Egyptian Informatics Journal (2014) **15**, 1–12

Cairo University

Egyptian Informatics Journal

www.elsevier.com/locate/eij
www.sciencedirect.com

ORIGINAL ARTICLE

Optimum Resource Allocation of Database in Cloud Computing

Fatma A. Omara *, Sherif M. Khattab, Radhya Sahal

Faculty of Computers and Information, Cairo University, Egypt

Received 19 January 2013; revised 7 January 2014; accepted 14 January 2014
Available online 14 February 2014

KEYWORDS

Virtualization;
Resource allocation;
PSO;
Query optimizer;
Calibration

Abstract Cloud computing is a new generation of computing based on virtualization technology. An important application on the cloud is the Database Management Systems (DBMSs). The work in this paper concerns about the Virtual Design Advisor (VDA). The VDA is considered a solution for the problem of optimizing the performance of DBMS instances running on virtual machines that share a common physical machine pool. It needs to calibrate the tuning parameters of the DBMS's query optimizer in order to operate in a what-if mode to accurately and quickly estimate the cost of database workloads running in virtual machines with varying resource allocation.

The calibration process in the VDA had been done manually. This manual calibration process is considered a complex, time-consuming task because each time a DBMS has to run on a different server infrastructure or to replace with another on the same server, the calibration process potentially has to be repeated. According to the work in this paper, an Automatic Calibration Tool (ACT) has been introduced to automate the calibration process.

Also, a Greedy Particle Swarm Optimization (GPSO) search algorithm has been proposed and implemented in the VDA instead of the existed greedy algorithm to prevent the local optimum states from trapping the search process from reaching global optima. The main function of this algorithm is to minimize the estimated cost and enhance the VMs configurations.

The ACT tool and the GPSO search algorithm have been implemented and evaluated using TPC-H benchmark queries against PostgreSQL instances hosted in Virtual Machines (VMs) on the Xen virtualization environment.

* Corresponding author. Tel.: +20 22734030.
E-mail addresses: f.omara@fci-cu.edu.eg (F.A. Omara), s.khattab@fci-cu.edu.eg (S.M. Khattab), radhya.sahal@grad.fci-cu.edu.eg (R. Sahal).

Peer review under responsibility of Faculty of Computers and Information, Cairo University.

Production and hosting by Elsevier

1. Introduction

Cloud computing is a new generation of computing. It allows users to use computational resources and services of data centers (i.e., machines, network, storage, operating systems, application development environments, application programs) over the network to deploy and develop their applications [1]. The main feature of cloud computing is providing self-service pro-

1110-8665 © 2014 Production and hosting by Elsevier B.V. on behalf of Faculty of Computers and Information, Cairo University.
http://dx.doi.org/10.1016/j.eij.2014.01.002

visioning, which allows the users to deploy their own sets of computing resources [2]. The cloud computing technology is based on virtualization. Virtualization is a technology that separates computation functions from physical hardware. It allows the users to partition and multiplex physical machine infrastructure (e.g., CPU, memory, I/O, storage, and network interface cards) [3]. The applications are running on virtual machines instead of physical ones. The **V**irtual **M**achine (VM) is a software implementation of a computing environment to simulate a physical machine directly executing on physical hardware [4]. The **V**irtual **M**achine **M**onitor (VMM) is used to create and manage the VMs (e.g., Xen, VMware, VirtualBox, and KVM) [5]. The virtual machine configuration or resource allocation controls the sharing of physical resources (CPU, memory, I/O bandwidth) allocated to VMs. The problem of optimizing the performance of the virtualized applications (i.e., the applications that run on VMs) is critical to the success of the cloud computing paradigm, because VM configuration affects the application performance [2,6].

On the other hand, The Database Management System (DBMS) is considered one of the applications deployed on the cloud. Each DBMS instance has its own tuning parameters. The tuning parameters interact with cost model in DBMS' query optimizer to change the performance (e.g., CPU parameters and buffer parameters) [7]. DBMS needs to calibrate its tuning parameters in order to be aware of virtualized environment and produce an accurate estimated cost. Indeed, DBMS faces a challenge of tuning resource allocation because each workload (a set of SQL statements) has different characteristics and needs different resource allocation. In other words, how DBMS instances can get a benefit of resource allocation for each VM in the shared physical pool, this called Virtualization Design Problem (VDP) [7–9]. Virtual Design Advisor (VDA) is a technique that offers a solution for such problem. It gives recommended configurations for multiple VMs running different workloads among shared resources [2,7–9]. It explores the characteristics of workloads to distinguish their intensity (e.g., CPU or I/O intensive, etc.) and makes a decision for best resource allocation for VM which run this workload. The DBMS has a query optimizer tool to choose the best execution plan based on the estimated cost. The cost model is a module in the query optimizer tool which is responsible for the cost estimation. Database cost model expresses the total resources consumption for a given workload. It depends on static assumptions for tuning parameters to generate the execution plan. In fact, the accuracy of the execution of the current resources consumption is considered a problem for database's cost model.

In other words, the query optimizer's cost model is not aware of virtualized environment because it takes the default values of tuning parameters. So, the query optimizer parameters are needed to be calibrated in order to be aware of different resource allocation in virtualized environment. Each time, the DBMS instance moves from one infrastructure to another, or the DBMS instance is replaced by another DBMS instance in the same infrastructure, the calibration process is repeated. Unfortunately, this process had been executed manually. So, the calibration process is needed to be automated in order to save time, money and produce an accurate estimated cost. In this paper, an **A**utomatic **C**alibration **T**ool (ACT) has been introduced to tune parameters of DBMS query optimizer in virtualized environment to solve the manual calibration problem in the VDA.

On the other hand, a **P**article **S**warm **O**ptimization (PSO) is considered a modern evolutionary algorithm which is used to explore the search space of a given problem [10]. It is used to find optimal or near-optimal solutions for maximization/minimization search problems. As stated previously, the VDP is considered a search problem which tries to minimize the allocation cost of virtualized resources for database systems in cloud environment [2,7–9]. In this paper, a search algorithm called **G**reedy **P**article **S**warm **O**ptimization (GPSO) has been proposed to overcome the local optimum problem of the existed greedy algorithm in the VDA. The proposed GPSO algorithm is considered an amalgamation of heuristic greedy search and particle swarm optimization to optimize configurations based on the workload profile in virtualized environments. The GPSO algorithm has been implemented in the VDA enumerator module, which initially makes an equal resource allocation of VMs and adapts these allocations based on the estimated cost obtained by cost models of the database system query optimizer.

To evaluate the ACT tool and the GPSO search algorithm, prototype experiments have been conducted based on the optimal CPU allocation for the different virtual machines. Tests have been performed using PostgreSQL 8.4.8, running TPC-H benchmark queries as workloads [11,12]. The experimental results show that the ACT runtime increases linearly with the number of calibration sampling points, and the GPSO algorithm can provide effective configurations for different types of workloads than that the existed greedy algorithm.

The rest of this paper is organized as follow; the related works are described in Section 2. The calibration problem in the VDA is described in Section 3. The proposed automatic calibration tool for DBMS query optimizer is discussed in Section 4. In Section 5, the optimization problem in the VDA will be handled. In Section 6, the proposed GPSO algorithm will be discussed. In Section 7, the ACT and the GPSO algorithm evaluation results are introduced. In Section 8, the paper is concluded; also a brief outlook into the future work is given.

2. Related work

There are many research papers in the field of performance optimization of applications running in virtualized environments [8,9,13], and resource allocation [14,15]. A related problem to the work of this paper is the virtualization design problem which addresses the question of how to optimally (with respect to application throughput) partition the resources of a physical machine over a number of VMs, each running a potentially different database appliance (i.e., preconfigured DBMS and a set of workload queries) [7–9]. In [8,9], the virtual design advisor has been presented to solve the virtualization design problem by using the query optimizer, which is typically built-in in most DBMSs, as a cost model to evaluate potential resource partitioning configurations. This "*what-if*" usage of the query optimizer has also been used in non-virtualized environments to justify upgrades of resources based on the predictions of the expected improvement in workload performance [16,17]. In [2], the virtual design advisor has been used to optimize the performance of database appliances that had been deployed in the Amazon EC2 cloud. Ideally, the

performance of the calibration process is an important task in many performance optimization problems [8,9,18,19]. When the calibration process is tedious, its automation becomes of benefit to the overall optimization framework.

The virtual design advisor employs a white-box approach for modeling the performance of the DBMS [8,9]. On the other hand, the black-box approach for performance modeling has been used in [13] to drive an adaptive resource control system that dynamically adjusts the resource share of each tier of a multi-tier application within a virtualized data center. The two approaches; black-box and white-box have been used to solve the resources provisioning problem for DBMS on the top of IaaS cloud [20].

Soundararajan et al. [15] have considered the storage resource in addition the CPU and memory resources. They found that the resource configuration would affect the performance which is considered a challenge in the resource allocation problem. The resource allocation problem is a classical problem that gets instantiated with emerging resource consolidation settings, such as machine virtualization and cognitive radio networks [14]. In the latter, radio spectrum is shared among cognitive radios, and the resource allocation problem is formulated as an optimization problem to achieve max–min rate sharing among the users.

Recently, resource allocation is one of the most important challenges in the cloud computing technology sector that face the cloud provider regardless of the hierarchy of services. Especially, how the cloud provider can meet the clients' **S**ervice **L**evel **A**greements (SLAs) and maximize total profit. In [21,22], the SLA-based resource allocation problem for multi-tier cloud applications is considered for a distributed solution for each processing power, data storage, and communication resources. The problem is cast as a three-dimensional optimizations problem. Also, the cost-performance tradeoff in cloud IaaS has been addressed, where the problem has been formulated as a multi-objective optimization [23]. The proposed model was built based on a fine grained charging model and a normalized performance model. The implementation using genetic algorithms and the experimental results have proved the effectiveness of the proposed model.

On the other hand, there is a wealth of existing proposed approaches using a Particle Swarm Optimization (PSO) in various domains in general and in dynamic environments in particular. The basic PSO is as an optimization technique for static environments [10]. In the real world, however, many applications are non-stationary optimization problems; they are dynamic, meaning that the environment and the characteristics of the global optimum can change timely. Several successful PSO algorithms have been developed for dynamic environments. One of these algorithms is fast multi-swarm optimization algorithm (FMSO) [24]. It uses two types of swarm; one to detect the promising area in the whole search space and the other swarm is used as a local search method to find the near-optimal solutions in a local promising region in the search space. Another approach is used to adapt PSO in dynamic environments [25]. It is based on tracking the change of the goal periodically. This tracking is used to reset the particle memories to the current positions allowing the swarm to track a changing goal with minimum overhead [25]. Cooperative Particle Swarm Optimizer (CPSO) has been introduced for employing cooperative behavior to significantly improve the performance of the original PSO algorithm [26]. This is achieved by using multiple swarms to optimize different components of the solution vector cooperatively. While the original PSO uses a population of D-dimensional vectors, CPSO partitions these vectors into D swarms of one-dimensional vectors, each swarm representing a dimension of the original problem.

According to the work in this paper, an algorithm, called **G**reedy **P**article **S**warm **O**ptimization (GPSO) has been proposed to optimize the allocation of shared resources to minimize the estimated cost and enhance VM configuration.

3. Calibration problem in virtual design advisor

The Virtualization Design Problem (VDP), the Virtual Design Advisor (VDA) solution, and the calibration problem in VDA will be discussed.

3.1. Virtualization Design Problem (VDP)

In the VDP, N VMs run on a shared physical machine pool and each VM runs its own instance of a N instances of a DBMS [8,9]. The shared physical pool is represented by M different resources.

Each VM has a workload, whereby W_i represents the workload on the ith VM. The VDP raises the following question: "What fraction r_{ij} of each shared physical resource j should be allocated to each VM_i to optimize the overall performance of the workloads W_i?" [7–9,27]. *The set of resource allocated shares to the ith VM can be represented as a vector*:

$$R = [r_1, r_2, \ldots, r_M] \tag{1}$$

For example, without loss of generality, with three shared resources (CPU, memory, I/O), that is, $M = 3$, an allocation of 50% CPU, 30% memory, and 25% I/O to VM1 results in the vector $R_1 = [0.5, 0.3, 0.25]$. We assume that each workload W_i has a relevant cost under resource allocation R_i. This cost is represented by:

$$Cost(W_i, R_i) \tag{2}$$

The total cost for all workloads is represented by:

$$\mathrm{Cost}(\mathcal{R}) = \sum_{i=1}^{N} Cost(W_i, R_i) \tag{3}$$

The objective of the VDP is getting an appropriate resource allocation to minimize the overall cost for all workloads, that is, to find:

$$arg\ min(cost(\mathcal{R})) \tag{4}$$

The VDP was defined and solved in [7–9]. The next section explains in detail the virtual design advisor as a solution for the VDP.

3.2. Virtual Design Advisor (VDA)

The architecture and design of the Virtual Design Advisor (VDA), which was introduced as a solution for the virtualization design problem is shown in Fig. 1 [8,9]. The VDA is divided into two modules; configuration enumeration, which includes the search algorithm, and the cost model. The modules interact to produce the recommended configurations using a calibration process. The calibration process tunes the cost

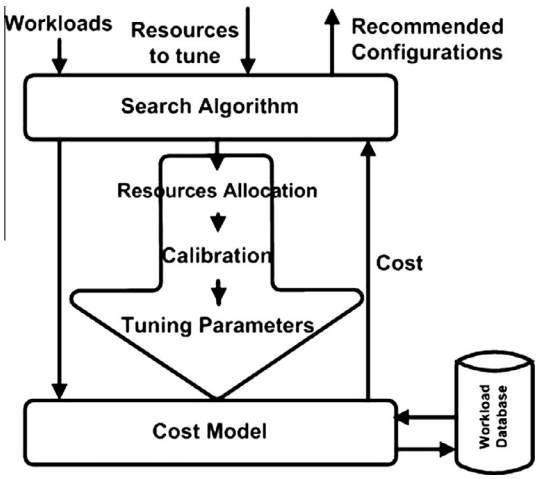

Figure 1 Virtualization Design Advisor (VDA) architecture.

model parameters according to each enumerated configuration. A brief description of both modules is presented next.

3.2.1. Configuration enumeration module

The configuration enumeration module is used to enumerate resource allocation for the VMs. It implements a search algorithm, such as greedy search and dynamic programming, for enumerating candidate resource allocation. The VDA uses a greedy search algorithm, which is based on iterating until no performance gain can be incrementally achieved [8,9]. Each iteration, a small fraction of a resource is de-allocated from the VM that will get hurt the least and allocated to the VM that will benefit the most. The greedy algorithm makes the decisions of increasing and decreasing the resources allocated to VMs based on the estimated cost of the given workloads.

3.2.2. Cost model

The VDA employs the cost model of the DBMS query optimizer after augmenting it with virtualization awareness. The cost model reflects a VM with a certain resource allocation by setting appropriate parameter values of the query optimizer. The query optimizer in a DBMS estimates the cost of an execution plan of a given SQL workload (W_i) on a DBMS instance (D_i) using the following vector of optimizer tuning parameters:

$$P_i = [p_{i1}, p_{i2}, \ldots, p_{il}]) \tag{5}$$

The optimizer's tuning parameters strongly affect the best execution plan choice. The DBMS cost model can be described by the following function [11]:

$$Cost_{DB}(W_i, P_i, D_i) \tag{6}$$

The VDA faced a problem to tune the cost model of a DBMS instance that runs in a virtualized environment. This problem can be described as the DBMS cost model which depends on a set of query optimizer tuning parameters (P_i), whereas the configuration enumerator outputs candidate resource allocation (R_i). So, a calibration process is needed to map this resource allocation into the relevant tuning parameter values.

3.3. Calibration problem in VDA

In the VDA, calibration is a process that is used for mapping each resource allocation into a corresponding set of values of the query optimizer's tuning parameters. For each tuning parameter, there is a calibration equation which used to describe the relationship between the tuning parameter and the corresponding resource allocation. In general, the calibration equation is described as:

$$P_i = f(R_i) \tag{7}$$

where R_i is the set of resource fractions allocated to the ith VM.

This process uses a calibration model that is constructed empirically and consists of a set of calibration equations [8,9,27]. By this, the query optimizer becomes aware of the virtualized environment it runs in. In other words, the query optimizer chooses an optimal execution plan by estimating and comparing the costs of a set of plans based on the given resource allocation.

Unfortunately, the calibration process is done manually, which is considered a tedious process and has to be repeated for each different combination of DBMS and server hardware specifications. By automating the calibration process would save both time and efforts. This paper focuses on the design and implementation of a tool to automate the manual calibration process. In other words, this paper addresses the question: "*how much time would be saved by automating the calibration process to avoid repeating the manual process every time the DBMS has to run on a different server infrastructure, or the DBMS is replaced with another DBMS?*" The proposed tool will be described in Section 4.

4. Automatic calibration tool

The Automatic Calibration Tool (ACT) is considered the first contribution of this paper. The ACT automates the cost model calibration process, which is considered an important part in the virtual design advisor. The ACT hides the details and complexities of the calibration process from the DB administrator. The output of ACT, namely the calibration model, is used to adapt the query optimizer's tuning parameters to the virtual machine's resources allocation.

The calibration model is basically a set of equations that calculate the tuning parameter values based on given resource allocation. This section starts by an overview of the architecture and configuration of the ACT followed by a description of its two modules, namely the controller module and the worker module.

4.1. The ACT overview

According to Fig. 1, the calibration process maps between resource allocations and the tuning parameters of the query optimizer's cost model. According to the manual calibration, the calibration process has to be repeated manually when the VDA is to be redesigned for different DBMSs, and when the same DBMS is moved to a new physical infrastructure with different CPU speed, physical memory size, etc. So, the more possible configurations a physical infrastructure can offer, the more time and complexity it takes for the calibration pro-

cess, especially if it is done manually. The proposed automatic calibration is more accurate and useful than manual calibration because of time and cost saving.

We assume that the user of ACT has an expert knowledge of the DBMS's query optimizer and cost model, as well as, the internals of the DBMS cost model to know which tuning parameters should reflect the runtime environment (e.g., CPU speed and memory size) of the DBMS, which resource allocation affects which tuning parameters, and which parameters are dependent on other parameters, so that the parameter equation (PE) needs more than one calibration query to evaluate it [8,9]. This information is needed to craft the calibration queries and define their corresponding cost Eq. (6). Also, the ACT allows its user to choose the type of the automatic calibration, either cold-cache or warm-cache. In cold-cache calibration, the ACT starts with empty buffer pool cache in the DBMS. In the warm-cache calibration, the calibration database's buffer pool is warmed up before measuring the calibration queries' runtime. Fig. 2 depicts the architecture of the ACT tool. It contains two main modules, the controller and the worker, that interact to automate the calibration process of the query optimizer's tuning parameters. The controller module runs on the host machine while the worker module runs on a virtual (guest) machine with different allocations.

4.2. Controller module

The controller is the main module in the ACT. It runs on the host machine (called Dom0 in Xen terminology [28]). It receives inputs from the tool user, and produces the calibration model as a set of equations, in which the independent variables are the resource allocations, and the dependent variables are the tuning parameters. To prepare the inputs to the controller, the system of cost equations (CEs) is solved by the tool user, whereby the unknowns are the calibration (tuning) parameters and the equations represent the costs corresponding to carefully crafted SQL queries (called calibration queries). The cost of each calibration query is represented by exactly one cost equation that is formulated in terms of the calibration parameters. The inputs to the controller module are the calibration queries and the solution of the cost equations (CEs), that is, a set of parameter equations (PEs) with calibration query costs as the independent variables and calibration parameters as the dependent variables. The worker module (will be described in the next subsection) evaluates the cost of the calibration queries for each configured resource allocation and calculates the corresponding tuning parameter values by direct substitution into the PEs. The controller module outputs a calibration model by running a regression analysis on the (resource allocation, calibration parameter) value pairs. The work flow of this module is shown in Fig. 3.

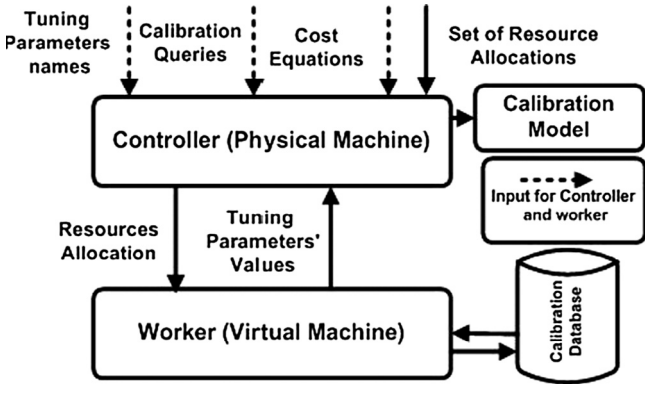

Figure 2 ACT tool architecture.

4.3. Worker module

The worker is the second module in the ACT. It runs in a guest VM. The worker module receives its inputs from the controller module and sends its output back to the controller. It uses the calibration database for executing the input queries.

As mentioned earlier, the worker module evaluates the cost of the calibration queries for each configured resource allocation and calculates the corresponding tuning parameter values by direct substitution into the PEs. Whereas the query cost in the cost equations (CEs) is measured in units of sequential page read, the measured cost by the worker is in seconds. Therefore, a renormalization process takes place to convert be-

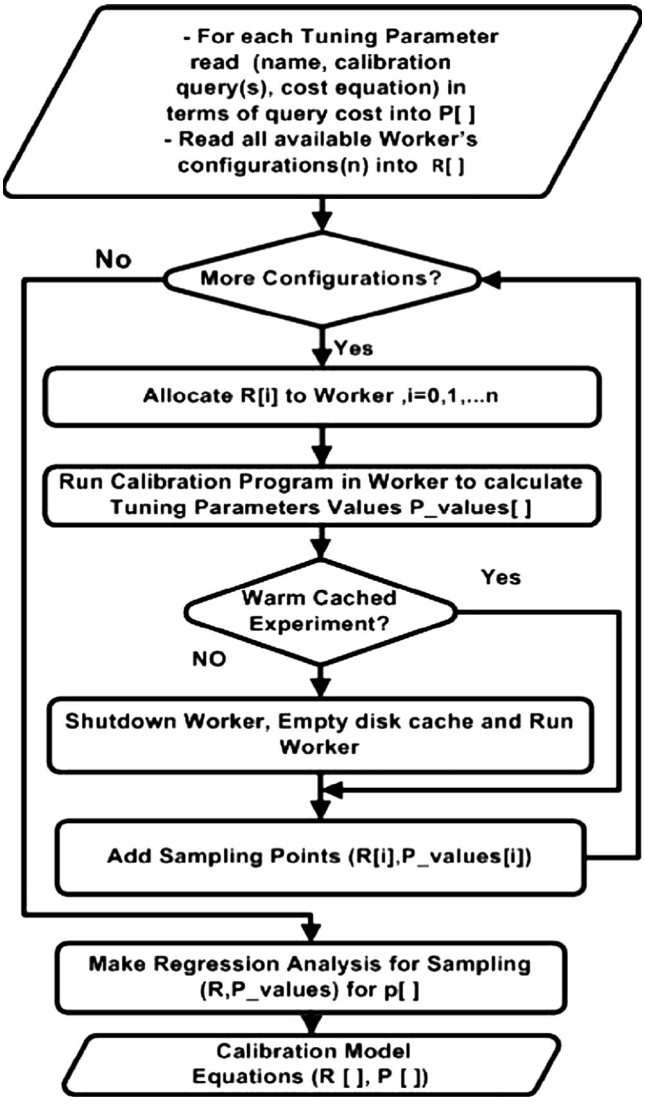

Figure 3 Controller module operations.

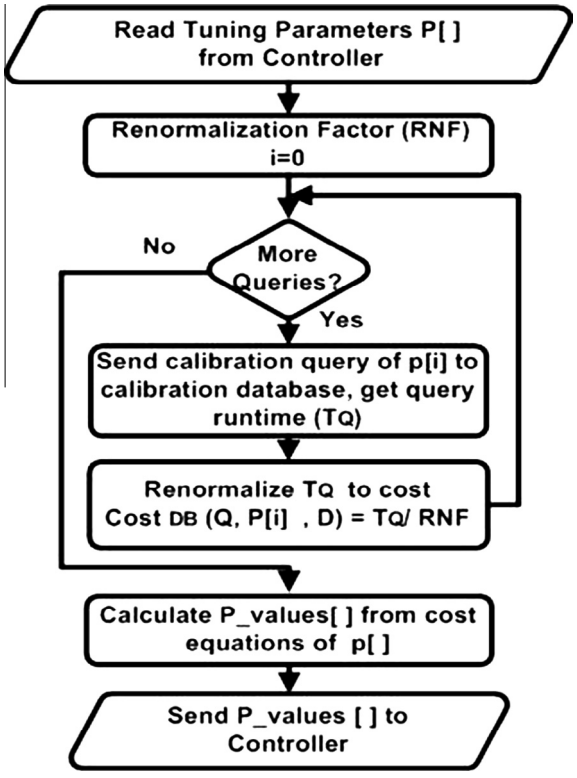

Figure 4 Worker module operations.

tween the measured cost in seconds and the cost unit in the CEs [8]. To this end, a Renormalization Factor (RNF) is calculated as an estimate of a single sequential I/O operation. The work flow of this module is shown in Fig. 4.

5. Optimization problem in virtual design advisor

The search algorithm in the virtual design advisor uses the calibration process to enumerate configurations for the VMs. The search algorithms use the "*what-if*" mode of the query optimizer's cost model [7]. The "*what-if*" mode can be expressed as what will be the estimated cost of the given query workload under the candidate resource allocation. The search algorithm modifies the query optimizer's tuning parameters using the calibration process. The calibration process can profile the intensively of workload even CPU-intensive or non-CPU-intensive and guides the VDA to allocate the suitable amounts of resources to each VM. The VDA uses a heuristic greedy algorithm, which suffers from the problem of being trapped in local optimums [8,9]. So, a new algorithm, called GPSO is introduced, based on PSO to overcome local optimum problem.

6. The proposed Greedy Particle Swarm Optimization (GPSO) algorithm

Currently, the VDA uses a greedy search algorithm, which is based on iteratively improving the cost function until no cost reduction can be achieved [8,9]. More specifically, in each iteration, a small fraction (called a *share*) of a resource is de-allocated from the VM that will get hurt the least and allocated to the VM that will benefit the most. In more details, the greedy algorithm makes a decision for increasing and decreasing the allocated resources to VMs based on the estimated cost of the given workloads. At the end, the greedy search algorithm gives a report of the recommended configuration for all VMs. The greedy algorithm suffers from the problem of being trapped in local optimums [8,9]. So, the particle swarm optimization search algorithm based on greedy algorithm will be used to reduce trapping in local optimum. First, a brief description of PSO will be given.

6.1. Particle swarm optimization

A **P**article **S**warm **O**ptimization is one of the modern evolutionary algorithms used to explore the search space of a given problem. Kennedy and Eberhart first have proposed this algorithm in 1995 [10]. PSO simulates the social behavior of individuals (particles) of certain kinds of animals (e.g., birds' flocks and fish schools). In PSO, the population of particles is typically called a swarm, whereas each the swarm. The idea of PSO is based on introducing the observation of swarming movement to the field of evolutionary computation [29,30].

Each particle moves in a D-dimensional space (D usually represents the number of decision variables). Each particle is thus described by a tuple of vectors (X_i, V_iP_i, G_i), where each vector represents the current position, the velocity vectors, the personal best position that the particle has achieved, and the global best position that is tracked by the entire swarm to ith particle along each of the D dimensions respectively.

Initially, the PSO algorithm chooses candidate solutions randomly within the search space. Then, they move in randomly-defined directions based on best of itself and of its peers. Each iteration of the algorithm, the particles evaluate their positions toward a goal. They update their own velocities using globally best positions and their previous positions and then use these velocities to adjust their new positions. The used equation to update the velocity and position for each particle are:

$$v_{id}(t+1) = wv_{id}(t) + c_1r_1[pbest_{id}(t) - x_{id}(t)] + c_2r_2[gbest_d(t) - x_{id}(t)] \quad (8)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \quad (9)$$

where all of parameters are represented in dth dimension at time t, $v_{id}(t)$ is the velocity of ith particle, $w{\cdot}v_{id}(t)$ is the inertia component responsible for keeping the particle moving in the same direction, $w(w \in [0.8, 1.2])$ is an inertia weight that determines how much the previous velocity is preserved [31], $x_{id}(t)$ is the position of the ith particle, $pbest_{id}(t)$ is the personal best position for the ith particle, $gbest_d(t)$ is the globally best position (the swarm's global best candidate solution at time t), c1, c2 are positive acceleration coefficients ranging from 0 to 4, and r1, r2 are random numbers drawn from the uniform distribution $U[0, 1]$. The search is a repetitive process, and the stopping criteria are that either the maximum number of iterations is reached or the minimum error condition is satisfied.

6.2. Greedy particle swarm optimization algorithm

A hybrid of the heuristic greedy search and intelligent particle swarm optimization is proposed as a new algorithm to overcome the local optimum states to global ones. This algorithm

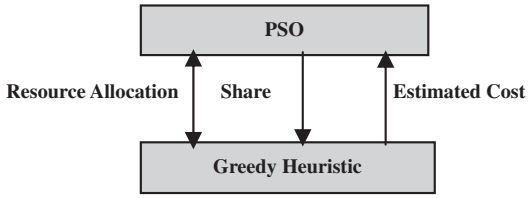

Figure 5 GPSO in VDA enumerator module.

is called **G**reedy **P**article **S**warm **O**ptimization (GPSO). The proposed GPSO algorithm is required more computation but is succeeded to enhance the result VM configurations in many cases. Fig. 5 depicts the idea of the proposed GPSO algorithm. The main idea is that the GPSO algorithm uses PSO algorithm to tune the *share* parameter of the heuristic greedy algorithm to reduce the situations in which the greedy algorithm gets trapped into local optima. Two modules have been implemented within GPSO algorithm which they interact to find the recommended configuration as follows:

(1) The greedy module enumerates resource allocations for the VMs based on the estimated cost of the given workloads.
(2) The PSO module sends to the greedy module candidate *shares* (particles) and VMs configurations and then receives the updated VMs configurations and the corresponding estimated cost for these configurations.

In this setting, the particles of the PSO module are the *shares* parameters to be tuned and the dimensions of the particles are the number of resources. This work focuses on one resource (the CPU), and thus, the particles in PSO has a single dimension. In the other words, the *share* parameter serves as the only dimension of particle position. The improved PSO, SSM-PSO, is used to avoid invalid-solution cases [32]. The effect of the GPSO algorithm is achieved by iteratively running the heuristic greedy algorithm with a new *share* computed using PSO. In each iteration, the heuristic greedy is started from the last solution (the configuration of the global best) reached in the previous iteration, which is considered as a local optimum. The GPSO algorithm has been implemented in the VDA enumerator (search) module.

6.3. Configure of standard PSO factors

The parameters of PSO influence the optimization performance. PSO needs to predefine numerical coefficients (the maximum velocity, inertia weight, momentum factor, societal factor, and individual factor) and swarm size. The ability to globally optimize the solution relies greatly on the setting of these parameters. The maximum velocity and inertia weight are employed to balance global exploration and local exploitation. A large value of inertia weight facilitates better global exploration ability, whereas a small value enhances local exploitation capability. In other words, they affect the ability of escaping from local optimization and refining global optimization. The societal and individual factors determine the ability of exploring and exploiting. The size of swarm balances the requirement of global optimization and computational cost [30,33,34].

In GPSO algorithm, the coefficients of PSO component, $r1$ and $r2$, are generated randomly, $c1 = c2 = 2$, and a constant momentum factor, $mc = 0.3$, is adopted. The PSO component has a gradually decreasing inertia weight factor. The inertia factor w decreases linearly between 0.9 and 0.4 as in the following equation [33]:

$$w = (w_{max} - w_{min}) \times \frac{(Iter_{max} - Iter_{now})}{Iter_{max}} + w_{min} \tag{10}$$

where $Iter_{max}$ is the maximum number of PSO iterations, $Iter_{now}$ is the current number of iterations in the running PSO, w_{max} is the maximum inertia value, which equals 0.9 and w_{min} is the minimum inertia value, which equals 0.4.

6.4. Fitness function

To evaluate each particle (*share* parameter) performance, the total of estimated costs is calculated using given workloads under candidate VMs configuration as described in Eq. (3).

6.5. The GPSO algorithm

The GPSO algorithm steps are listed as follows:

(1) Initially, equal allocation of each resource is assumed as the initial configuration for all VMs ($1/N$ of each resource is allocated to each VM).
(2) The fitness function is defined to minimize the cost as described in Eq. (3), and then the positions (*share* values) of the particles are chosen randomly. The search space includes all the possible fractions except the fractions that cause a resource allocation that is either greater than the maximum allocation (100%) or less than the minimum allocation (0%). These constraints reduce error occurrence and can be described by the following:

$$Min\,(R_i) - share > 0$$

$$Max\,(R_i) + share < 100$$

Moreover, the search space boundaries $[X_{min}, X_{max}]^D$ are restricted in $[0.001, 0.1]$. This restriction means that each share parameter can be any value between 0.1% and 10%. In this work, only one resource, CPU, is used (i.e., one-dimensional vectors for particles), and thus, GPSO is used to find a best particle (share value) to tune CPU allocation $X = (x_1, x_2, \ldots, x_n)$.
(3) GPSO operates then in iterations. Iteratively, each particle evaluates its position by running the greedy algorithm and determines its personal best position. The global best share and VM configuration are then determined. The initial VM configuration of the greedy algorithm for each particle is the VM configuration which was tuned by the global best particle of the previous iteration. Each particle then updates its own velocity using its previous velocity, the inertia weight, its previous position, its personal best position, and best particle in terms of fitness in the entire population (global best position). Each particle then uses the calculated velocity to adjust its new position.
(4) After the iterations terminate, the configuration of the best particle so far is output as the final VM configuration R.

As stated previously in the listed steps, for each iteration and for each particle, the greedy algorithm uses the new share and the previous optimal configuration as the initial state. The previous configuration is the local optimum, and when the *share* value is changed by PSO, this allows the greedy algorithm to escape from the trap of the local optimal solution to a global optimal solution.

7. The ACT tool and GPSO algorithm evaluation

This section presents an experimental evaluation of the proposed ACT tool and GPSO algorithm.

7.1. Experiment setup

The experiment described here uses PostgreSQL 8.4.8 database system installed in a machine with Core2 Duo T5870 2.00 GHz processor, 4 GB memory, and CentOS 5.5 operating system. The virtual machine monitor used was Xen [28], which is an open source virtualization platform. Xen-based para-virtualization has been used to improve the hypervisor performance when it maps resources directly into the guest operating system [5]. Amazon EC2 is based on Xen virtualization, and thus, this experiment setup is similar to a cloud computing environment.

7.2. Performance metrics

Four metrics are used to measure the performance.

(1) The speed of the ACT tool, measured in units of time (minutes).
(2) The total estimated cost of workloads (in terms of sequential page fetches) is computed by selecting CPU parameters (*cpu_tuple_cost* and *cpu_operator_cost*) as a shared resource then setting these parameters appropriately according to the resulted calibration model on a warm database.
(3) Cost improvement measures relative performance as in the formula [8,9,23]. This metric is computed based on the estimated cost of the query optimizer. In this work, using two algorithms (greedy and GPSO), the formula is as follows:

$$improvment = \frac{Est\_Cost_{Greedy} - Est\_Cost_{GPSO}}{Est\_Cost_{Greedy}} \quad (11)$$

where Est_Cost_{Greedy} and Est_Cost_{GPSO} are the total estimated cost under greedy and GPSO configuration, respectively.
(4) Cost improvement per unit time is computed as follows:

$$Cost\_improvment\_per\_time = \frac{Cost\_improvment}{avg(runtime)} \quad (12)$$

7.3. The ACT tool evaluation

Two of PostgreSQL descriptive parameters have been used in this evaluation. The *cpu_tuple_cost* represents an estimate of the CPU cost of processing one database tuple. The cost is measured in terms of the cost of a sequential page fetch from the disk. The *cpu_operator_cost* represents an estimate of the CPU cost of processing each operator in a WHERE clause [8]. This subsection presents a step-by-step scenario of running ACT's calibration process followed by the ACT speed measure. Fig. 6 depicts the scenario of using ACT in calibrating two of PostgreSQL's tuning parameters, namely (*cpu_operator_cost*, *cpu_tuple_cost*). The scenario steps were described in details in [27].

On the other hand, to assess the speed of ACT tool, its total runtime (i.e., runtime of both the controller and worker modules) has been measured under a varying number of resource allocation configurations with both cold-cache and warm-cache calibration. Other factors that affect the runtime include the DBMS, physical machine computing power, and values (not just number) of configurations.

With cold-cache, the controller restarts the worker's VM with each resource configuration, increasing the total runtime. With one, the ACT tool consumed 4 min to run one configuration (50% CPU and 50% memory) with cold-calibration. On the other hand, with warm cache, the worker module is started first, before ACT tool runs, and the calibration database is warmed up. With one configuration (50% CPU and 50% memory), it took ACT 1.6 min to finish the calibration process. Fig. 7 shows that ACT runtime linearly increased with number of configurations even cold or warm cache. Also, cold-cache experiment takes long time comparable with warm-cache experiment.

7.4. The GPSO algorithm evaluation

This section presents an experimental comparison between the proposed GPSO algorithm and the greedy algorithm.

7.4.1. GPSO algorithm swam size variation

We vary the size of the swarmin the PSO module of the GPSO algorithm within the range [10–100] to test the GPSO algorithm for two different workloads running on two VMs. We repeat each experiment ten times and report the average. We found that the total (and variance) estimated cost of the workloads for small swarm size is greater than the estimated cost of large swarm size. Consequently, the cost improvement per unit time is calculated in next subsection using different swarm sizes and two search spaces to obtain the best swarm size.

7.4.2. GPSO algorithm search space ranges variation

The GPSO algorithm performance is evaluated by varying the swarm size in two search space boundaries, [0.01% -10%] and [0.1%-10%], to choose the feasible swarm size. The first search space contained 100 points, whereas the second contained 1000 points. Each point in search space represents a value of the share parameter, which is used as a controller of the greedy heuristic algorithm. The GPSO cost improvement over greedy per time unit is used to compare the two search spaces using Eq. (12) (see Fig. 8).

According to the results in Fig. 8, the first search space improvement is better than the second until the swarm size reaches 50, at which point the improvement decreased in the two search spaces nearly with the same ratio. As a result, the first search is used with swarm size 10 in the following experiments. Table 1 gives the experimental setup of the GPSO algo-

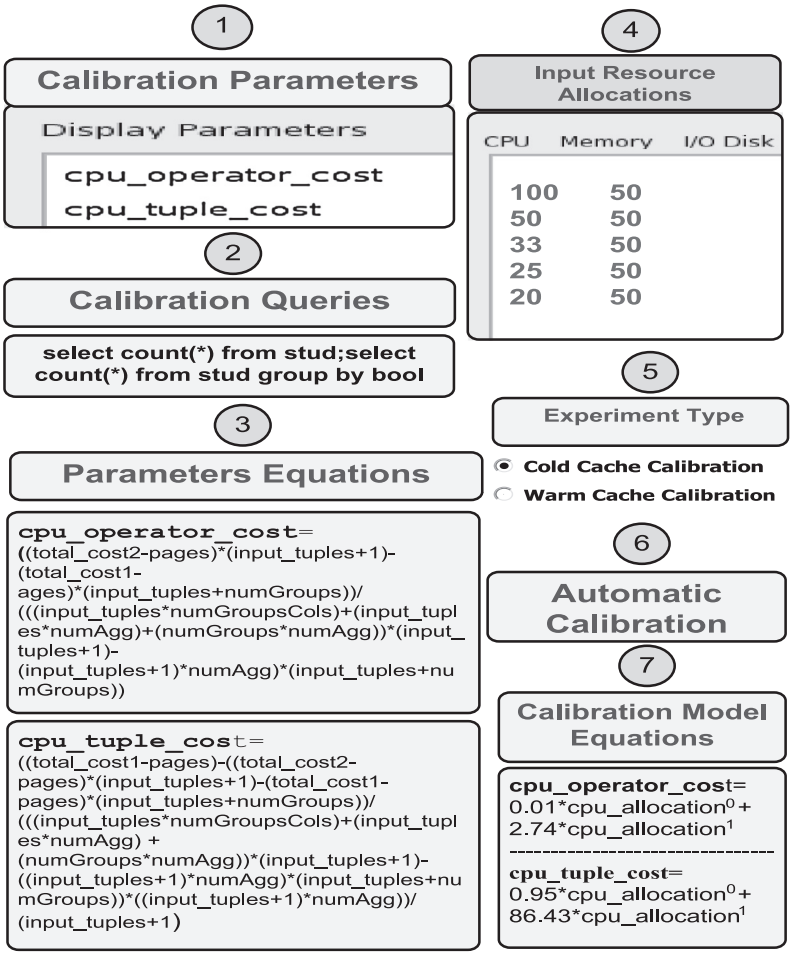

Figure 6 PostgreSQL experiment scenario using ACT tool.

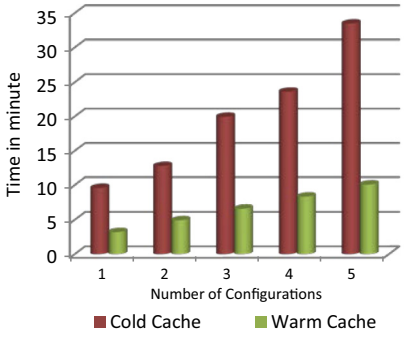

Figure 7 The runtime of ACT with cold-cache and warm-cache calibration.

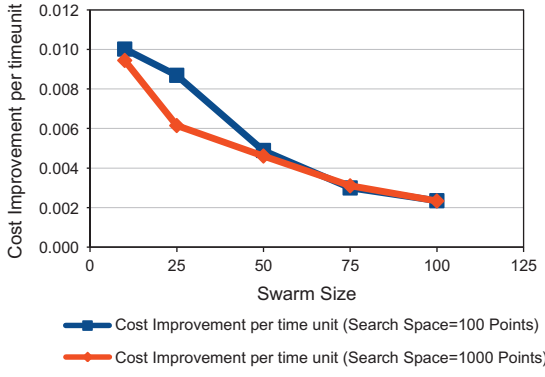

Figure 8 Effect of swarm size on cost improvement for two search spaces total cost for two VMs.

rithm. The number of executions represents the number of independent experiments done. The greedy algorithm starts with equal allocations for all VMs and with a *share* parameter of (5%).

7.4.3. GPSO algorithm with identical workloads

The aim of this experiment is to conclude that the GPSO algorithm partitions the shared resource, CPU, into equal allocations when the workloads are identical – i.e., the GPSO

Table 1 The parameter values for GPSO experiments.

Swarm size (number of particle)	10
Number of iterations	50
Number of executions	10
Search space range values	0.1%-10%; 100 points

algorithm is efficient in detecting the identical workloads, which reflects the fair distribution of the shared resource. Fig. 9 shows the estimated costs for 10 VMs that run 10 identical copies of TPC-H Q1 query workloads. The graph plots three estimated costs as identical columns of the two algorithms, greedy and GPSO, and the default configuration.

7.4.4. GPSO algorithm with random workloads

In this experiment, random TPC-H workloads are generated to test the improvement of overall performance. Twenty queries are generated by the same method described in the [8]. Each workload consists of a random combination of between 10 and 20 workload units. A workload unit can be either 1 copy of TPC-H query Q17 or 66 copies of a modified version of TPC-H query Q18 [8,9]. Each VM runs one workload. Each algorithm starts with 2 VMs and increases by 1 VM until it reaches 20 VMs. Fig. 10 shows the total estimated costs for three configurations and shows the decreased ratio in the GPSO algorithm estimated cost. The estimated cost obtained by the GPSO algorithm is lower than the estimated cost obtained by the greedy algorithm. In other words, the GPSO algorithm outperforms the greedy algorithm with respect of estimated cost.

The performance improvement of the proposed GPSO algorithm is calculated using the Eq. (11). The results are plotted in Fig. 11. It is noted that the greatest improvement appeared when the greedy algorithm has local optimum at 19 workloads. The greedy cannot improve configuration and stopped in initial configuration (default configuration) while the GPSO algorithm can be improve by using another share to escape from this local optimum.

According to the result, the GPSO algorithm achieves better allocations in terms of total cost at the expense of runtime. Although, there is time overhead of execution runtime that the GPSO algorithm is slower as compared to the greedy algorithm. Since the distribution of shared resources is considered an off-line process in VDA, the GPSO algorithm is acceptable for obtaining near optimal configurations for VMs.

The combination of the GPSO algorithm with any profiling technique for random workloads characteristics in terms of resource consumption (e.g., CPU, Memory, and I/O) gives the perception for the intensivity of workloads. This perception can guide the cloud provider to allocate an appropriate

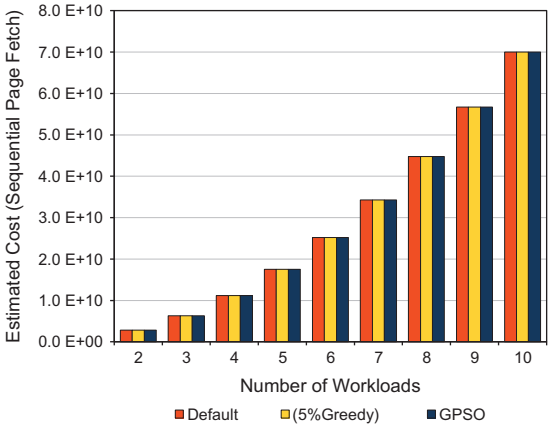

Figure 9 Cost for identical workloads.

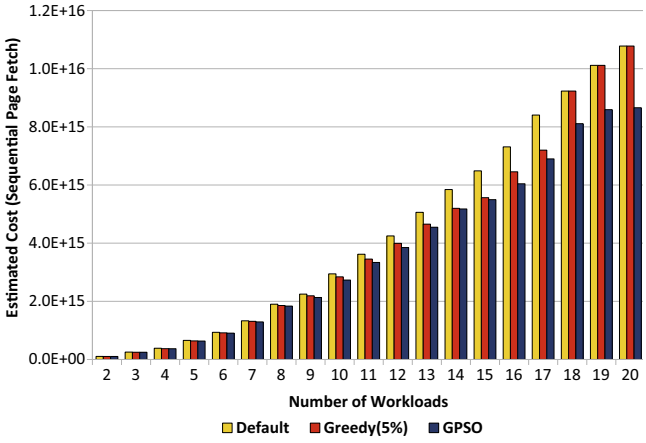

Figure 10 Cost comparison for up to 20 random workloads on TPC-H database.

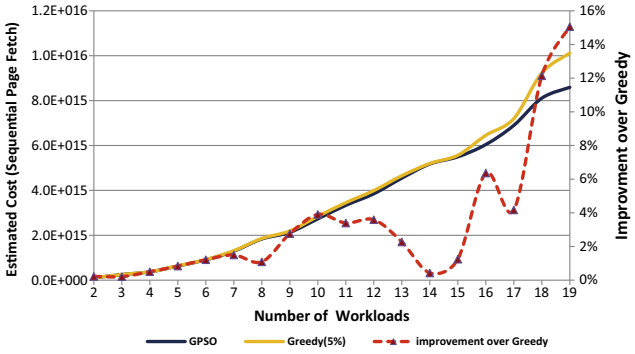

Figure 11 The GPSO algorithm cost improvement over greedy algorithm.

amount of resources to incoming workloads. The provider can arrange the workloads over multiple pools based on the intensively of workloads or use cloud bursting to maintain strict SLA even when some incoming workloads are heavily CPU-intensive. Where cloud bursting means that an application deployment model in which an application runs in a private cloud or data center bursts into a public cloud when the demand for computing capacity spikes [35]. The advantage of such hybrid cloud deployment is that an organization only pays for extra compute resources when they are needed [35]. On the other hand, the GPSO algorithm can be used continuously to capture the randomness of the dynamic workloads variation by implementing it again periodically or on particular events and changed the resource allocation periodically in each time interval.

8. Conclusions and future work

According to the work in this paper, the virtual design advisor has been improved by proposing, and implementing the Automatic Calibration Tool (ACT) tool. The function of this proposed tool is to automate the process of calibrating the tuning parameters of the query optimizer of databases in a what-if mode, so that it can estimate the cost of running workloads in virtualized environments quickly and accurately. The

ACT has been evaluated using an experiment to tune parameters of the PostgreSQL DBMS. The experimental results show that the ACT runtime increased linearly with the number of resource configurations.

Also, a hybrid particle swarm optimization based on the heuristic approach namely, GPSO, which used to minimize the total cost of workloads on the cloud environment has been introduced. The GPSO algorithm has been evaluated using TPC-H queries and PostgreSQL database. According to the results, it is found that the GPSO algorithm behaves better than the heuristic approach by enhancing the fitness value to avoid a local optimum and find global optimum when possible.

This work can be extended in at least two ways. First, the ACT tool can be extended to automate "*black-box*" calibration, which does not need the DBMS's cost model internals. Second, the ACT tool can be extended to intelligently select the resource configurations (sampling points in the regression analysis) that results in quick convergence to the calibration model equations. This will be making by building a profiling technique to obtain the statistical methods that deal with different workloads behavior.

Another option to extend this work is considering other resources such as I/O performance and network bandwidth, and mix of QoS to provide a more flexible approach.

On the other hand, the GPSO algorithm fitness function could be upgraded for dynamic workloads to involve two factors: (1) weighted factor of cost and time to influence the share parameter in order to improve the GPSO to choose the best share toward minimizing the GPSO estimated cost and runtime and (2) define penalty factor which reflects the SLA between the users and cloud provider to handle the SLA violation.

References

[1] Qi Zhang LC, Boutaba Raouf. Cloud computing: state-of-the-art and research challenges. J Int Serv Appl 2010;1(1):7–18.

[2] Soror AA, Minhas UF, Aboulnaga A, Salem K, Kokosielis P, Kamath S. Deploying database appliances in the cloud. IEEE Data Eng Bull 2009;32(1):13–20.

[3] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, et al. Xen and the art of virtualization. In: Proceedings of the nineteenth ACM symposium on operating systems principles. Bolton Landing, NY, USA; 2003. p. 164–77.

[4] Goldberg RP. Survey of virtual machine research. IEEE Comput 1974;7:34–45.

[5] Rose R. Survey of system virtualization techniques. Lisbon, Portugal: Theses (Electrical Engineering and Computer Science) MS non-thesis Research Papers (EECS); 2004.

[6] Computing C. Handbook of cloud computing; 2010. <http://www.springerlink.com/index/10.1007/978-1-4419-6524-0>.

[7] Soror AA, Aboulnaga A, Salem K. Database virtualization: a new frontier for database tuning and physical design. In: Proceedings of ICDE workshops (SMDB 2007); 2007. p. 388–94.

[8] Soror AA, Minhas UF, Aboulnaga A, Salem K, Kokosielis P, Kamath S. Automatic virtual machine configuration for database workloads. In: Proceedings of ACM SIGMOD international conference on management of data (SIGMOD'08). Canada: Vancouver; 2008. p. 953–66.

[9] Soror AA, Minhas UF, Aboulnaga A, Salem K, Kokosielis P, Kamath S. Automatic virtual machine configuration for database workloads. ACM Trans Database Syst 2010;35:1–47.

[10] Kennedy J, Eberhart RC. Particle swarm optimization. In: IEEE international conference on neural networks; 1995. p. 1942–8.

[11] PostgreSQL 8.3.18 Documentation. <http://www.postgresql.org/docs/8.3/static/plpgsql.html> [April 16, 2012 6:30 PM].

[12] TPC-H Homepage. <http://www.tpc.org/tpch/> [2001, 9-11-11 1:29 AM].

[13] Padala P, Shin KG, Zhu X, Uysal M, Wang Z, Singhal S, et al. Adaptive control of virtualized resources in utility computing environments. In: Proceedings of the 2nd ACM SIGOPS/EuroSys European conference on computer systems. Lisbon, Portugal; 2007. p. 289–302.

[14] Mitran P, Long Le, Rosenberg Catherine, Girard André. Resource Allocation for Downlink Spectrum Sharing in Cognitive Radio Networks, in Proceedings of the VTC Fall, 2008, pp. 1–5.

[15] Soundararajan G, Lupei D, Ghanbari S, Popescu AD, Chen J, Amza C. Dynamic resource allocation for database servers running on virtual storage. In: Proceedings of the 7th conference on file and storage technologies. San Francisco, California; 2009. p. 71–84.

[16] Narayanan D, Thereska E, Ailamaki A. Challenges in building a DBMS resource advisor. IEEE Data Eng Bull 2006;29:40–6.

[17] Skelley A. DB2 advisor: an optimizer smart enough to recommend its own indexes. In: Proceedings of the 16th international conference on data, engineering; 2000. p. 101.

[18] Air TC. Force inst of tech Wright–Patterson AFB OH and shrum, calibration and validation of the checkpoint model to the air force electronic systems center software database: storming, media; 1997.

[19] Sheng Weihua, Schurmans S, Odendahl Maximilian, Leupers Rainer, Ascheid Gerd. Automatic calibration of streaming applications for software mapping exploration. In: Proceedings of the international symposium on system-on-chip (SoC), IEEE; 2011. p. 136–42.

[20] Rogers J, Papaemmanouil O, Dietrich UCW. A generic auto-provisioning framework for cloud databases. In: Proceedings of the ICDE workshops; 2010. p. 63–8.

[21] Goudarzi H, Pedram M. Maximizing profit in cloud computing system via resource allocation. In: Distributed computing systems workshops, international conference, vol. 0; 2011. p. 1–6.

[22] Goudarzi H, Pedram M. Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems. In: Cloud computing, IEEE international conference on cloud computing. Los Alamitos, CA, USA; 2011. p. 324-31.

[23] Kong S, Li Y, Feng L. Cost-performance driven resource configuration for database applications in IaaS cloud environments. In: Ivanov I, van Sinderen M, Shishkov B, editors. Cloud computing and services science. New York: Springer; 2012. p. 111–29.

[24] Li Changhe, Yang Shengxiang. Fast multi-swarm optimization for dynamic optimization problems. In: Proceedings of the fourth international conference on natural computation, IEEE; 2008. p. 624–8.

[25] Carlisle A, Dozier G. Adapting particle swarm optimization to dynamic environments. In: Proceedings of the international conference on artificial intelligence (ICAI); 2000. p. 429–34.

[26] Van den Bergh Frans, Engelbrecht Andries Petrus. A cooperative approach to particle swarm optimization. IEEE Trans Evol Comp 2004;8(3):225–39.

[27] Sahal R, Khattab SM, Omara FA. Automatic calibration of database cost model in cloud computing. In: 2012 8th International conference on informatics and systems (INFOS); 2012. p. CC-25–CC-34.

[28] Williams DE. Virtualization with Xen: Including Xenenterprise, Xenserver, and Xenexpress: Syngress Publishing; 2007.

[29] Blackwell T. Particle swarm optimization in dynamic environments. In: Yang S, Ong Y-S, Jin Y, editors. Evolutionary computation in dynamic and uncertain environments. Berlin, Heidelberg: Springer; 2007. p. 29–49.

[30] Jinxia R, Shuai Y. A particle swarm optimization algorithm with momentum factor. In: 2011 fourth international symposium on

proceedings of the computational intelligence and design (ISCID); 2011. p. 19–21.
[31] Shi Y, Eberhart R. A modified particle swarm optimizer. In: Proceedings of IEEE international conference on evolutionary computation; 1998. p. 69–73.
[32] Liu Y, Qin Z, He X. Supervisor-student model in particle swarm optimization. IEEE Cong Evol Comput (CEC) 2004;1:542–7.
[33] Jun-yi C, Bing-gang C. Design of fractional order controllers based on particle swarm optimization. In: 2006 1ST IEEE conference proceedings of the industrial electronics and applications; 2006. p. 1–6.
[34] Tao X, Jun W, Xiaofeng L. An improved particle swarm optimizer with momentum. In: Proceedings of the IEEE congress on evolutionary computation (CEC 2007); 2007. p. 3341–5.
[35] What is Cloud Bursting. <http://searchcloudcomputing.techtarget.com/definition/cloud-bursting> [2012, 7-11-12].