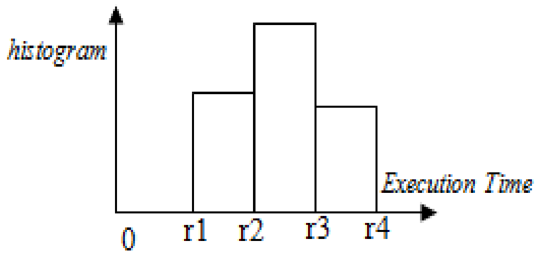

Fig. 2: Uniform distribution of execution time

**The scheduler technique:** Is responsible for constructing a queue of query executions. Our proposed scheduler presents an enhancement on the Shepherd technique that was proposed by Chi *et al.* (2013). This enhancement is presented by considering the related queries running in the system and taking into account the resources utilization. The Shepherd technique makes decisions about which query to execute next based on some defined rules. If query q follows a stored query template, the distribution of the execution time of this template is presented in a histogram with multiple buckets (r1, r2) as shown in Fig. 2. As in the preprocessing process, our scheduler detects if the submitted query has a known historic template or not. It considers the possible values of the execution time of query q while considering their SLA missed deadline cost. It relies on the probability distribution of the query execution time, instead of relying on a single point predication for the execution time of a query.

At a given time t, it computes the expected cost reduction between choosing a query q to run now or choosing another query to run and delay q further. Therefore the scheduler saves the waiting time of queries because it doesn't put the query within the queue unless

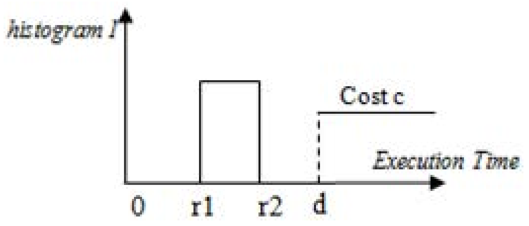

Fig. 3: Decomposition of query execution time histogram

it will improve the performance of the execution. The expected cost reduction is known as the priority score of q at time t and the query with the highest score starts the execution. So, our used technique saves the waiting time of queries because it doesn't put the query within the queue unless, it will improve the execution performance. The query q if started at time t, q's finishing time is uniformly distributed between the buckets assuming that each query has the SLA cost function with a single deadline d and the cost of missing the deadline. The scheduler computes the priority score for each histogram bucket at time t as shown in Fig. 3, the overall query priority score is computed by combining the decomposition of each bucket in the histogram.

Also, it is assumed that the further delay for each query follows the exponential distribution that's because the query execution time is not fixed but instead is changed. Hence, the distribution of the query execution time can be calculated and the overall Shepherd score P for each query q at time t can be computed as mentioned by Maghawry *et al*. (2014a, b). The proposed enhancement on the Shepherd technique depends on considering how much CPU and I\O cost will be needed to complete the queries execution and such information is obtained from the query optimizer of the database management system. The proposed scheduler orders the queries execution in a queue based on the shared requested data between the queries.

For example, a given submitted query may do many I/Os when executing alone, although the same query can run faster with the existence of another query that reads the same data, this is achieved because the requested data is found in the buffer pool which reduces the amount of necessary I/O's. Our scheduler efficiency is evaluated and presented by Maghawry *et al*. (2014a, b).

**Resource allocation process:** Is responsible for allocating each query to the appropriate available resources based on the ranking functions that are presented by Maghawry *et al*. (2012). The ranking function computing is based on the cloud resources' information such as: RAM amounts and CPU speed to select the appropriate resource for execution. Moreover, it dispatches the queries to the corresponding worker node and takes into consideration each worker node's capability to run the queries concurrently with minimizing the contention at the resources. The output of this module is the list of the assigned resources that are responsible for the queries execution. In this module, the Merging process and Allocation processes are combined to exploit the shared data among the queries then assigns the queries to suitable resources based on the resources ranking function to improve the query processing performance as presented by Maghawry *et al*. (2012).

**Workload management module:** During the queries execution, the module is responsible for controlling the queries execution across the running worker nodes. It ensures that nodes are used effectively and utilized as fully as possible without overloading any node. However, this module has processes that collect information about the nodes performance to diagnose and handle any failure or load imbalance that may occur during the execution. The resources load imbalance is handled by generating an assessment plan that redistributes the queries execution over the replicated nodes. The main advantages of this module are the improvement of the query processing performance and overall query response time through exploiting the database replicas in handling the workload imbalance. Furthermore, this module considers the resources contention, revising the feedback loop about the resources execution after the queries allocation. It holds the following main processes that were presented and described in our previous research (Maghawry *et al*., 2014a, b).

**Observer process:** Every 15 sec it dynamically collects the processor utilization values for each running worker node during the queries execution which is ideal (Performance Monitoring, https://software.intel.com/en-us/articles/use-windows-performance-monitor-for-infrastructure-health) for benchmarking scenarios. The performance information is the percentage of processing time which each worker node spends on processing queries. If these values exceed specific thresholds then this means a failure can occur with the current execution. If the load on the processor of any worker node exceeds 80% (Dam and Fritchey, 2009) it indicates that there is a possible overload on this node. In this case, it generates a notification containing the updated information of the loaded worker node and then it sends this notification to the planner process to generate an assessment plan to handle the failure that may occur during the execution.

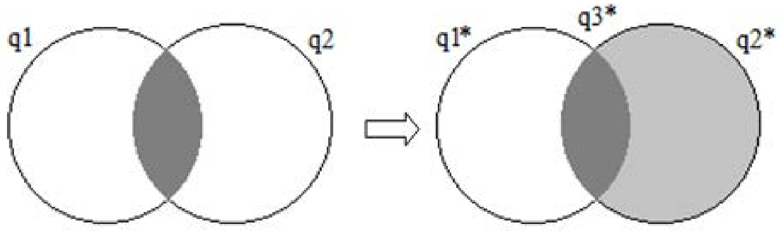

Fig. 4: Partition the queries results

**Planner process:** Initiates when receiving a notification from the Observer process that contains information about the loaded node. It collects the performance information about the available replicas of the loaded node. In addition, it assigns the failure queries on the loaded node to the most available unloaded replica to recover the load and contention on the worker's node. Therefore, it generates an assessment plan by replacing the loaded worker node with the appropriate replica to handle the worker node contention. Furthermore, the plan contains the queries that are assigned to be executed on the chosen replica. Finally, the assessment plan is sent to the Responder process for execution.

**Responder process:** Receives the assessment plan from the Planner as a notification. This process uses the information in the assessment plan to specify the failed queries on the loaded worker node and the suitable replica for executing these queries. The main goal of this process is to recover the nodes contention through applying the enhanced workload distribution plan, therefore it dispatches the failed queries to the corresponding replica and finally kills the most recent queries execution on the loaded node.

**Integrator module:** In this study, the integrator module implementation details are presented. This module is responsible for retrieving the concurrent queries results from the distributed nodes then returns the queries results to the users. Moreover, it partitions the results in the case of merged queries to get the overlapping data and the remaining unshared data between the original queries as presented by Chen *et al*. (2011). This partitioning process is responsible for constructing the results of reconstructed queries from the original queries that are constructed in the merging process.

For instance if there is data sharing between two submitted queries $q_1$ and $q_2$ as shown in Fig. 4. They will be merged in Merging process then sent to the resources for execution, after the answers to $q_1$*, $q_2$* and $q_3$* are returned, partition the queries $q_1$ and $q_2$ results to the queries $q_1$*, q2* and q3* to eliminate overlapping data. So, the results of $q_1$ can be computed directly from the results of $q_1$* and $q_3$*. The result to $q_2$ can be obtained directly from the results to $q_2$* and $q_3$*. Finally, the results of the finished queries are prepared by collecting the results of the same queries from the distributed instances.

The set Q = {$q_1$, ..., $q_n$} is used to represent the queries submitted and also record all queries that can be merged in Q* = {<$q_1$, $q_2$ >, ... <$q_3$, $q_4$, ..., $q_n$>}. For each merged queries, all the information about their original queries and their new merged query are recorded as a group <q*, $q_1$, $q_2$>. For example <q*, $q_1$, $q_2$>, the following information is recorded:

- <q*, $cond_1$, $q_1$>
- <q*, $cond_2$, $q_2$>

Where $cond_1$ is the condition used to compute the result of the original query $q_1$ from the result to the merged query q* and $cond_2$ is the condition used to compute the result of the original query $q_2$ from the result to the merged query q*. This condition is specified in the merging process that is based on the queries predicate.

After evaluating queries in Q*, the results are returned from the resource to the integrator module, it can compute the results to the original queries $Q_1$ = {$q_1$, ..., $q_n$} by checking the recorded information such as <q*, $cond_n$, $q_1$> that means the result of $q_1$ is a sub-set of the result of q* and its results can be computed by applying the condition $cond_n$ on q*.

The main goal of this module is to collect the results from running resources then prepare the results to be sent back to the users, it also takes into the consideration the results of the merged queries to be partitioned before sending to the users.

## RESULTS

**Experimental results and evaluation:** We evaluated the performance of the proposed work on Amazon EC2 (Amazon Elastic Compute Cloud (EC2): http://aws.amazon.com/ec2/) using standard small and medium instances in the US-West region. By default, a small instance (t2.micro) has the following hardware configurations: 1 EC2 Compute Unit (i.e. 1 virtual core), 1 GB of main memory, 30 GB of local instance storage and a 64-bit platform. Medium instances (t2.medium) have the following hardware configurations: 1 EC2 Compute Unit (i.e. 2 virtual core), 4 GB of main memory, 30 GB of local instance storage and a 64-bit platform. The experimental setup consists of 10 machines-1 master node and 9 worker nodes.

For each worker node instance type, we used Microsoft Windows Server 2008 R2 as the operating system configured with a Microsoft SQL Server 2008 R2 as the database server. The master machine runs on a Inter Core 7, 2.60GHz CPU, 6GB of main memory and 1TB hard disk. Different numbers of queries are processed by the proposed system and the technique is implemented in Microsoft Visual Studio, 2010. Over the lifetime of the experiment, varied types of active instances were used to evaluate the proposed system, each run was repeated five times and the average execution time values reported. After each execution, the database system was restarted to clean up the bufferpool and to bring the database system to a steady state.

One of our contributions in this study is the evaluation of our proposed query processing approach is conducted over a real world cloud using the Amazon EC2 infrastructure provisioning service unlike our previous research and some researches relied on simulated environments (Maghawry *et al*., 2012, 2014a, b).

The TPC-H database (Transaction Processing and Database Benchmark, http://www.tpc.org/tpch/) is used as a dataset (scale factor 1) to test the proposed work. The TPC-H database has eight relations: CUSTOMER, LINEITEM, NATION or DERS, PART, PARTSUPP, REGION and SUPPLIER. The proposed system supports several SQL queries in cloud computing systems with the stored data relations on the worker nodes. Generally, each table is divided horizontally into x parts, each part replicated y times and are saved in different worker nodes, y is usually less than the number of worker nodes in a cluster which y = 2 for most cloud systems (Zhao *et al*., 2010). The master node has the storage information about each replica of each partition. We focus on using a read-only SQL query, TCPH queries are used with the form of generalized selection, projection and join.

Each module in our system was previously compared with other approaches (Chen *et al*., 2011; Chi *et al*., 2013) to evaluate its efficiency. Therefore in this study, we aim to evaluate the efficiency of the whole system by the integration of the modules within one working architecture. For comparisons, the SQL query processing algorithm called ESQP and stands for Efficient SQL Query Processing proposed by Zhao *et al.* (2010) has been chosen. Because they similarly proposed an architecture that addresses all query processing issues instead of focusing on only one issue related to the query processing approach. They used data replicas in cloud storage for processing the SQL queries. ESQP did not deal with just single issue such as query optimization or scheduling as in most of the research.

So in this study to the efficiency of the proposed approach, it was compared with an ESQP (Efficient SQL Query Processing) approach that is described by Zhao *et al.* (2010). Their approach was inspired by the MapReduce idea, in which a job is divided into several tasks. When a user submits a query in their approach, a master node divides a user query into several sub-queries; every sub-query has to wait in the queue of the worker nodes where the query data is stored. To balance the load, their approach relies on the scheduling strategy in dispatching the sub-query. The idea of their query optimization is to balance all waiting queues on the worker nodes. Before choosing a sub-query, they calculate the length of the waiting queue for each of the worker nodes.

The variance of each list is computed and the sub-query corresponding to the smallest variance is assigned to the worker node but their work has limitations on executing large scales of concurrent queries which brings additional scalability problems (Zhao *et al*., 2010). Their query processing approach consists of three key components:

- Query transformation: Decomposes each user query into a set of independent sub-queries that can be executed in parallel on the nodes
- Query dispatch: Schedules and assigns the sub-queries to the worker nodes. The scheduling technique is relied on their approach to achieve load balancing across the nodes
- Sub-query execution: Worker nodes process sub-queries and return the results to the clients as soon as possible, even if only one record of all results is ready. They believe that current query processing can save a lot of time so that they don’t need to wait for the results to be transferred even if only one record of all results is ready. But in some cases, the results of sub-queries are not just the results of the original query so in this case some re-treatment is required and they stop the mission timer at the point that the first result is received by client. Therefore, we consider that the results of the query cannot be returned to users until all sub-queries on the worker nodes are completely executed

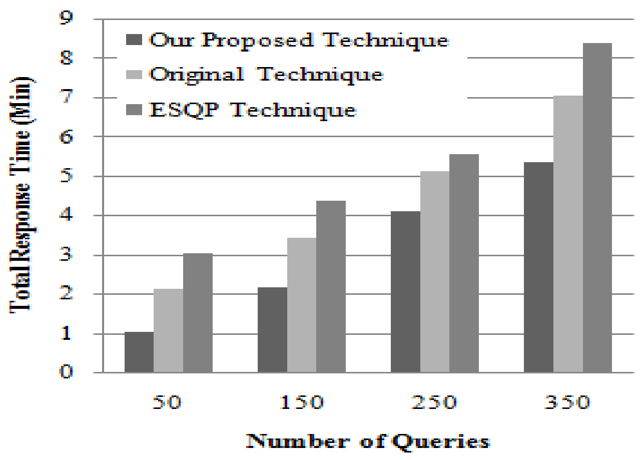

Fig. 5: The average queries execution time for different query types

Previous technique addresses several issues in the query processing topic by handling increasing numbers of worker nodes and also they depend on their scheduling technique to get the load balance which uses a heuristic method-variance that needs a more accurate measurement of load balance (Zhao *et al*., 2010). Our proposed query processing technique combines the query optimization and query resource allocation in addition to observing the concurrent queries execution over the running resources and responding to any load imbalance by applying the proposed workload management module on the cloud environment.

A key goal of the proposed processing technique is to minimize the query response times and improve resource utilization in the cloud environment. Response time is used as the metric in the experiments which is the interval between the query started and its results returned to the user so the experimental results in this section demonstrate the performance of the proposed approach. For evaluation, the experiments were set with different machines capabilities and different numbers and types of workloads. Each experiment was executed five times and the average response time is calculated by applying the proposed technique against the ESQP technique. Also, it was compared against a normal technique which dispatches the queries directly for execution.

The results are shown in Fig. 5. Figure 5 shows the average execution time in minutes for the different numbers of concurrent queries with short and long running queries workloads by applying the proposed technique, ESQP and normal technique. The queries with select and join types were used in the proposed experiment. As shown in the results, the average execution time of the queries by applying the proposed system can reduce the queries response time over the two other approaches. This is done because query merging and scheduling techniques are integrated in our proposed technique and this integration improves the performance of the queries execution.

In executing 350 queries, the specific instance was hit with 70% of the queries to test the proposed workload technique. During the execution, the processor utilization of the corresponding instance exceeds 80% that means there is a load on this instance and this load can lead to execution failure. Therefore by applying the proposed Workload Management Module, the latest queries with failure execution on this instance are assigned to its suitable replica to continue their execution without overloading the instances. ESQP has issues in handling a load with an increasing number of queries and worker nodes because of the computation of the large matrix. In addition, it relies on just their scheduling techniques to get the load balance and does not revise the resources feedback loop about execution after the resources allocation as exists in our workload management processes so as shown in the results, the average execution time of the queries with handling the load imbalance can reduce the queries response time over the ESQP by 40% and over the normal techniques by 28%.

The proposed technique is also tested with simple queries in the select-project query form of $\pi_L\ (\sigma_p(R))$ where L is the output attributes list, P is the predicate of selection and R are the tables that are queried. Different numbers of queries are submitted and the average response time is computed. As the results shown in Fig. 6, applying the proposed technique also improves the queries execution time over the ESQP by 37% and over the normal techniques by 35%.

The query optimization time is also computed to test the performance of the proposed technique and ESQP technique. The query optimization time, is the interval between the queries submitted, optimized, scheduled and

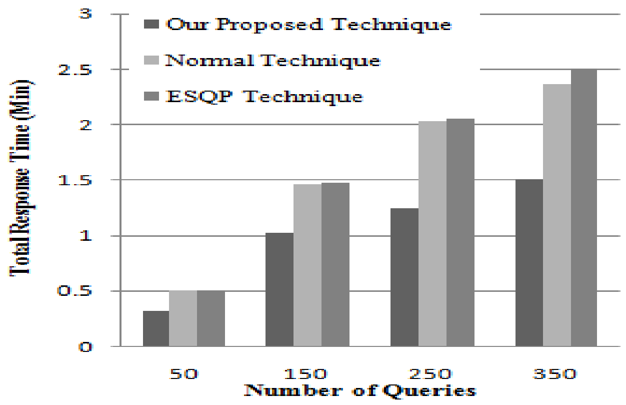

Fig. 6: The average queries execution time for the same query type

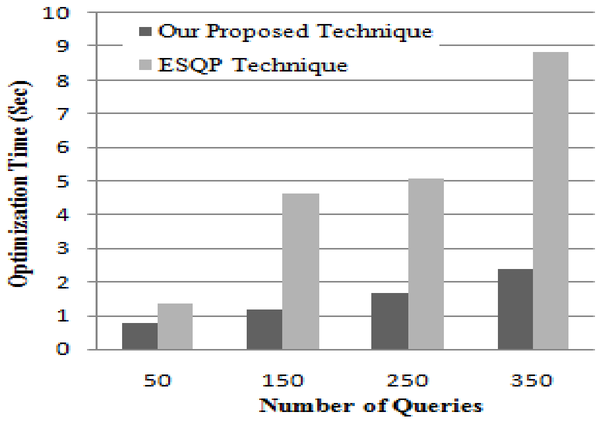

Fig. 7: The average queries optimization time

assigned to the workers nodes. The average of the total query optimization time for different numbers of queries is shown in Fig. 7. The ESQP technique needs to construct and compute a matrix to schedule the sub-queries assignment on worker nodes which takes more time for the queries optimization so the results show that the time taken to optimize the submitted queries by applying our proposed technique is less than applying the ESQP technique by 70%.

To test the performance of the proposed technique against the ESQP, we measured the query throughput for each technique which is the number of executed queries per second. At first, the query throughput was measured for a single instance to know how it will behave on a single server and then we used it as a reference in our comparisons, the same was done using the distributed optimized techniques. The results retrieved from the optimized techniques should be equal to or better than the single instance results.

As noticed from the single instance graph in Fig. 8, the database response time was almost constant within a region but then started to deteriorate so this will be marked as the database optimal performance region (30-40 queries). Therefore, the average response time of the optimal performance region was used to calculate the single instance throughput. The same approach was followed with both the proposed technique and the ESQP technique and the average response time of different numbers of queries are shown in Fig. 9. From the previous results, we can conclude in Table 1 which calculates the query throughput. The results show that the proposed technique has the highest query throughput against the ESQP technique. This is achieved because the proposed approach has the query merging approach which reduces the number of queries that will be scheduled and allocated over the resources so, it improves the query throughput over the proposed approach.

Finally, we set different types of instances to test the proposed system using machines with different capabilities. Half of the worker nodes are assigned to t2. micro and the other half to t2. medium. As the proposed approach integrates the query processing techniques in

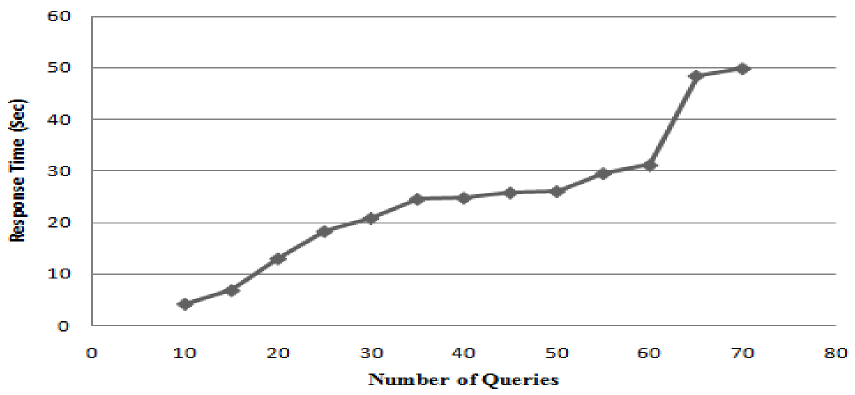

Fig. 8: The query throughput for single instance

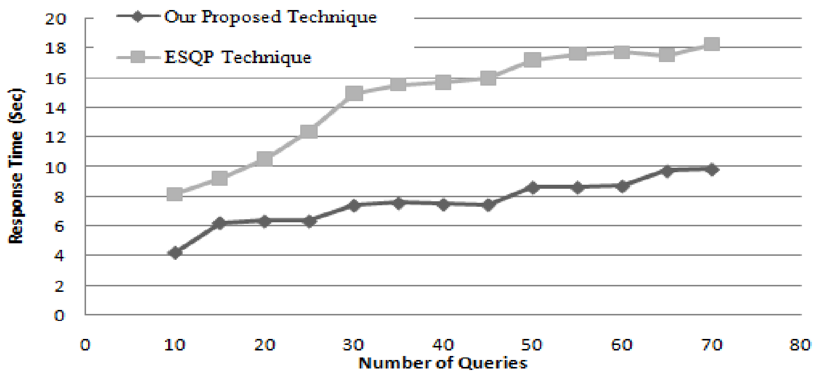

Fig. 9: Measuring the query throughput

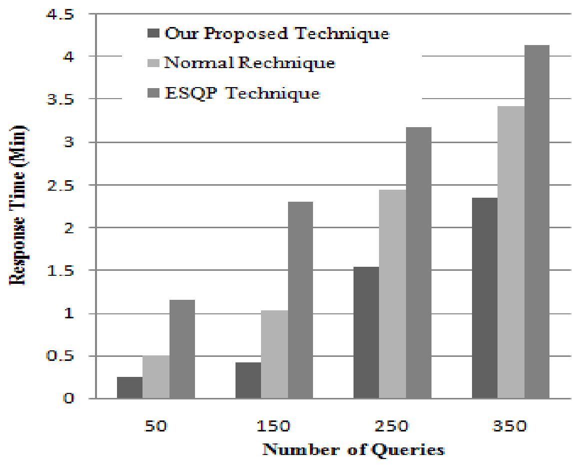

Fig. 10: The average queries execution time for different machine capabilities

Table 1: Measuring the query throughput

| Parameters | Number of queries | Response time (sec) | Query throughput (# queries\sec) |
|---|---|---|---|
| Our proposed technique | 40 | 7.4 | 5.4 |
| ESQP technique | 40 | 15.6 | 2.5 |
| Single instance | 40 | 24.7 | 1.6 |

an optimized way so as shown in Fig. 10 by applying the proposed approach it improves the queries execution time over the ESQP by 57% and over the normal technique by 38% using different types of instances capabilities.

## DISCUSSION

In this study, we introduced a methodology for performing query processing to enhance the overall performance of queries execution over cloud environments. The increasing number of users requests data from deployed virtual nodes over the cloud can lead to increased load in data management systems. This can cause node contention with rapidly changing computational properties, therefore, the proposed technique overcomes two significant challenges: Minimizing queries response times and maximizing the utilization of cloud resources.

For improving the response time, the query optimization and scheduling techniques are proposed. In addition to reducing resources contention, the efficient workload management technique is proposed by involving a feedback about the resources execution performance comprising of observing, planning and responding to any overloaded instance during the queries execution.

The proposed technique manages the life cycle of the client requests, selecting the worker nodes and balancing the load across these virtual nodes and so it is essential to incorporate a collection of techniques such as; optimizing, scheduling and load management to achieve efficient accessing and querying across the distributed data sources. These techniques are integrated and evaluated in one working approach of the integrated query processing methodology to improve the query execution performance.

Experiments have been performed in a real world by the Amazon EC2 infrastructure provisioning service which is considered one of the contributions. The evaluation of the proposed query processing approach is conducted using different measures such as; the query response time, the query throughput and query optimization time. The measurement is conducted over different query types and machine capabilities.

The results prove that the proposed query processing approach with workload characterization over the cloud improves the queries execution time over the ESQP by 37% and over the normal techniques by 35%. Also using different types of instances, it improves the queries execution time over the ESQP by 57% and over the normal technique by 38%.

In addition, the results prove that the time taken to optimize the submitted queries by applying the proposed technique is less than by applying the ESQP technique by 70%.

## CONCLUSION

This study proposes an efficient query processing approach on structured data in cloud to reduce resource contention and handle the degradation in the query processing performance. The key innovation in our approach is the efficiency of the processes arrangement within the combined approach's modules which highly improves the queries processing performance. Our proposed approach is evaluated by applying a combination of query optimization, scheduling and workload management techniques in terms of the query processing performance. The evaluation is conducted over a real world cloud using the Amazon EC2 infrastructure provisioning service which considered one of our contributions in this study. The results prove a significant benefit against existing approaches with regards to the overall query processing performance.

## REFERENCES

Ahmad, M., A. Aboulnaga, S. Babu and K. Munagala, 2011a. Interaction-aware scheduling of report-generation workloads. VLDB J. Int. J. Very Large Data Bases, 20: 589-615.

Ahmad, M., S. Duan, A. Aboulnaga and S. Babu, 2011b. Predicting completion times of batch query workloads using interaction-aware models and simulation. Proceedings of the 14th International Conference on Extending Database Technology, March 21-24, 2011, ACM, Uppsala, Sweden, ISBN: 978-1-4503-0528-0, pp: 449-460.

Akdere, M., U. Cetintemel, M. Riondato, E. Upfal and S.B. Zdonik, 2012. Learning-based query performance modeling and prediction. Proceedings of the 2012 IEEE 28th International Conference on Data Engineering (ICDE), April 1-5, 2012, IEEE, Washington, DC, USA., ISBN: 978-1-4673-0042-1, pp: 390-401.

Avnur, R. and J.M. Hellerstein, 2000. Eddies: Continuously adaptive query processing. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 15-18, 2000, ACM, Dallas, Taxas, ISBN: 1-58113-217-4, pp: 261-272.

Brandic, I. and S. Dustdar, 2011. Grid vs Cloud-A technology comparison. Inf. Technol. Methods Appl. Comput. Sci. Inf. Technol., 53: 173-179.

Chen, G., Y.G. Wu, J. Liu and G.W.M. Yang, 2011. Optimization of sub-query processing in distributed data integration systems. J. Network Comput. Appl., 34: 1035-1042.

Chi, Y., H. Hacigumus, W.P. Hsiung and J.F. Naughton, 2013. Distribution-based query scheduling. Proc. VLDB. Endowment, 6: 673-684.

Chi, Y., H.J. Moon, H. Hacigumus and J. Tatemura, 2011. SLA-tree: A framework for efficiently supporting SLA-based decisions in cloud computing. Proceedings of the 14th International Conference on Extending Database Technology, March 21-24, 2011, ACM, Uppsala, Sweden, ISBN: 978-1-4503-0528-0, pp: 129-140.

Dam, S. and G. Fritchey, 2009. SQL Server 2008 Query Performance Tuning Distilled. 2nd Edn., Apress, New York, USA., ISBN: 978-1-4302-1902-6, Pages: 497.

Duggan, J., O. Papaemmanouil, U. Cetintemel and E. Upfal, 2014. Contender: A resource modeling approach for concurrent query performance prediction. oceedings of 17th International Conference on Extending Database Technology, March 24-28, 2014, Athens, Greece, ISBN: 978-3-89318065-3, pp: 109-120.

Duggan, J., U. Cetintemel, O. Papaemmanouil and E. Upfal, 2011. Performance prediction for concurrent database workloads. Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, June 12-16, 2011, ACM, Athens, Greece, ISBN: 978-1-4503-0661-4, pp: 337-348.

Ganapathi, A., H. Kuno, U. Dayal, J.L. Wiener and A. Fox *et al*., 2009. Predicting multiple metrics for queries: Better decisions enabled by machine learning. Proceedings of the IEEE 25th International Conference on Data Engineering ICDE'09, March 29-April 2, 2009, IEEE, Shanghai, China, ISBN: 978-1-4244-3422-0, pp: 592-603.

Gupta, C., A. Mehta and U. Dayal, 2008. PQR: Predicting query execution times for autonomous workload management. Proceedings of the International Conference on Autonomic Computing ICAC'08, June 2-6, 2008, IEEE, Chicago, Illinois, USA., ISBN: 978-0-7695-3175-5, pp: 13-22.

Jarke, M., M. Jeusfeld and C. Quix, 2014. Data-centric intelligent information integration from concepts to automation. J. Intell. Inf. Syst., 43: 437-462.

Kossmann, D. and T. Kraska, 2010. Data management in the cloud: Promises, state-of-the-art and open questions. Database Spectr., 10: 121-129.

Krompass, S., D. Gmach, A. Scholz, S. Seltzsam and A. Kemper, 2006. Quality of Service Enabled Database Applications. In: Service-Oriented Computing-ICSOC 2006, Asit, D. and W. Lamersdorf (Eds.). Springer Berlin Heidelberg, Berlin, Germany, ISBN: 978-3-540-68147-2, pp: 215-226.

Krompass, S., H. Kuno, U. Dayal and A. Kemper, 2007. Dynamic workload management for very large data warehouses: Juggling feathers and bowling balls. Proceedings of the 33rd International Conference on Very Large Data Bases, September 23-28, 2007, VLDB Endowment, University of Vienna, Austria, ISBN: 978-1-59593-649-3, pp: 1105-1115.

Lee, R., M. Zhou and H. Liao, 2007. Request window: An approach to improve throughput of RDBMS-based data integration system by utilizing data sharing across concurrent distributed queries. Proceedings of the 33rd International Conference on Very Large Data Bases, September 23-28, 2007, VLDB Endowment, University of Vienna, Austria, ISBN: 978-1-59593-649-3, pp: 1219-1230.

Li, J., A.C. Konig, V. Narasayya and S. Chaudhuri, 2012. Robust estimation of resource consumption for sql queries using statistical techniques. Proc. VLDB. Endowment, 5: 1555-1566.

Luo, G., J.F. Naughton and S.Y. Philip, 2006. Multi-Query SQL Progress Indicators. In: Advances in Database Technology-EDBT 2006. Yannis, I., H.S. Marc, J.W. Schmidt, F. Matthes and M. Hatzopoulos *et al*., (Eds.). Springer Berlin Heidelberg, Berlin, Germany, ISBN: 978-3-540-32960-2, pp: 921.

Maghawry, E.A., R.M. Ismail, N.L. Badr and M.F. Tolba, 2012. An Enhanced Resource Allocation Approach for Optimizing Sub Query on Cloud. In: Advanced Machine Learning Technologies and Applications. Hassanien, A.E., A. Badeeh, M. Salem, R. Ramadan and T.H. Kim (Eds.). Springer Berlin Heidelberg, Berlin, Germany, ISBN: 978-3-642-35325-3, pp: 413-422.

Maghawry, E.A., R.M. Ismail, N.L. Badr and M.F. Tolba, 2014a. An enhanced queries scheduler for query processing over a cloud environment. Proceedings of the 2014 9th International Conference on Computer Engineering & Systems (ICCES), December 22-23, 2014, IEEE, Cairo, Egypt, ISBN: 978-1-4799-6593-9, pp: 409-414.

Maghawry, E.A., R.M. Ismail, N.L. Badr and M.F. Tolba, 2014b. Queries Based Workload Management System for the Cloud Environment. In: Advanced Machine Learning Technologies and Applications. Hassanien, A.E., A.B.M. Salem, R. Ramadan and T.H. Kim (Eds.). Springer International Publishing, Cham, Germany, ISBN: 978-3-319-13460-4, pp: 77-86.

Maghawry, E.A., R.M. Ismail, N.L. Badr and M.F. Tolba, 2016. Enhancing query optimization technique by conditional merging over cloud computing. Proceedings of the 1st International Conference on Advanced Intelligent System and Informatics (AISI2015), November 28-30, 2015, Springer International Publishing, Beni Suef, Egypt, ISBN: 978-3-319-26688-6, pp: 347-356.

Paton, N., M.A.D. Aragao, K. Lee, A.A. Fernandes and R. Sakellariou, 2009a. Optimizing utility in cloud computing through autonomic workload execution. Bull. Tech. Committee Data Eng., 32: 51-58.

Paton, N.W., C.J. Buenabad, M. Chen, V. Raman and G. Swart *et al*., 2009b. Autonomic query parallelization using non-dedicated computers: An evaluation of adaptivity options. VLDB. J., 18: 119-140.

Paton, N.W., D.M.A. Aragao and A.A. Fernandes, 2012. Utility-driven adaptive query workload execution. Future Gener. Comput. Syst., 28: 1070-1079.

Schroeder, B., H.M. Balter, A. Iyengar and E. Nahum, 2006. Achieving class-based QoS for transactional workloads. Proceedings of the 22nd International Conference on Data Engineering ICDE'06, April 3-7, 2006, IEEE, Carnegie Mellon University, Pittsburgh, Pennsylvania, ISBN: 0-7695-2570-9, pp: 153-153.

Sheikh, M.B., U.F. Minhas, O.Z. Khan, A. Aboulnaga and P. Poupart *et al*., 2011. A bayesian approach to online performance modeling for database appliances using gaussian models. Proceedings of the 8th ACM International Conference on Autonomic Computing, June 14-18. 2011, ACM, Karlsruhe, Germany, ISBN: 978-1-4503-0607-2, pp: 121-130.

Subramanian, I., C. McCarthy and M. Murphy, 2000. Meeting performance goals with the HP-UX workload manager. Proceedings of the 1st Workshop on Industrial Experiences with Systems Software WIESS, October 79-80, 2000, Ibrarian, San Diego, California, pp: 79-80.

Tian, F. and D.J. DeWitt, 2003. Tuple routing strategies for distributed eddies. Proceedings of the 29th International Conference on Very Large Data Bases, September 9-12, 2003, VLDB Endowment, Berlin, Germany, ISBN: 0-12-722442-4, pp: 333-344.

Tozer, S., T. Brecht and A. Aboulnaga, 2010. Q-Cop: Avoiding bad query mixes to minimize client timeouts under heavy loads. Proceedings of the 2010 IEEE 26th International Conference on Data Engineering (ICDE), March 1-6, 2010, IEEE, Long Beach, California, ISBN: 978-1-4244-5445-7, pp: 397-408.

Zhao, J., X. Hu and X. Meng, 2010. ESQP: An efficient SQL query processing for cloud data management. Proceedings of the second international Workshop on Cloud Data Management, October 26-30, 2010, ACM, Toronto, Ontario, Canada, ISBN: 978-1-4503-0380-4, pp: 1-8.