JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 29, 793-810 (2013)

DOI:10.1688/JISE.2013.29.5.1

Harnessing Cloud Computing for Dynamic Resource Requirement by Database Workloads

CHEE-HENG TAN[1] AND YING-WAH TEH[2]
[1,2]*Faculty of Computer Science and Information Technology*
University of Malaya
50603 Kuala Lumpur, Malaysia
E-mail: {[1]cheeheng@siswa; [2]tehyw@}um.edu.my

In this paper we illustrate a mechanism to address a very important aspect in the Cloud hosting strategy – resource management. We want to show how database hosting can benefit from this technology better, together with analysis on some of the causes and concerns of hosting databases in the Cloud. A graphical presentation is illustrated, where input is obtained from database workloads to gauge the adequacy of the resource, and plan for future. The essence of the proposal is based on linear regression correlation between workloads' execution time and corresponding server load. This paper strives to address the requirement prior to resource contention, so that underlying hardware planning in the Private Cloud can be carried out in orderly fashion and with adequate timeline. Subsequently Fuzzy Logic concept is utilized to represent the variables' relationship in the proposal, to ease the deterministic calculation in the algorithm during development phase. A supervisory mechanism to ensure validity of SQL response time used as input to the algorithm is proposed, so to ensure solidarity on the input parameters. The objective is to ensure credible transactions can run freely without hardware resource constraint throughout the tenure of the application offering, in association with the Cloud computing technology.

Keywords: resource planning, workload management, cloud computing, SQL performance tuning, private cloud, database hosting

1. INTRODUCTION

Traditional data hosting technology is fast being replaced by Cloud Computing. The potential of Cloud Computing is tremendous in view of demand from today's IT infrastructure. Voted as Top 11 technologies of the decade [1] by *IEEE Spectrum*, Cloud Computing is enabling the agility required to accelerate the time to market of new products and services while reducing the cost to design, build, deploy and support these products and services, and is considered as generally best practice for Enterprise Architecture [2]. It is changing the way IT services are perceived, delivered and consumed.

There are various perceptions in defining Cloud Computing. Zhang *et al.* [3] defines Cloud Computing as evolution of grid computing, and it comprises of thin clients, Grid Computing and Utility Computing. Buyya *et al.* [4] differentiate between Cloud Computing and Grid Computing at the *virtualization* level, where Cloud is defined as next-generation data centers with nodes "virtualized" through hypervisor technologies, dynamically "provisioned" on demand as a personalized resource collection. The virtualization provides the ease and flexible capability on resource allocation, which is the key motivation for this paper. Foster *et al.* [5] compares Cloud and Grid in length; and from

Received May 7, 2012; revised August 10, 2012; accepted September 21, 2012.
Communicated by Jan-Jan Wu.

dynamic resource provisioning perspective, Cloud is deemed more flexible than Grid, as Cloud is leveraging virtualization technologies more extensively. Zhang *et al.* [6] define Cloud Computing in a more end-user-friendly way, by quoting: *Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction*. It is this ease of effort in application hosting that makes Cloud a popular and fascinating choice. The financial service firm Merrill Lynch estimates that within the next five years, the annual global market for Cloud Computing will surge to $95 billion. In a May 2008 report, Merrill Lynch estimated that 12% of the worldwide software market would go to the Cloud in that period [7]. Public Cloud vendors are building extremely large-scale, commodity-computer Data Centers in low cost locations, and they uncovered factors of 5 to 7 decrease in cost of electricity, network bandwidth, operations, software, and hardware available at these very large economies of scale [8].

This resource management proposal on database server is planned and designed, taking into perspective the current Cloud vendors' services offering. At the time when we wrote this, Microsoft has released SQL Azure [9] which takes advantages and benefits of Public Cloud. IBM SmartCloud Application Services at the PaaS level [10] is one example that enables on-premise Cloud hosting. In our discussion, we focus on on-premise solution, due to data integrity and security reasons. It is not the problem with the technology itself, as human always found ways to address shortcomings or challenges technically. We think that the current perception and skepticism of Enterprises on security and reliability will delay the Public Cloud adaptation. In this case, Private Cloud is the easier solution. As described by Harms and Yamartino [11], the Horseless Carriage Syndrome when automobiles were introduced in early 20th century perhaps will slow down the embracement of SQL Azure; however the economics of the Cloud might overwhelm the constraining factors in time to come. Amazon is taking a step forward by introducing AWS GovCloud (US), which is hosted in Amazon Web Services [12]. Its compliance with US International Traffic in Arms Regulations (ITAR) and Federal Information Processing Standard (FIPS) Publication 140-2 is hoped to prove to the world its robustness of data hosting in Public Cloud. Google claims its strength in data security via ten components of Google's multi-layered security strategy incorporated in Google Apps [13]. Oracle through its Exalogic Elastic Cloud product provides similar offering for Public and Private Cloud, plus Hybrid Cloud that is capable of Cloud bursting [14].

Public Cloud Computing has been widely accepted and deployed for web-based application. However its adoption for mission critical database operations is still at early stage. While we anticipate that Public Cloud will mature and flourish, we want to write this paper to detail on the resource administration mainly targeting databases hosted in on-premise Private Cloud as we expect that database hosting on Private Cloud is going to thrive for quite a while. Another important challenge that needs to be noted when hosting database system in Public Cloud is the *data locality* issue. Database operations are mostly IO intensive. As described by Foster *et al.* [5], when the resource needs to be scaled, distance of the data relative to the available computational resources also increases. This posts a bottleneck in term of IO as moving data in WAN for data processing is much slower compared to local disk storage. More efficient data-aware schedulers [15] are needed than what is available today.

Nevertheless, our presented mechanism is applicable for resource planning in Public or Hybrid Cloud from resource management perspective. We agree with Harms and Yamartino [11] that the full advantage of Cloud Computing can only be properly unlocked through proper intelligent resource management, which is the essence of what we propose in this paper. Dutreilh *et al.* [16] analyses and presents the challenges of the automated resource management policies in the Cloud. We think that the automated resource allocation approach is suitable for scaling of resource at the application layer. We take motivation from here, and suggest another way to suit the database hosting environment. Due to problem with current RDBMS licensing model and the unpredictable nature of SQL queries, we think that over-dynamic resource allocation paradigm will take a while before it is widely adopted for database hosting in Cloud.

In the model we proposed, unless the allocated resource is provisioned specifically for a short timeframe of surged transactions, the resource has the tendency to accumulate and stay in the *VM*, via static on-demand request. At this moment, it is not easy to map QoS requirements to low-level resource requirement such as CPU and memory requirements [6]. This is especially true in database hosting that has many variables in its operations. Hence dynamic resource allocation model is not presented. Nevertheless, since the hosting utilizes virtualization paradigm in Cloud, online dynamic resource provisioning is possible with today's Cloud technology.

Hosting databases in this technology also makes Load Testing easier, as the same set of hardware can be provisioned in Public Cloud or Virtual Private Cloud. This can be accomplished as virtualization allows entire *VM* state to be transmitted across the network from 1 data center to another. Hence there is no outage requirement to conduct Load Testing in exiting host and still making the Load test result relevant to the actual hosting environment. Our proposed model is designed to work during steady-state ongoing production operations, taking advantage of the ease of resource provisioning offered by virtualization in the *infrastructure layer* [6]. The novelty of this proposal takes 2 keys parameters: The total elapsed time taken for the database operations and the server load at the corresponding instance. The linear correlation between these 2 parameters serves the fundamental of this proposal. By leveraging the running time history, the proposed scheme obtains the trend of history and is possible to predict how much resource is needed in the future. The focal point when resource will hit breakpoint serves the main objective of this paper. Subsequently we present a mechanism to examine resource requirement by the SQL transactions to ensure only tuned workloads are running in the host.

We presented our research motivations, challenges and objectives in section 2. Subsequently we examine the related researches in section 3. Section 4 details the algorithm of our proposal. Subsequently section 5 shows how the proposal works by experiments. Section 6 discussed the equilibrium needed for obtaining accurate data versus system overhead. Then a guardian to ensure legitimacy of the input data is discussed in section 7.

2. RESEARCH MOTIVATIONS, CHALLENGES AND OBJECTIVES

In large and complex Enterprises, the cost of running businesses spans across multiple domains. For each domain to forecast the exact and precise cost model needed for the business operations is crucial for the companies to accurately predict its ROI. To remain

competitive in today's rapidly changing global economy, an organization needs to ensure its operational cost is spent only adequately for the business. From IT Organization's perspective, this translates to the requirement for an efficient architectural design of application services, where all detailed requirements are known in advance for monetary planning. As application's data manipulation is provided by running SQL from the back-end databases, activities at this database layer are focused in this paper. To operate the database in oversized hardware to cushion the fear of impact from lack of required processing power will not be cost effective as it incurs unnecessary wastage, and to go down too little will be too less for the database operations. During the onset capacity planning stage, the actual requirement of computational power and storage, whether it's designed during the startup or meant for subsequent growth of the database operations, will not be entirely known. Also, subsequent scalability and performance must continue to meet the demand of the business. As the profit of the business very much depend on the precision-match of the planned cost of operations and the intended performance, a solution needs to arrive quickly. For databases that support fast growing services, the surge on demand for processing power is inevitable, and to secure adequate resources in time is a challenge. This is especially true in large and diverse IT organization. The mechanism in this paper gives the business adequate time to plan for Capital Expenditure on the needed hardware for database hosting. From IT management perspective, capital expenditure can happen only during certain time in the fiscal year, hence future requirement of resource need to be predicted accurately approximately 6 months in advance. To build an on-premise Private Cloud infrastructure for database hosting will required quite substantial amount of investment, hence the monetary planning on hardware is required. For instance, the list price for Oracle Exadata Database Machine Quarter Rack is USD 330K [17]. To upgrade from Quarter Rack to Half Rack will required the same amount of investment. With the mechanism proposed here, business can plan in advance the budget allocation and ROI in precise nature.

The model is intended to work during steady state of the database operation. It determines the amount of computational power to support the evolution of an application, so that profit of providing a product or service for current and future can be harvested optimally. To fulfill this objective, the condition of the involved SQL is scrutinized. During the tenure of the product or service offering, some SQL in the database are likely not created or tuned optimally. In situation where the data structure changes, the SQL performance might decline. These SQL are analyzed to ensure no wastage of resource, before the *VM* resource is expanded as per the model's suggestion. In other words, the workload needs to be kept in optimum condition.

We take initiative from workload management proposal by Mateen *et al.* [18]. We agree that characterizing the dynamic workload in a complex application is a challenge that needs to be addressed via autonomous strategies. Workload characterization involves accurate planning during the initial configuration stage, as well as future prediction of the evolved workload. The fast adoption of tools for Autonomic Computing is hence should be of utmost priority for reliable budgetary planning of its IT infrastructure. The autonomic workload management proposed by the authors needs to have self-optimization, self-configuration, self-inspection, self-prediction, self-organization and self-adoption features. We build this model with these criteria for Autonomic Computing sets in perspective.

Our focus is on single-tenant, multiple database instances on-premise Private Cloud

solution. However the proposed resource planning mechanism also works for multi-tenants Cloud offering. The key parameters in this proposal are based on total SQL Elapsed time and the VM's loads. In this case, SQL Elapsed time in the databases serving multi-tenants can be pooled to provide the required input to the system.

With the current and future resource state known in prior, the development team can be made sensitive to the amount of resource utilized by their application, thus measuring the success and profitability of their products or services offering by assessing the resource utilization against the generated revenue. Cloud Computing incorporates Utility Computing in larger view, and this metering mechanism is Utility Computing in micro perspective. This gives the enterprises an option to know what, why and how they spend.

The workload fed into the model is allowed to vary in size and timeframe. Assume all n databases in the *VM* are already running in steady state condition. The n numbers of databases are running 7 days/week, 24 hours/month. One week of continuous data capture for active database is believed to be long enough to capture all SQL running for the application, yet short enough that performance data from the beginning of the timeframe is still relevant. However there are applications that active only during certain time of the year, for instance the Employee Performance Management software that is active only 2-3 times a year, or the Annual Enrolment utility for employees living in United States. This model can be adjusted to run the data capturing mechanism selectively, only to take into consideration the top SQL run during these peak timeframes.

Amid the consequence of incurred overhead, the data collection's frequency, duration and time windows can be adjusted. It is imperative for some background processes to run in the host, for instance backup operations and audit tracing. Data collection can avoid these maintenance timeframes to preserve the accuracy of the model.

Another assumption and important criteria to this model is that the entire application and its architecture are assumed to be optimally structured, and codes are properly written and compiled during the infancy stage of the product. This preferred strategy is detailed by Del Rosso [19], where it is employed by Nokia Research Center to develop and tune its software product family. It is to consider the performance starting from the requirement gathering phase of the application development life cycle, and let the model to work on gradual dynamic change in the system. The typical architectural configuration for a new database is to allocate the VM with a set of resource, and arrive at the initial hardware set via load testing, *i.e.* utilizing HP LoadRunner software [20] to characterize the desired workload. The test is simulated on a set of transactions and the expected concurrent loads. During steady state operation mode, sporadic addition of new transactions often does not warrant another load testing setup. However these can accumulatively consume all the allocated resource until the capacity breakpoint is exceeded, especially when the load is added gradually over time, in contrast to significant surge in amount of transactions which is noticeable. If the transaction volume is expected to increase in tandem with the predicted future business grow, it is imperative to estimate accurately how much more resource to be provisioned to the initial instance.

3. RELATED WORKS

In the context of resource management and allocation, we highlight some of the significant contributions by researchers.

Comellas *et al.* [21] shows how Cloud elasticity advantage can be harnessed by typical web hosting application, by presenting Cloud Hosting Provider (CHP). This elastic web hosting provider can properly react to the dynamic load received by web applications and achieve revenue maximization of the provider by performing an SLA-aware resource management. CHP Considers not only the response time of the web transactions, but also price, penalty & cost associated with the web offering. The proposed architecture focuses on web applications, not scrutinizing on database SQL as discussed in this paper. However it portrays the ultimate goal of what we want to achieve for database transactions.

Iqbal *et al.* [22] utilizes Eucalyptus-based test bed cloud to offer multi-tier Web application owners maximum response time guarantees while minimizing resource utilization. The approach taken is similar to what we proposed, where the initial set of resource is identified, and subsequent resource is dynamically altered to satisfy the dynamic workload needs. This paper takes the focal point where each tier (web, application and database) hits the resource bottleneck before reacting to have resource provisioned in order to meet SLA requirement on response time.

Curino *et al.* [23] introduces Relational Cloud – a transactional 'database-as-a-service' Cloud infrastructure, which has the objective to improve on existing approaches, in term of hosting for multi-tenancy database, scaling out for complex workloads and enhancing security of database hosting in Public Cloud. The component of particular relevant to our topic is the workload-aware approach, where the workloads are partitioned and replicated at a much fine-grained level than existing techniques. The proposed model is named Schism [24] using graph representation to depict the database and workload. The graph-partitioned-based algorithm divides a task into several sub-tasks, assigned to different machines to achieve near-linear elastic scale-out. This approach minimizes the required number of machines while meeting the query performance requirement.

In regulating the supply and demand of computing resources at market equilibrium, Buyya *et al.* [4] proposes architecture for market-oriented allocation of resources within Clouds, and presented a vision for the creation of global Cloud exchange for trading services. The latter is similar to transactions in stocks or commodities exchanges, where the supply and demand determines the providers' and buyers' prices. The objectives to meet during the negotiation strategy between the providers and users are to secure appropriate level of QoS to establish SLAs, and then utilizing appropriate mechanisms and algorithms for allocation of *VM* resources to meet SLAs. Then there is also discussion on risks management associated with violation of the agreed SLAs. An *et al.* [25] employ the same approach, and introduce agent-based automatic negotiation model, where the agents make contracts for resource leases from the Cloud Infrastructure, and at the same time calculate a decommitment monetary penalty to break out from the committed contact. Negotiation strategies of multiple buyers and providers to buy and sell Cloud resources are presented, taking initiative from real world stocks exchanges.

Das *et al.* [26] provide an insight of how scalability can be achieved in multitenant database from another perspective. This paper addresses elasticity, the ability to vary load by controlling the amount of resources, through the database migration technique. The authors conducted experiment in ElasTraS, a database system designed for supporting multitenant cloud applications. Data for a tenant is migrated out from the multitenant database without incurring significant downtime, hence moving data out from existing

VM to remote *VM* for resource re-organization purpose. The live migration approach is similar to the scalability product offered by *EMC Symmetrix VMAX Enhanced Virtual LUN Technology* on Oracle database [27]. This technique provides a different view compared to what we have suggested here, where the database is moved out all together to another more properly scaled *VM*.

Ganapathi *et al.* [28] uses KCCA algorithm [29] in a statistical model to predict the response time of queries, then utilizes the result to predict the resource requirements for Cloud computing applications. The authors experimented their approach on MapReduce jobs, which are jobs modeled by Google [30], parallelized and distributed across many machines to achieve reasonable response time. The proposed model estimates the execution time of these jobs with claimed high prediction accuracy. It uses correlation between the real and estimated execution time to evaluate the effectiveness of their method. Workload prediction is not discussed in our approach, but it is an interesting topic worth exploring in our subsequent works.

4. THE ALGORITHM

In this section, firstly we describe how the hardware resource usage is tracked and monitored, and determine the trigger point when the threshold is breached. Secondly, the accuracy of mechanism is guarded by a policy to ensure only valid and tuned SQL are allowed to be included for the calculation of the resource threshold values.

4.1 Resource Usage Tracking

We define 1 minute as interval to aggregate the total SQL Elapsed time from all databases in the host. 1 minute is deemed appropriate timeframe as the Server Load is only accurate with the snapshot duration is small.

The captured data is stored in an array, A, which is a multi-dimensional in nature. Depending how long the data capture and aggregation will run, there will be p snapshots in the A array, and A is $A = [t_1, t_2, \ldots, t_i, t_p]$. Each array element in A is an array, t_i corresponds to the 1-minute snapshot. It is represented by $t_i = [C_i, S_i, S'_i]$, where

C_i = the Server Load in the *VM* averaged in 1-minute interval
S_i = total Elapsed Time of all SQL in the 1-minute interval

corresponds to C_i. This variable denotes time needed to run all iterations for all SQL in the interval. It considers all variables affecting the SQL's runtime, *i.e.* time required by database engine, server condition, network latency, disk IO *etc.* The elapsed time data can be sourced from stored database repository, *i.e.* Automatic Workload Repository [31] from Oracle RDBMS.

S'_i = total DB time of all SQL in the 1-minute interval corresponds to C_i. This variable represents the duration needed by the database to process all SQL including all their iterations in the same snapshots interval used to define S_i. DB time denotes the time needed only by the database engine to process the queries. These values are expected to be obtained from standard database repository too.

Let's take 1 snapshot, $t_1 = [C_1, S_1, S'_1]$. These 3 values are adequate to depict the re-

source planning mechanism which is shown in section 5. When all the values from t_1 to t_p are obtained, the graphical representation of S and S' versus C can be depicted by linear plots, as presented in section 5 too.

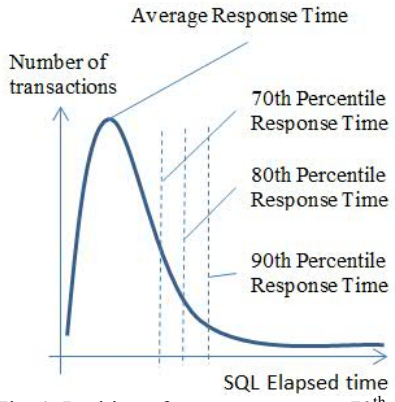

Fig. 1. Position of new parameters – 70th & 80th Percentile Response Time.

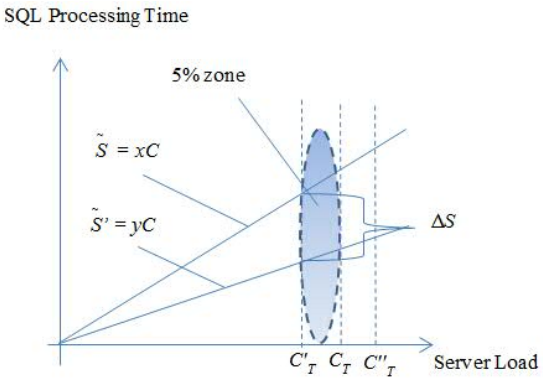

Fig. 2. Parameters obtained from Initial Load Testing.

It is appropriate to explain now, the method to obtain the value of C_T, the threshold value where resource in the host is deemed hitting the maximum utilization. This value is illustrated in Fig. 1. To arrive at C_T, we are taking the minimum Server Load value for each set of transaction to abide with the Service Level Agreement, incorporating a *safety factor*. For particular set of transaction, the *90th percentile response time* [32] is typically recorded in SLA. Hence this is the limit which needs to be complied. During load testing, the ramping up of the load can be used to determine the corresponding Server Load, $C''_{T\text{-}i}$ when the *90th percentile response time* for particular set of transaction is reached. The same is done for all transaction sets, and these values are passed into an array, $C'' = [C''_{T\text{-}1}, C''_{T\text{-}2}, \ldots, C''_{T\text{-}i}, \ldots, C''_{T\text{-}r}]$. With this, $C''_T = min(array\ C'')$.We define a new parameter, the *80th percentile response time* as depicted in Fig. 1, and corresponding Server Load, $C_{T\text{-}i}$ for each transaction set. So $C = [C_{T\text{-}1}, C_{T\text{-}2}, C_{T\text{-}i}, \ldots, C_{T\text{-}r}]$, and $C_T = min(array\ C)$. We choose C_T as the Server Load threshold, so that there is still time to react for hardware planning and provisioning. Then we define *70th percentile response time* in similar fashion, and obtained $C'_T = min(array\ C')$. We define that when 5% of transactions accumulate between C'_T & C_T, resource provisioning mechanism is triggered, and another block of hardware needs to be added to the *VM*. This is shown in Fig. 2. This 5% zone is adjustable, as explained in a bit. Any tuned transactions that go beyond C_T is almost not in compliance with the SLA and emergency action is needed to investigate the state of the host, and if there is no abnormality found, the hosting hardware needs to be enlarged immediately.

As experimented by Ferrari and Zhou [33] and Gunther [34], the correlation between Server Load and server processing time is linear in nature, provided that the server is not running into resource constraint. Hence to arrive at the 2 lines mathematically, we employed the *Linear Regression* methodology [35]. Take Elapsed Time as example, for each data point, it is defined as $S_i = xC_i + b + \varepsilon_i$. However in our case, value b is as-

sumed 0 as observed from the actual host itself that even with multiple background daemons running, the server load is close to 0 and these variables is negligible. We can safely assume this value as in our case, as the server resources are abundant. However if the hardware resource in the server or *VM* is restricted, for instance if there is only a single CPU and 1 GB of physical memory, value of b could be 1 or 2 as a result of the system overhead, and it needs to be acknowledged in the formula. In this case C'_T is sensitive to the constantly-running overhead processes in the server or *VM*. Nevertheless, this will only complicate the formulation in the algorithm, without achieving significant desired result. The explained model will work properly in the environment with larger resource, to curtail the system overhead; however if the overhead is large and unavoidable, a workload pre-check mechanism can be incorporated, to ensure a robust workload control plan is in place. The objective of mechanism is to avoid and disregard timeframe when the system is running non-database related overheads before inputting the workload data into the model. In defining the legitimate workloads for input to the model, we assume IT organization has a well-designed maintenance window to cater for unavoidable system overhead, especially the backup operations, where business transactions during this timeframe are kept to minimum.

To explain our case, $S_i = xC_i + \varepsilon_i$. The linearly fitted value, $\tilde{S}_i$ is the value fitted exactly on the regression line, and is denoted as $\tilde{S}_i = xC_i$. Hence, the residuals, $\varepsilon_i = S_i - \tilde{S}_i$, are the differences between the actual and fitted values of Elapsed Time. This variable is not elaborated for our discussion in this paper, but will serve as a critical component for our subsequent work in developing an adaptive system to reduce the noises in the system. We need only to calculate the value of x to fulfill our requirement here. Using *Least Squares Derivation* method, with N number of data points, the value of x is obtained as

$$x = \frac{\sum_{i=1}^{N} C_i \tilde{S}_i - \frac{\sum_{i=1}^{N} C_i \sum_{i=1}^{N} \tilde{S}_i}{N}}{\sum_{i=1}^{N} C_i^2 - \frac{(\sum_{i=1}^{N} C_i)^2}{N}}. \tag{1}$$

With this, the regression line is plotted using $\tilde{S} = xC$, and similarly for DB time, $\tilde{S}' = yC$. To measure the representability of the regression lines to the data points, we use the correlation coefficient (r), defined as

$$r = \frac{\frac{\sum_{i}^{N} (C_i - \bar{C})(\tilde{S}_i - \bar{\tilde{S}})}{N}}{\sqrt{\frac{\sum_{i}^{N} (\tilde{S}_i - \bar{\tilde{S}})^2}{N}} \sqrt{\frac{\sum_{i}^{N} (C_i - \bar{C})^2}{N}}}. \tag{2}$$

r is confined to value between 0 and 1 in our case. 1 denotes that there is a perfect linear correlation between C and $\tilde{S}'$, while 0 shows no correlation. Intermediate values show partial correlations. We will use value of r in section 5 to gauge the accuracy of our experiment.

We also define another parameter $\Delta S = (\tilde{S} - \tilde{S}')/\tilde{S}' * 100$. ΔS corresponds to C'_T. This delta of $\tilde{S}$ & $\tilde{S}'$ can be used to gauge if the host condition is still viable for optimal database transactions. During steady state database operations, if $\tilde{S} = xC$ becomes steeper, ΔS

is then reached for Server Load < C'_T. The reason could due to the fact that the physical reads or logical reads in the database are not efficient. This directly points to either the IO subsystem is not functioning optimally or the database cache is not sufficient. The noises from the operating system can also contribute to this, for instance new auditing daemon is running on the host, additional host monitoring utility is running *etc.* The noises ideally are undesired, for a mission critical application running stead-state operations. However in real live environment, system overhead is inevitable. For instance, the *VM* and database backups will cause significant overhead and these cannot be ignored. In this case a maintenance window is defined, and the workload input will avoid this timeframe when feeding into the algorithm, to preserve the model accuracy.

The new ΔS needs to be shrunk by taking appropriate measures, *i.e.* fixing the host environment or increase database cache. If all has been done but value of x is still steeper than before, a new C'_T will need to be defined. In this case C'_T is the Server Load value corresponds to ΔS. It is to note that C_T & C''_T stay as constant, hence the C'_T & C_T gap is enlarged. When this happens, the probability of transactions to fall into the 5% zone increases. The 5% zone is a hypothesis figure, and it should be adjusted appropriately based on particular application's SLA.

When block of new hardware is added to existing *VM*, Load Testing is conducted again to find out values of x, y, C_T, C'_T & C''_T, and subsequently ΔS. With these values, this resource tracking model is reconfigured to watch for subsequent need of hardware provisioning. In this case, as the base *VM* is already running stead-state production, the scrambled data can be pushed to Public Cloud with the same hardware configuration so that Load Testing can be conducted without jeopardizing the SLA requirement of existing databases. The advantage of hosting the databases in the *VM* in this case, is that the same configuration of hardware can be provisioned easily in the Public Cloud, hence making the Load Testing relevant.

4.2 SQL Tracking

During the tenure of the database life cycle, it is imperative to keep the DB time of all transactions as close as possible to the initial Load Testing. In other words, the line $\tilde{S}' = yC$ as in Fig. 2 should not change ideally. To do this we need to ensure that the SQL are running as optimally as possible during the initial load test. To explain this SQL verification mechanism, define another array, B, which has 30-minute interval in each of its element. Depending on how long the data capture operation is going to run to properly represent all potential SQL in the databases, there is q samples in B, $B = [u_1, u_2, \ldots, u_i, \ldots, u_q]$. Then there are n numbers of databases running in the host, $DB = [db_1, db_2, \ldots, db_j, \ldots, db_n]$. Take 1 30-minute snapshot, u_1 to represent activity in other snapshots, and define s_1 = collection of SQL elapsed time of top m number of SQL in all n databases, running in u_1. The top m SQL is ranked in descending order by total elapsed time. s_1 is a collection of SQL ID.

Hence, s_1 is $s_1 = [ss_1, ss_2, ss_3, \ldots, ss_k, \ldots, ss_m]$, where ss_k = SQL elapsed time on a SQL k that runs in database db_j with y iterations in the 30-minute interval, defined as $\sum_{i=1}^{y} \binom{y}{\ell} \mu_k$, where μ_k = mean elapsed time of the SQL.

Top m SQL is defined as SQL that exceeds x duration of runtime including all its iteration in u_1. Top m SQL are dominant resource consumer in the host.

The accurate way to gauge the effectiveness of a SQL is to compare its actual μ with the benchmarked value, in this case the minimum of μ found in all the legitimate data points in Fig. 1. So for ss_k, we label the minimum of μ as $\mu_{k\text{-min}}$. For each SQL ID in s_i, the minimum μ is stored in an array, $U = [\mu_{1\text{-min}}, \mu_{2\text{-min}}, \mu_{3\text{-min}}, \ldots, \mu_{k\text{-min}}, \ldots, \mu_{m\text{-min}}]$. Hence, data in U is to be benchmarked when SQL tuning is taking place.

5. EXPERIMENTAL RESULTS

We do not have the liberty to a mission-critical application with enough SQL transactions that is hosted on Cloud instance for illustration. We show the concept of our model base on a single server, which can be proliferated to application on Cloud *VM* without much variation. The data for the experiment is gathered from a Sun Solaris server, powered by 4 Sun Solaris SPARC64-VII CPU with 4-core architecture, 64GB RAM and external SAN running on ZFS File system. The application runs on SAP ERP software, on ECC6 HRM Module. The application is OLTP in nature, servicing Human Resource Management System for a large organization.

The backend is running on a single instance Oracle 11g database. In this scenario, a week of data on actual transactions is collected, with Server Load taken as average in 1-minute interval. SQL Elapsed and DB time are collected in tandem with the 1-minute interval timeframe. In this case we simplify the model by having only 1 database running in the host.

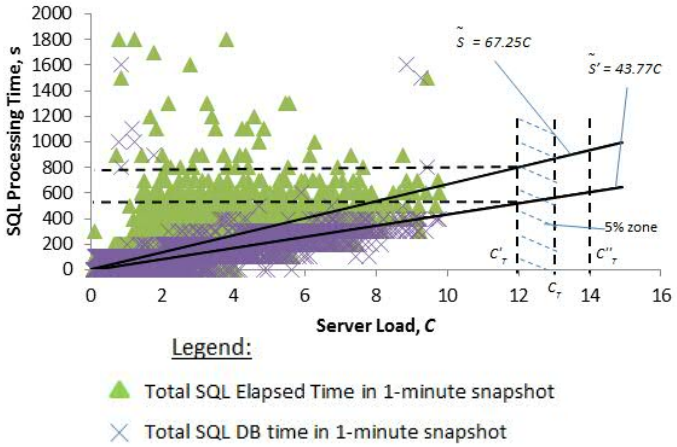

Fig. 3. Experimental results that show relationship between $\tilde{S}_i$, $\tilde{S}_i'$ and C_i.

All 3 variables – Server Load (C), Elapsed time ($\tilde{S}$) and DB time ($\tilde{S}'$) are quantitative. When the data points of these 3 values are mapped, a scatter graph is generated as in Fig. 1. As the correlation between C and $\tilde{S}$, as well as C and $\tilde{S}'$ are linear as explained in section 4.1, regression lines are drawn mathematically. They are interpolated on the cluster of scatter plot data, to statistically describe the trend of the SQL Elapsed and DB time. As mentioned in section 4.1, the slope of the regression line can be obtained from Eq. (1).

There are a total of 10621 samples (collection of the data points in Fig. 3) gathered in the 1 week period for each set of Elapsed Time, $\widetilde{S}$ and DB time, $\widetilde{S}'$ As mentioned the relationship are $\widetilde{S} = xC$ and $\widetilde{S}' = yC$ respectively, and calculated using the data points' values we have got $x = 67.25$ and $y = 43.77$. Hence as seen in Fig. 3, 2 strong positive regression lines are drawn. There are outliers in the graph, and they are understandably to be caused by noises in the server outside the control of the RDBMS. These can possibly cause by the auditing processes in the Operating System which spike occasionally while the application transactions are running, File System backup that incurs IO contention and monitoring daemon to name a few. To gauge the accuracy of the regression lines, we use the correlation coefficient, r, as defined in section 4.1. Using Eq. (2), we obtain $r = 0.72$ for the regression line on Elapsed Time, and $r = 0.78$ for the regression line on DB Time. We think that these 2 values show that the fit of the 2 linear models is fairly acceptable. In other words we can assume that these noises are not affecting the correlations too much. For more accurate plots, these noises will need to be investigated and fixed at OS level, or if ever desired, the outliers in Fig. 3 can be excluded to increase the accuracy of the regression lines. With these equations, the limit when the server is hitting resource constraint can be further derived.

2 values from initial Load testing during pre-cutover are to be noted, before the database goes into steady-state production mode. They are C'_T & ΔS. The initial C_T is set at 13 and $\Delta S = 55\%$ respectively. After about a year running into steady-state production mode, the C_T value reduced to 12 with ΔS stays at 55%, as depicted in Fig. 3.

To represent these 2 properties properly, Fuzzy Logic is employed, as illustrated below:

Step 1: Determine when to examine the host environment and adjust C_T, using *Fuzzy Computing with Words* [36]:

If *SQL* elapsed time is very much higher than *SQL DB* time, the host environment is near suboptimal condition.

With *Fuzzy Implication method* [37]:

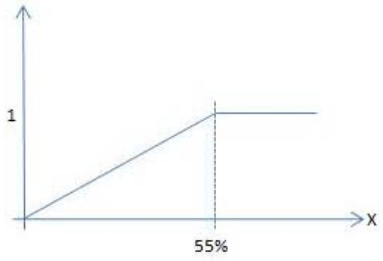

Fig. 4. Membership Function for ΔS, $A(u)$.

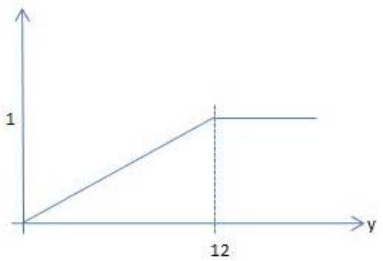

Fig. 5. Membership Function for C, $B(v)$.

Fig. 4 shows membership function for ΔS, $A(u)$. $A(u)$ is

$$A(u) = \begin{cases} 1 & \text{if } u \geq 55 \\ \dfrac{u}{55} & \text{if } 0 \leq u \leq 55. \\ 0 & \text{if otherwise} \end{cases} \tag{3}$$

With $u = 55$, corresponding Server Load limit, C'_T is obtained, as in Fig. 2.

Fig. 5 shows membership function for Server Load, $B(v)$ in the server. $B(v)$ is

$$B(v) = \begin{cases} 1 & \text{if } v \geq 12 \\ \dfrac{v}{12} & \text{if } 0 \leq v \leq 12\,. \\ 0 & \text{if otherwise} \end{cases} \tag{4}$$

Subsequently, the constraining relation, $R = A(u) \Rightarrow B(v)$.

Step 2: Determine if database transactions need additional hardware:

If more data points fall between C'_T & C_T, trigger point for hardware planning and provisioning is near.

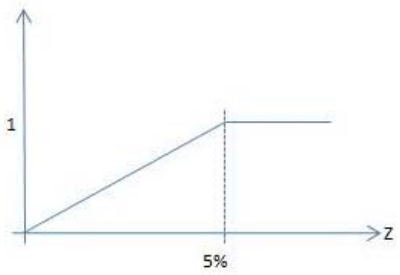

Fig. 6. Membership Function for ρ, $C(w)$.

Fig. 6 shows membership function for density of data points, ρ, between C'_T & C_T.

$$C(w) = \begin{cases} 1 & \text{if } w \geq 5 \\ \dfrac{w}{5} & \text{if } 0 \leq w \leq 5\ . \\ 0 & \text{if otherwise} \end{cases} \tag{5}$$

The membership function $A(u)$ is obtained from initial Load Testing as when ΔS is defined. Subsequently from the limit of $A(u)$, $B(v)$ is determined during steady-state operations. Using limit of $B(v)$, $C(w)$ is derived.

6. MODEL ACCURACY AND SYSTEM OVERHEAD

The snapshot interval to obtain the data points is set to 1 minute in the experiment. It is worthy to note that the smaller the interval, the more accurate the data is. Caution

needs to be taken here on the workload to collect $\tilde{S}$, $\tilde{S}'$ and C values, as these data computation in the database should not incur too much overhead. In system with not as powerful hardware, 1-minute interval could incur high overhead to the host environment. In contrast when there is more resource available in the host, sampling interval can be small. This can be represented by Fuzzy rule in the form of:

R: *If* <*x is P*>, *then* <*y is Q*>. This is translated to *If* <*Server load is low*>, *then* <*sampling granularity is small*>

Then with Fuzzy predicates *P* & *Q* as Fuzzy sets on *U* = *domain of x*, *V* = *domain of y*, define

P(*x*) for '*x is P*' and *Q*(*y*) for '*y is Q*', and define,
$T(R) = T[P(x) \Rightarrow Q(y)]$ for every *x* in *U* and every *y* in *V*.
Using *Mamdani implication* [38] which is appropriate in this case,
$T[P(x) \Rightarrow Q(y)[= min[P(x), Q(y)]$

With this, the appropriate overhead values of *P* (Server Load) and *Q* (sampling granularity) can be coded.

7. SQL OPTIMIZATION

In real situation, there are un-optimized transactions that disguise the actual need of computational power. SQL must not be allowed to run wildly. Following explains situations on the behavior of these SQL.

For the same SQL which runs multiple iterations, either via bind variables or literal values, its μ, which is the average execution elapsed time, may change but the execution plan stays the same. Few scenarios could lead to this, for instance if the data involved in the SQL increased significantly and statistic has not been gathered in time, or if there is skewed histogram in the data resulted from data change. Another situation that can lead to this is when there is high resource contention in the *VM* going beyond *Server Load Threshold*. These are represented in Fig. 7.

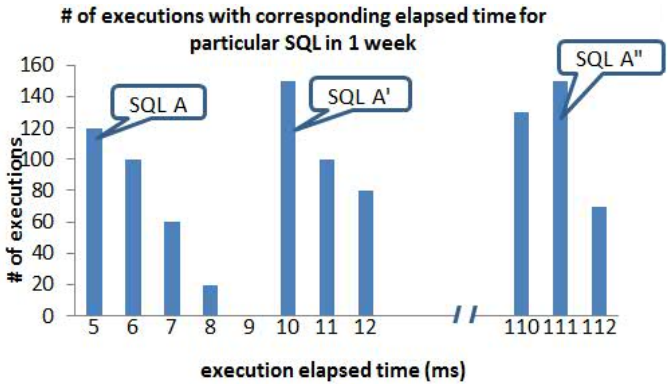

Fig. 7. Runtime variation of particular SQL in 1 week.

Fig. 7 shows Elapsed Time of a SQL executed in each 30-minute segment in *B* array. There are 10 occurrences of the SQL execution in 1 week. *SQL A* was the original statment, optimally tuned, and there is no hardware contention in the server. To explain this further, there are few key aspects to define optimally tuned statement in this case. During end users acceptance test, the buy-off transaction response time could be set as benchmark. Then when the database is running in steady-state production mode, the RDBMS engine can self-tune the SQL, as with Oracle 11g Automatic SQL Tuning [39] which is a refinement from SQL Tuning Advisor [40]. There is also case where particular SQL is intentionally forced to run on particular execution path to maintain desired response time.

A' is a result of data being added to the tables involved, and it goes undetected by the RDBMS. A'' illustrates the scenario when data is added significantly to the tables used by the queries before tables' statistic is gathered, or necessary indexes have not been considered and worked on with new data. In another scenario there can be resource contention in the *VM* resulted in A''. Another scenario which is not shown in Fig. 7 is that *SQL A* changes its execution plan, as a result of accidental drop of an index in a table or sudden surge of cache memory consumption due to sudden increment of table data. These adversely result in excessive physical reads and the μ diverges significantly.

Above are a few situations that affect the accuracy of the model. These SQL need to be tuned before resource capacity tracking model can report the resource utilization state accurately. As defined earlier, $U = [\mu_{1\text{-}min}, \mu_{2\text{-}min}, \mu_{3\text{-}min}, \ldots, \mu_{k\text{-}min}, \ldots, \mu_{m\text{-}min}]$. The data in this array is used for the purpose of benchmarking and tuning the involved SQL as described in section 4.2.

8. SUMMARY

Cloud Computing is inevitably going to be the hosting trend for the future. At this juncture, we think on-premise Private Cloud will dominate database hosting for most Enterprises in the near future. The proposed resource management model is useful and important for this type of Cloud Instance, where computing resource allocation needs to be planned in advance. It provides a visibility to plan, procure and allocate the needed resource, along with the activities that happen in the host itself. The scrutiny on the SQL ensures the model correctly delivers its promise. The introduced Fuzzy functions are hoped to aid in coding stage of the model. As the model is constructed from detailed collection of SQL execution time versus server loads, the data points are able to accurately depict the real processing scenario in the host.

9. FUTURE WORKS

As information of the SQL is stored in a repository as proposed in the model, we want to take this advantage to evaluate and investigate if these SQL can be good candidate for further tuning and enhancement. We will look at potential materialization of the SQL, so that the physical or logical reads on tables or views can be reduced. Another objective is to increase the chance for execution plan reuse in the database as materialization can reduce queries' footprint in database cache. Along the way researching on materials for this paper, we also realize that Artificial Neural Network could be an inter-

esting area to venture into, utilizing the learning mechanism to reduce noises in the system.

REFERENCES

1. S. Upson, "Cloud computing – It's always sunny in the cloud," *IEEE*, http://spectrum.ieee.org/static/special-report-top-11-technologies-of-the-decade, 2011.
2. M. Glas and P. Andres, *Achieving the Cloud Computing Vision*, Oracle Corporation, 2011.
3. S. Zhang, S. F. Zhang, X. B. Chen, and X. Z. Huo, "The comparison between cloud computing and grid computing," *Computer Application and System Modeling*, *International Conference on Computer Application and System Modeling*, Vol. 11, 2010, pp. 72-75.
4. R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Journal of Future Generation Computer Systems*, Vol. 25, 2009, pp. 599-616.
5. I. Foster, Y. Zhao, I. Raicu, and S. Y. Lu, "Cloud computing and grid computing 360-degree compared," in *Proceedings of Grid Computing Environments Workshop*, 2008, pp. 1-10.
6. Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, Vol. 1, 2010, pp. 7-18.
7. R. King, *Bloomberg BusinessWeek*, http://www.businessweek.com/technology/content/aug2008/tc2008082_445669.htm, 2008.
8. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds – A Berkeley view of cloud computing," UC Berkeley Reliable Adaptive Distributed Systems Laboratory, 2009.
9. Microsoft, "SQL azure – Moving business intelligence to the cloud," *Wipro Technologies*, 2011.
10. Saugatuck Technology, *Platform-as-a-Service: What ISVs Need for World-Class Cloud Offerings*, IBM, 2011.
11. R. Harms and M. Yamartino, *The Economics of the Cloud*, Microsoft, 2010.
12. Amazon, *Amazon Web Services: Risk and Compliance*, Amazon.com Inc., 2012.
13. Google, *Security Whitepaper: Google Apps Messaging and Collaboration Products*, Google Inc., 2010.
14. Oracle – Exalogic, *Oracle Exalogic Elastic Cloud: A Brief Introduction*, Oracle Corporation, 2011.
15. I. Raicu, Y. Zhao, I. Foster, and A. Szalay, "Accelerating largescale data exploration through data diffusion," in *Proceedings of International Workshop on Data-Aware Distributed Computing*, 2008, pp. 9-18.
16. X. Dutreilh, N. Rivierre, A. Moreau, and J. Malenfant, "From data center resource allocation to control theory and back," in *Proceedings of the IEEE 3rd International Conference on Cloud Computing*, 2010, pp. 410-417.

17. Oracle – Price, *Oracle Engineered Systems Price List*, Oracle Corporation, 2012.
18. A. Mateen, B. Raza, M. Sher, M. M. Awais, and N. Mustapha, "Workload management: A technology perspective with respect to self characteristics," *International Journal of Physical Sciences*, Vol. 7, 2011, pp. 1482-1492.
19. C. D. Rosso, "The process of and the lessons learned from performance tuning of a product family software architecture for mobile phones," in *Proceedings of the 8th Euromicro Working Conference on Software Maintenance and Reengineering*, 2004, pp. 270.
20. HP, *Application Performance Testing in VMware Environments – Identify and Control Performance and Capacity Risks*, Hewlett-Packard Development Company, 2007.
21. J. O. F. Comellas, I. G. Presa, and J. G. Fernández, "SLA-driven elastic cloud hosting provider," in *Proceedings of the 18th Euromicro Conference on Parallel*, *Distributed and Network-based Processing*, 2010, pp. 111-118.
22. W. Iqbal, M. N. Dailey, and D. Carrera, "SLA-driven dynamic resource management for multi-tier web applications in a cloud," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster*, *Cloud and Grid Computing*, 2010, pp. 832-837.
23. C. Curino, E. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich, "Relational cloud: A database-as-a-service for the cloud," in *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research*, 2011, pp. 235-240.
24. C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: a workload driven approach to database replication and partitioning," *Journal of VLDB Endowment*, Vol. 3, 2010, pp. 48-57.
25. B. An, V. Lesser, D. Irwin, and M. Zink, "Automated negotiation with decommitment for dynamic resource allocation in cloud computing," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, Vol. 1, 2010, pp. 981-988.
26. S. Das, S. Nishimura, D. Agrawal, and A. E. Abbadi, "Live database migration for elasticity in a multitenant database for cloud platforms," UCSB Computer Science, Technical Report No. 2010-09, 2010.
27. "EMC symmetrix VMAX using EMC SRDF/timefinder and oracle database 10g/11g," *EMC*, 2011.
28. A. Ganapathi, Y. P. Chen, A. Fox, R. Katz, and D. Patterson, "Statistics-driven workload modeling for the cloud," in *Proceedings of International Conference on Data Engineering Workshop*, 2010, pp. 87-92.
29. A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. Jordan, and D. Patterson, "Predicting multiple performance metrics for queries: Better decisions enabled by machine learning," in *Proceedings of IEEE International Conference on Data Engineering*, 2009, pp. 592-603.
30. J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, Vol. 51, 2008, pp. 107-113.
31. B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin, "Automatic SQL tuning in oracle 10g," in *Proceedings of the 30th International Conference on Very Large Databases*, Vol. 30, 2004, pp. 1098-1109.
32. TPC BENCHMARK™ C, "Standard specification," Revision 5.11, 2010.
33. D. Ferrari and S. N. Zhou, "An empirical investigation of load indices for load bal-

ancing applications," in *Proceedings of the 12th IFIP WG 7.3 International Symposium on Computer Performance Modelling*, *Measurement and Evaluation*, 1987, pp. 515-528.
34. N. J. Gunther, *UNIX Load Average Part 1*, TeamQuest Corporation, 2010.
35. J. C. Principe, N. R. Euliano, and W. C. Lefebvre, *Neural and Adaptive Systems*, John Wiley & Sons, Inc., NY, 2000, pp. 7-17.
36. L. A. Zadeh, "Fuzzy logic and computing with words," *IEEE Transactions on Fuzzy Systems*, Vol. 4, 1996, pp. 103-111.
37. C. R. Alavala, *Fuzzy Logic and Neural Networks*, New Age International (P) Ltd., New Delhi, India, 2008, pp. 30-33.
38. M. Ganesh, *Introduction to Fuzzy Sets and Fuzzy Logic*, Prentice Hall Inc., Upper Saddle River, NJ, 2008, pp. 151-156.
39. P. Belknap, B. Dageville, K. Dias, and K. Yagoub, "Self-tuning for SQL performance in oracle database 11g," in *Proceedings of IEEE International Conference on Data Engineering*, 2009, pp. 1694-1700.
40. D. D. Li, L. Han, and Y. Ding, "SQL query optimization methods of relational database system," in *Proceedings of the 2nd International Conference on Computer Engineering and Applications*, Vol. 1, 2010, pp. 557-560.

Chee-Heng Tan received his B.Eng. and M.Sc. from University Science of Malaysia. Currently he is a Ph.D. student at Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. His current research interests include database automation and performance, cloud computing, fuzzy logic and statistical regression analysis.

Ying-Wah Teh received his B.Sc. and M.Sc. from Oklahoma City University and Ph.D. from University of Malaya. He is currently a Senior Lecturer at Information Science Department, faculty of Computer Science and Information Technology, University of Malaya. His research interests include data mining, text mining and document mining.