RESEARCH ARTICLE OPEN ACCESS

An Improved Search Algorithm for Resource Allocation in Cloud Databases

Radhya Sahal[1], Sherif M. Khattab[2], Fatma A. Omara[3]

Computer science department, Cairo University/Faculty of computer science, Egypt, Cairo

----------**********----------

Abstract:

The Virtual Design Advisor (VDA) has addressed the problem of optimizing the performance of Database Management System (DBMS) instances running on virtual machines that share a common physical machine pool. In this work, the search algorithm in the optimization module of the VDA is improved. An Exhaustive Greedy algorithm (EG) studies the effectiveness of tuning the allocation of the shared resources (the share values); and presents a mathematical analysis of the effect of the share values on reaching an optimal solution. Also, it studies the effect of the share values of resources on the feasibility and speed of reaching an optimal solution. On the other hand, the particle swarm optimization (PSO) heuristic is used as a controller of the greedy heuristic algorithm to reduce trapping into local optima. Our proposed algorithm, called Greedy Particle Swarm Optimization (GPSO), was evaluated using prototype experiments on TPC-H benchmark queries against PostgreSQL instances in Xen virtualization environment. Our results show that the GPSO algorithm required more computation but in many test cases succeeded to escape local optima and reduce the cost as compared to the greedy algorithm alone. Also, the EG search algorithm was faster than the GPSO algorithm when the search space of the share values grows.

Keywords: **- Virtualization, Resource Allocation, Particle Swarm Optimization, Greedy Search, Query Optimizer.**

----------**********----------

I. INTRODUCTION

Cloud computing 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 provisioning, which allows the users to deploy their own sets of computing resources [2].

Cloud computing relies on virtualization to partition and multiplex physical machine infrastructure [4]. A virtual machine configuration or resource allocation controls the sharing of physical resources 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 cloud computing, and indeed VM configuration affects application performance [2, 5].

Database Management Systems (DBMS) are important consumers of cloud resources. the Virtualization Design Problem (VDP) studies how DBMS instances can get a resource allocation for each VM out of the shared physical pool [6, 7]. The Virtual Design Advisor (VDA) is a technique that offers a solution for the VDP problem. It provides recommended configurations for multiple VMs running different workloads over shared physical resources. VDA explores the characteristics of workloads to distinguish their resource-sensitivity (e.g., CPU-intensive or I/O-intensive) and makes a decision for the best resource allocation for VMs that run the workloads. VDP

is considered as a search problem to minimize the allocation cost of virtualized resources for database systems in cloud environment [2, 6, 7].

Whereas our previous works focused on automating the manual calibration process for tuning parameters of DBMS query optimizer and introduced search algorithm to improve the resource allocation in cloud databases [8-10]. In this work, an Exhaustive Greedy algorithm (EG) will be introduced for studying the effectiveness of tuning the allocation of the shared resources (the share values); and for presenting a mathematical analysis of the effect of the share values on reaching an optimal solution. Also, it will study the effect of the share values of resources on the feasibility and speed of reaching an optimal solution. In other words, EG search algorithm would benefit from using a share parameter value that is large, and thus allows for fast convergence, and at the same time gives a result that is as good as with smaller and slower-converging shares.

On the hand, we propose a search algorithm, namely Greedy Particle Swarm Optimization (GPSO), as a hybrid between two heuristics: greedy and particle swarm optimization. Particle Swarm Optimization (PSO) is an evolutionary algorithm that explores the search space of a given problem to find optimal or near-optimal solutions for maximization and minimization search problems. Although, there are a wide variety of search techniques such as Genetic Algorithm (GA), Tabu Search (TS), Simulated Annealing (SA), and the Evolution Strategy (ES), the PSO algorithm is considered simple in concept, easy to implement, and computationally efficient [9].

The goal of the proposed GPSO algorithm is to optimize resource configurations based on the workload profile in virtualized environments. The GPSO algorithm has been implemented in the VDA enumerator module. To evaluate our proposed GPSO algorithm, experiments have been conducted to allocate the CPU resource over virtual machines. Tests have been performed using PostgreSQL 8.4.8, running TPC-H benchmark queries as workloads [11, 12]. The results show that the GPSO algorithm can provide effective configurations for different types of workloads.

The combination of the proposed GPSO algorithm with a profiling technique that classifies workloads characteristics in terms of resource consumption (e.g., CPU, Memory, and I/O) gives an insight into the resource intensivity of workloads. This insight can guide the cloud provider to allocate an appropriate amount of resources to incoming workloads. The provider can arrange the workloads over multiple pools based on resource requirements or use *cloud bursting* to maintain strict SLA even when some incoming workloads are heavily resource-intensive. Cloud bursting is an application deployment model in which an application that runs in a private cloud or data center bursts into a public cloud when the demand for computing capacity spikes [13] The advantage of such a hybrid cloud deployment is that an organization only pays for extra compute resources when they are needed The proposed GPSO algorithm can be run periodically or on policy-defined events to capture the variation of the dynamic workloads.

The rest of this paper is organized as follows. Related work is described in Section II. The optimization problem addressed in this work is described in Section III. Section IV and Section V motivate and describe the key idea of the proposed algorithm. An experimental evaluation is presented in Section VI. Finally, conclusions and future work are in SectionVII.

II. RELATED WORK

A significant amount of research has been conducted in the fields of performance optimization of applications running in virtualized environments [7, 14] and resource allocation [15, 16].

A highly-related problem to the work of this paper is the virtualization design problem [6, 7], 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. In [7], the virtual design advisor was 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 DBMS 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 [17, 18]. In [2], the virtual design advisor has been used to optimize the performance of database appliances that were deployed in the Amazon EC2 cloud.

Performance model calibration is an important task in many performance optimization problems [7, 19, 20]. Automating tedious calibration processes is of great benefit to the overall optimization framework. The Automatic Calibration Tool (ACT) has been proposed in our previous work to automate the calibration process of the DBMS query optimizer within the virtual design advisor [8].

The virtual design advisor employs a white-box approach for modeling the performance of the DBMS [7]. On the other hand, the black-box approach for performance modeling has been used in [14] 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, also were used to solve resource-provisioning problem for DBMS on the top of IaaS cloud [21].

Resource allocation is an important challenge that faces cloud computing providers regardless of the hierarchy of services; especially, the question of how the cloud provider can meet the clients' Service Level Agreements (SLAs) and maximize total profit is of particular interest.

In [22, 23], the SLA-based resource allocation problem for multi-tier cloud applications is considered for a distributed solution for each of processing, data storage, and communication resources. The problem is cast as a three-dimensional optimization problem.

The cost-performance tradeoff in cloud IaaS was addressed in [24]. The problem was formulated as a multi-objective optimization. The proposed model was built based on a fine-grained charging model and a normalized performance model. The solution used genetic algorithms, and the experimental results proved the effective of the proposed model.

On the other hand, there is a wealth of proposed approaches using the Particle Swarm Optimization (PSO) heuristic technique in various domains in general and in dynamic environments in particular. Basically, PSO is an optimization technique for static environments [25]. In the real world, however, many applications pose 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. PSO has to adapt its ability to improve and track the trajectory of the changing global best solution in a dynamic environment.

One of these algorithms is fast multi-swarm optimization (FMSO) [26]. It uses two types of swarm: one to detect the promising area in the whole search space and the other 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 [27]. 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 [27].

Cooperative Particle Swarm Optimizer (CPSO) was introduced for employing cooperative behavior to significantly improve the performance of the original PSO algorithm [28]. 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. This work proposes an algorithm, called greedy particle swarm optimization (GPSO), to optimize the allocation of shared resources to minimize estimated cost and enhance VM configuration. By devising a profiling technique to obtain statistical models that deal with different workloads behavior, an intelligent resource provisioning can be achieved to adapt to dynamic workloads to any application workload.

III. RESOURCE ALLOCATION PROBLEM

This section is dedicated to discuss the virtualization design problem (VDP) and to illustrate the virtual design advisor (VDA) solution [6, 7].

A. Virtualization Design Problem (VDP)

In the virtualization design problem (VDP), *N* VMs run on a shared physical machine pool and

each VM runs its own instance of a DBMS. The shared physical pool is represented by M different resources. Each VM_i has a workload, W_i. The VDP raises the question: "What fraction r_{ij} of each shared physical resource j should be allocated to each VM_i in order to optimize the overall performance of the workloads W_i?" *[6-8]*. The set of allocated resource shares to the ith VM can be represented as a vector:

$$R = [r_1, r_2, \dots, r_M] \quad (1)$$

For example, consider three shared resources: CPU, memory, and I/O, that is, M=3. An allocation of 50% CPU, 30% memory, and 25% I/O to VM_1 results in the vector R_1= *[0.5, 0.3, 0.25]*.

We assume that each workload W_i results in a cost (e.g., running time) under resource allocation R_i. This cost is represented by:

$$Cost(W_i, R_i) \quad (2)$$

The total cost for all workloads is represented by:

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

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

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

The estimated cost reflects application performance. The work in this paper considers only the CPU resource. The VDP was defined and a solution was presented in [6, 7]. The next section explains in detail the virtual design advisor as a solution for the VDP.

B. 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 Figure 1 [7]. The VDA is divided into two modules: configuration enumeration, which includes the search algorithm, and cost model. The modules interact to make recommended configurations using a calibration process, an automation of which was proposed in our previous work [8]. The calibration model tunes the cost model parameters according to each enumerated configuration. A brief description of both modules is presented next.

1) Configuration Enumeration Module

The configuration enumeration module enumerates resource allocations for the VMs. It implements a search algorithm, such as greedy search and dynamic programming, for enumerating and searching candidate resource allocations [7]. The greedy algorithm makes the decisions of increasing and decreasing the resources allocated to VMs based on the estimated cost of the given workloads.

2) Cost Model

The VDA tunes the cost model of the DBMS query optimizer to reflect a VM with a certain resource allocation. This tuning is done by setting appropriate values for the query optimizer parameters. 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:

$$P_i = [p_{i1}, p_{i2}, \dots, p_{il}] \quad (5)$$

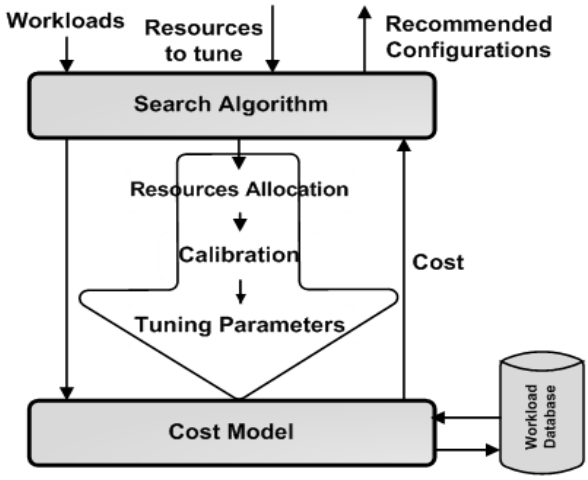

Fig. 1: Virtualization Design Advisor (VDA) Architecture.

3) Calibration

In VDA, calibration is a process to map each resource allocation into a corresponding set of values of the query optimizer's tuning parameters. This process uses a calibration model that is constructed as a set of calibration equations [7, 8]. The query optimizer becomes aware of the virtualized environment it runs in, and chooses an optimal execution plan by estimating and comparing the costs of a set of plans based on the given resource allocation [8].

C. Optimization Problem in VDA

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 [6]. 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 overall optimization process would ideally profile the intensivity of workload (e.g., CPU-intensive or non CPU-intensive) and guide 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 optima [7]. A new search algorithm based on PSO is proposed in this paper to reduce trapping in local optima, as will be described in Section V. The next section motivates the proposed search algorithm by analyzing the greedy search algorithm used in the VDA.

IV. GREEDY SEARCH ALGORITHM

In this section, we present the greedy search algorithm; follow that by studying the effectiveness of some of its parameters (the share values); and present a mathematical analysis of the effect of the share values on reaching an optimal solution.

A. *Greedy Search Algorithm*

The VDA uses a greedy search algorithm to decide on increasing and decreasing the amounts of resources allocated to VMs. The allocation is decided based on estimating the cost of the input workloads [7]. In each iteration of the greedy algorithm, 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. We note that by varying the *share* values, it is possible to obtain better solutions (less cost) than with fixed share values as used in [7]. This is illustrated in the next subsections.

B. *Effect of Share values in Greedy Search Algorithm*

In this section, we study the effect of the share values on the feasibility and speed of reaching an optimal solution. We noticed that for many problem instances, it is possible to reach an optimal result with more than one setting of the share values. In such cases, large values result in faster convergence to an optimal solution. In other words, the greedy search algorithm would benefit from using a share parameter value that is large, and thus allows for fast convergence, and at the same time gives a result that is as good as with smaller and slower-converging shares.

To illustrate the above-described effect of the share parameter, the greedy algorithm was run using two TPC-H workloads, which will be described later in Section VI, for 100 different share values starting from 0.1% to 10%. As shown in Figure 2, the same cost can be reached using different share values. Among these values, higher share values result in faster convergence. So, we need to obtain the optimal share value by exploring the relationship between the configurations calculated using theses share values.

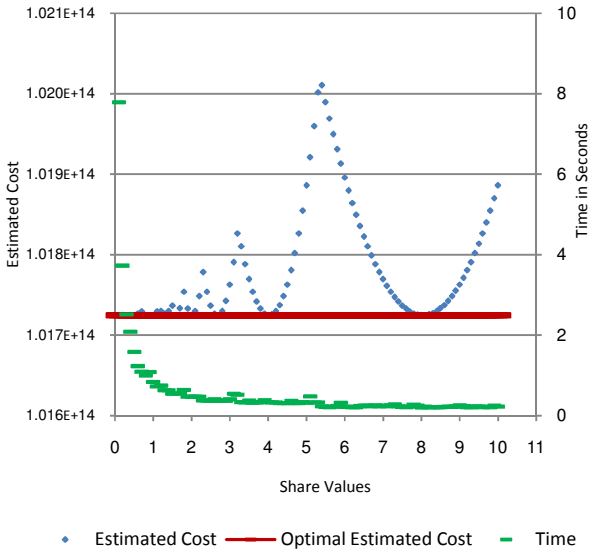

Fig. 2: Optimal cost can be reached using multiple share values. Experiment was done using two virtual machines running two different workloads

C. *Optimal Share Value*

For the greedy algorithm to reach an optimal configuration, the difference d_i between the default configuration def_conf_i and optimal configuration opt_conf_i of each VM_i has to be a multiple of the share value. The previous statement is a direct implication of how the greedy search algorithm works. Having a share value that is the greatest common divisor (GCD) of all the difference values d_i, ensures that the optimal configuration is reached and speeds up the convergence time. That is, the GCD of differences d_i is considered the optimal share value that results in the fastest convergence to an optimal estimated cost amongst all other share values. Hence, the GCD of differences for N VMs' optimal configurations is described as follows:

$$d_i = abs(def\_conf_i - opt\_conf_i) \quad (6)$$

$$optimal\_share = \text{GCD}(d_1, d_2, \ldots\ldots, d_n) \quad (7)$$

To illustrate the above definitions using an example, suppose that the optimal CPU configurations for four VMs are 30.8%, 22.2%, 29.3%, and 17.7%. For four VMs, and the default CPU configuration is 25%. Thus,

$d_1 = 5.8\,, d_2 = 2.8\,, d_3 = 4.3\,, d_4 = 7.3$

$and\ optimal\_share = \text{GCD}(5.8\,,2.8\,,4.3\,,7.3) = 0.1.$

So, 0.1 is the optimal share value which can cause the greedy algorithm to reach optimal configurations as fast as possible.

For another example, Figure 2 depicts that the optimal estimated cost was reached using multiple share values (0.1, 0.2, 0.4, 0.5, 0.8, 1.0, 1.6, 2.0, 4.0, and 8.0), and Figure 3 shows the CPU configurations that correspond to all share values. The default CPU configuration for the two VMs used is 50%, and the optimal CPU configurations are 58% and 42% for VM_1 and VM_2, respectively. That is,

$d_1 = 8\,, d_2 = 8\ and\ optimal\_share = \text{GCD}(8,8) = 8$

On the other hand, different share values result in different convergence time for the greedy algorithm. For example, the share values 0.1, 0.2, 0.4, 0.5, 0.8, 1.0, 1.6, 2.0, 4.0, and 8.0 result in reaching the optimal configuration after 80, 40, 20, 16, 10, 8, 5, 4, 2 and 1 iterations of the greedy algorithm, respectively. The largest share value, among all share values that guarantee reaching the optimal solution, results in the fastest running time of the greedy algorithm.

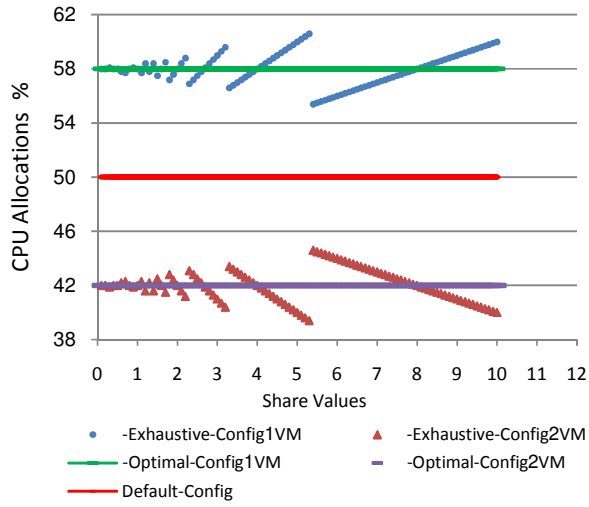

Fig. 3 VM Configurtions reached using different share values for two VMs. The optimal configuration is 58% and 42%.

D. *Effect of Search Space Size*

One can think of running the greedy algorithm exhaustively for a large set of share values and reporting the minimum cost that is reached over all the exhaustive runs. This exhaustive greedy algorithm would be affected by the size of the search space, the share values in this case. When the search space of share values is larger (i.e., with finer granularity of share value), the reached estimated cost is lower, and the running time is higher. Figure 4 depicts the effect of enlarging the search space of share values for two workloads running on two VMs. The estimated cost decreases with increasing search space until share value granularity is low enough to include a value that is a common factor of all allocation differences d_i, at which point increasing the search space merely results in increased running time without any improvement in estimated cost.

The result in Figure 4 is for one resource only (the CPU), and thus, the greedy algorithm has one share parameter. Increasing the number of resources results in exponentially larger search space for the share values and a corresponding increase in running time. We need another algorithm that reaches optimal share values efficiently even with more than one resource. In our proposed algorithm, we use the particle swarm optimization search technique to escape from local optima and to minimize the running time.

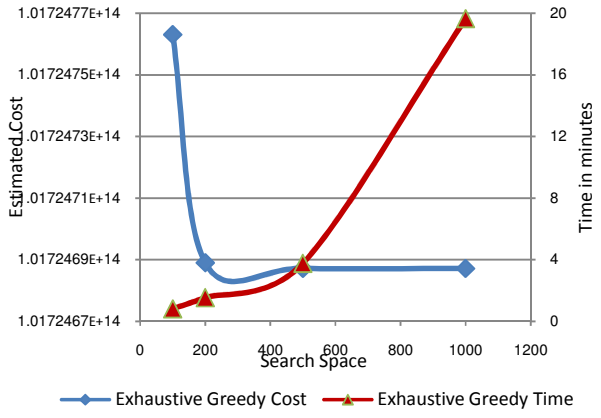

Fig. 4 Different Search Spaces for Exhaustive Greedy for two workloads

V. GREEDY PARTICLE SWARM OPTIMIZATION (GPSO)

In this section, we present our proposed hybrid search algorithm, namely the Greedy Particle Swarm Optimization (GPSO). We start by describing the particle swarm optimization and follow that by a description of the proposed hybrid algorithm.

A. Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is one of the modern evolutionary algorithms used to explore the search space of a given problem [25]. 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 candidate solution can be thought of as a particle within the swarm. The idea of PSO is based on introducing the observation of swarming movement to the field of evolutionary computation [29]. Each particle moves in a *D*-dimensional space (*D* represents the number of decision variables). Each particle is thus described by a tuple of D-dimensional vectors (X_i, V_i, P_i, G_i) , which (respectively) represent the current position,the velocity,the personal best position that the particle has achieved, and the global best position that is tracked by the entire swarm along each of the *D* dimensions.

Initially, the PSO algorithm chooses candidate solutions randomly. Then, each particle moves in randomly-defined directions based on best of itself and of its peers. During each iteration, the particles evaluate their positions towards a goal. They update their own velocities using a weighted function of globally best positions and their previous positions and then use these velocities to adjust their new positions. The used equations to update the velocity and position along each dimension for each particle are:

$$v_{id}(t+1) = wv_{id}(t) + c_1 r_1[pbest_{id}(t) - x_{id}(t)] + c_2 r_2[gbest_d(t) - x_{id}(t)] \quad (8)$$

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

where all parameters are represented in *d*th dimension at time t, $v_{id}(t)$ is the velocity of *i*th particle, $w.v_{id}(t)$ is the inertia component responsible for keeping the particle moving in the same direction, w(w ∈ [0.8, 1.2]) is an inertia weight that determines how much the previous velocity is preserved, $x_{id}(t)$ is the position of the *i*th particle, $pbest_{id}(t)$ is the personal best position for the *i*th particle, $gbest_d(t)$ is the globally best position, 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] [30]. The stopping criteria are that either the maximum number of iterations is reached or the minimum error condition is satisfied. An improved version of PSO, SSM-PSO, is used to avoid invalid-solution cases [31].

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 optima and refining global optimization. The societal and individual factors determine the ability of exploring and exploiting. The size of the swarm balances the requirement of global optimization and computational cost [32-34].

B. Greedy Particle Swarm Optimization (GPSO)

A hybrid of the heuristic greedy search and intelligent particle swarm optimization is proposed as a new algorithm to overcome the trapping into local optimum states away from global ones. We call this algorithm Greedy Particle Swarm Optimization (GPSO).

Figure 5 depicts the idea of the proposed GPSO algorithm. The main idea is that the GPSO algorithm uses PSO technique to tune the *share* parameter of the greedy algorithm. Whereas the greedy module enumerates resource allocations for the VMs based on the estimated cost of the given workloads, the PSO module sends to the greedy module candidate *shares* and VM configurations and receives updated VM configurations and the corresponding estimated cost for these configurations.

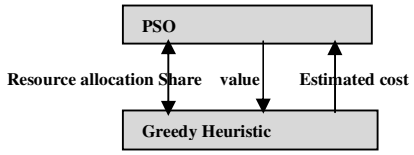

Fig.5 GPSO in VDA Enumerator Module

In this setting, the particles of the PSO module hold candidate values for the *share* parameter, and the dimensions represent the number of resources. 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. To evaluate each particle (share parameter value), the total of estimated costs is calculated using input workloads under candidate VM configuration as described in Eq.3. The total estimated cost serves as the fitness function for PSO.

The GPSO algorithm steps are 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 $X = (x_1, x_2, \ldots\ldots, x_n)$.

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.

VI. Experimental Evaluation

In this section, we present our experimental evaluation of the proposed GPSO algorithm. We start by describing the experiment setup and used

metrics. Then, an experiment to tune GPSO's swarm size is presented, followed by an experiment to show GPSO responsiveness to identical workloads. The ability of GPSO to escape from local minima is then demonstrated using an experiment, its responsiveness to variation of resource-intensiveness of workloads is demonstrated, and its speed is measured. Finally, a comparative study between GPSO and an exhaustive version of the greedy algorithms is described.

A. Experiment Setup

In our experiments we use PostgreSQL 8.4.8 as the DBMS running on 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 in its para-virtualization mode [35, 36]. By using Xen, we aim at mimicking at a very high granularity the Amazon EC2 cloud, which uses Xen virtualization as well.

In order to evaluate the effectiveness of the proposed GPSO algorithm, the TPC-H benchmark (with scale factor 1.0) is used to compare GPSO with the greedy search algorithm [7, 12, 37].

Our experiments compare the greedy search algorithm [2] to our proposed GPSO algorithm for allocating the CPU resource. Only the CPU-related parameters of PostgreSQL, namely *cpu_tuple_cost* and *cpu_operator_cost,* were calibrated. All experiments were done on a warm database. Each experiment was repeated ten times and averages are reported.

The *share* parameter, which is controlled by the PSO part of GPSO, has an upper bound of 10%. The size of the search space of the proposed GPSO algorithm is set to either 100 or 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. When the finest granularity of change is one tenth (0.1), a search space of 100 is in effect, which corresponds to the share values from 0.1 to 10. When the share ranges from 0.01 to 10 with a granularity of 0.01, a search space of 1000 points is created.

The GPSO algorithm was implemented in JAVA. This work focuses on one resource (the CPU), and thus, the particles in PSO has a single dimension. The input to the program is the number of shares. For the purpose of the experiment, the share parameter is initially randomly generated. The GPSO algorithm was set to terminate after reaching 50 iterations or when the incremental change in the total estimated cost across iterations becomes constant for five consecutive iterations. The greedy algorithm starts with equal allocations for all VMs and with a share parameter of (5%) [2].

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 [32]:

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

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

B. Performance Metrics

Four metrics were used to measure performance:

1) The total estimated cost of workloads (in terms of sequential page fetches) as estimated by the query optimizer of PostgreSQL for default configurations, greedy heuristic search algorithm configurations, exhaustive greedy algorithm configurations, and GPSO algorithm configurations.

2) Cost improvement, which measures relative performance [7, 24]. In this work, using two algorithms (greedy and GPSO), the formula for cost improvement is as follows:

$$improvement = \frac{Cost_{Greedy} - Cost_{GPSO}}{Cost_{Greedy}} \qquad (11)$$

where $Cost_{\text{Greedy}}$ and $Cost_{\text{GPSO}}$ are the total estimated cost under greedy and GPSO configurations, respectively.

3) Normalized cost improvement measures the amount of improvement in cost per unit of

running time. It is computed as the ratio between cost improvement and the average running time (in seconds) of the search algorithm (either greedy or GPSO).

$$normalized\_improvement = \frac{improvement}{avg(runtime)} \quad (12)$$

4) Runtime overhead measures relative runtime as follows:

$$runtime\_overhead = \frac{T_{GPSO} - T_{Greedy}}{T_{GPSO}} \quad (13)$$

where T_{GPSO} and T_{Greedy} are the total runtime of GPSO and greedy algorithms, respectively.

C. GPSO Experiments

We start by describing a set of experiments on the proposed GPSO algorithm.

1) GPSO Tuning

The size of the swarm in the PSO module of the GPSO algorithm was varied within the range [10-100] for two different workloads running on two VMs. Figure 6 plots the total estimated cost for all the ten experiments for different swarm sizes. It shows that the variation in total estimated cost decreases with increasing swarm size. In order to obtain a handle on the optimal swarm size, the normalized cost improvement (per unit of running time) was measured as will be discussed next.

The GPSO algorithm performance was evaluated by varying the swarm size within two search spaces, [0.01%-10%] and [0.1%-10%], to examine the characteristics of optimal swarm sizes. Figure 7 depicts the normalized cost improvement versus swarm size with two virtual machines.

According to the results in Figure 7, the highest normalized cost was reached with a search space of 100 points and a swarm size of 10. As a result, the following experiments used the 100-point search space with a swarm size of 10.

2) Identical workloads receive equal shares

The aim of this experiment is to verify that the GPSO algorithm equally partitions the shared CPU resource when the workloads are identical. Although the GPSO algorithm changed the value of the share parameter (from its initial value of 5%), the allocations and corresponding estimated cost did not change.

Figure 8 shows the estimated costs for up to 10 VMs that run identical copies of TPC-H Q1 query workloads. The graph plots the estimated costs reached by the greedy, GPSO, and the default configuration (equal allocations).

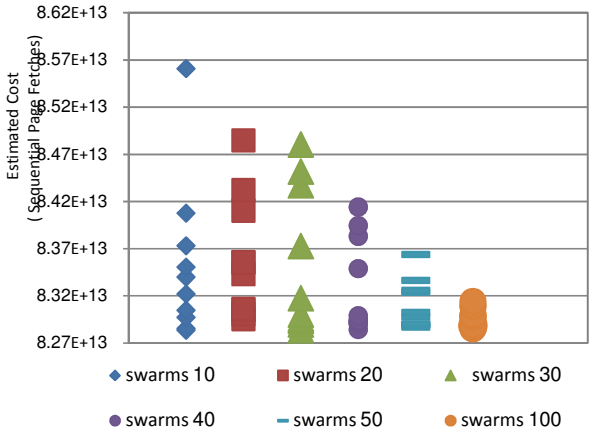

Fig. 6 Effect of swarm size on total estimated cost for two VMs.

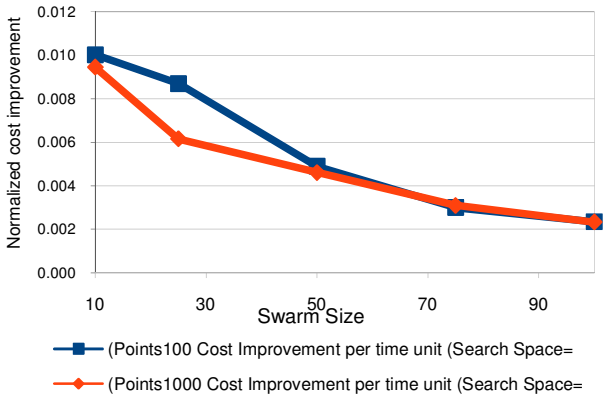

Fig. 7 Effect of swarm size on normalized cost improvement for two search spaces. Number of VMs = 2.

3) GPSO Escaping from local minimum

In this experiment, random TPC-H workloads were used. Twenty queries were generated using the same method described in [7]. Each workload consisted 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 [7]. Each VM had one workload. Each algorithm started

with 2 VMs and increased by 1 VM until it reached 20 VMs.

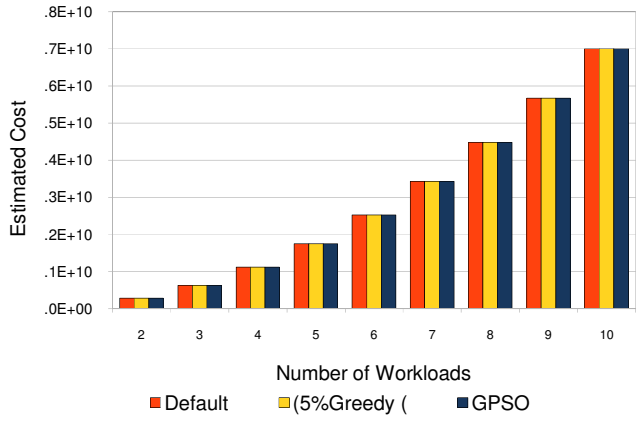

Fig. 8 Cost for identical workloads.

Figure 9 shows the total estimated costs for the two algorithms as compared to the default equal allocation. The estimated cost obtained by the GPSO algorithm was consistently lower than the estimated cost obtained by the greedy algorithm. In other words, the GPSO algorithm outperformed the greedy algorithm with respect to estimated cost. Figure 9 also shows that the cost improvement ratio fluctuated with the number of workloads with the most obvious change when moving from 19 to 20 workloads. At 20 workloads, the cost improvement decreased.

Figure 10 plots the same results as in Figure 9 with the cost improvement superimposed. It is noted that the greatest improvement happened when the greedy algorithm had a local optimum at 19 workloads. The greedy algorithm could not improve the resource allocation and stopped in the initial configuration (default configuration), whereas the GPSO algorithm was able to escape the local optimum by using another share value.

4) Responsiveness to varying CPU intensiveness

Figure 11 illustrates the varying CPU-intensiveness of workloads W1- W10 and the responsiveness of GPSO to their characteristics. GPSO was able to reallocate the CPU as new workloads jumped in. The fourth workload (W4) was non CPU-intensive and GPSO allocated to it the minimum CPU share. The eighth workload (W8) had the maximum CPU share, and it was a CPU-intensive workload.

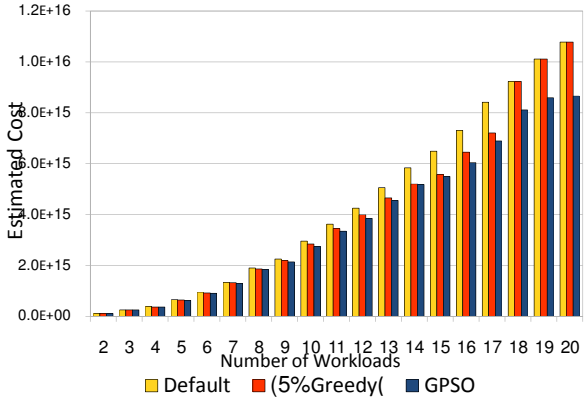

Fig. 9 Cost comparison for up to 20 random workloads.

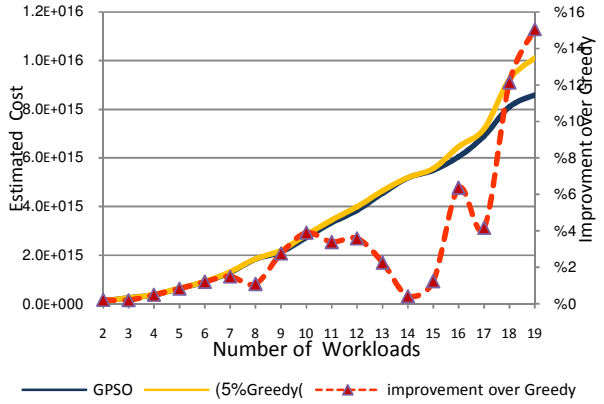

Fig. 10 GPSO cost improvement over greedy algorithm for up to 20 random workloads.

5) Runtime overhead of GPSO

Figure 12 shows the runtime overhead of GPSO. It shows that the GPSO algorithm was slower than the greedy algorithm. Also, it is noticed that the variation in GPSO runtime was due to the characteristics of the workloads not to the number of workloads alone.

Summary: According to our results, the GPSO algorithm achieved better allocations in terms of total cost at the expense of longer runtime. 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.

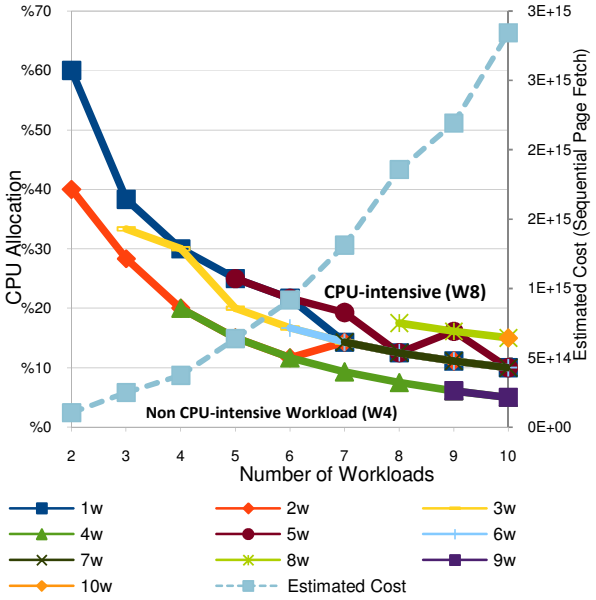

Fig. 11 CPU allocation for 10 workloads using GPSO.

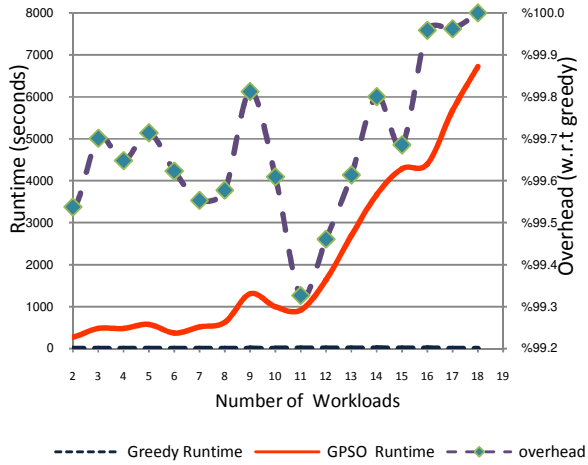

Fig. 12 Runtime overhead of GPSO (compared to greedy).

D. Comparative Study ofthe GPSO Algorithm

In this subsection, we present our comparative evaluation of the proposed GPSO algorithm against an exhaustive version (EG) of the greedy algorithm, which was described in Subsection D.

1) Estimated Cost

In this experiment, random TPC-H workloads were used. Fifteen queries were generated using the same method described in 3). The EG algorithm was compared to the default equal allocations, greedy algorithm which used 5% as share value, and the proposed GPSO algorithm. The EG algorithm with 50 iterations outperformed the other algorithms as shown in Figure 13. The EG algorithm was able to reduce the estimated cost even further by increasing the number of iterations to 100 as depicted in Figure 14. However, the EG algorithm suffers from an exponentially increasing running time with increasing problem size.

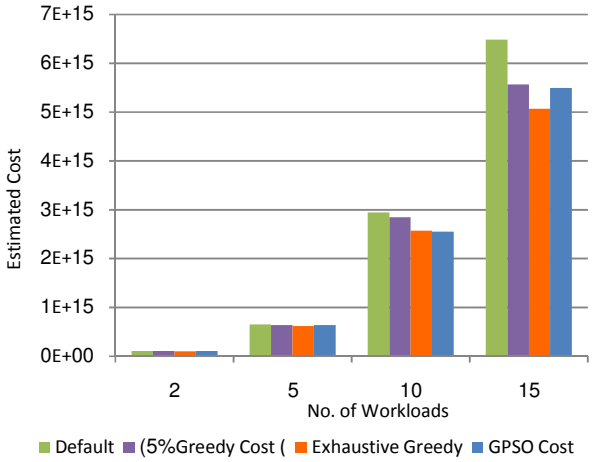

Fig. 13 Comparing GPSO, greedy, exhaustive greedy with 50 iterations, and default allocation w.r.t estimated cost.

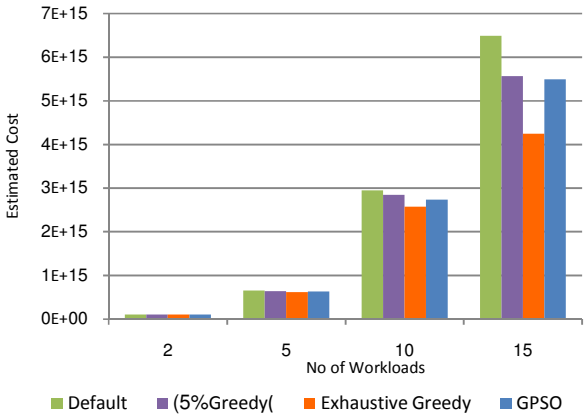

Fig. 14 Comparing GPSO, greedy, exhaustive greedy with 100 iterations, and default allocation w.r.t estimated cost.

2) Running Time

Figure 15 shows the runtime overhead of the compared algorithms. It shows that the EG algorithm was faster than the GPSO algorithm.

However, as noted in Subsection D, when search space grows in size, the running time of the EG algorithm grows exponentially. This trend is confirmed in the next experiment.

3) Search Space Size

The running time of the EG algorithm is more severely affected by search space size than the proposed GPSO. According to Figure 16, when the search space of the share values grows, the running time of the EG algorithm increased much faster than the proposed GPSO algorithm.

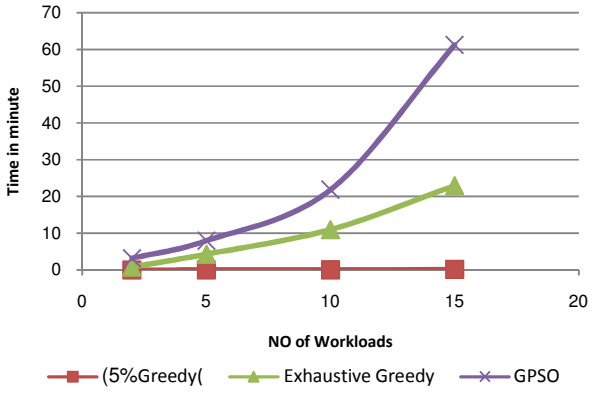

Fig. 15 Comparing GPSO, greedy, exhaustive greedy with 50 iterations, and default allocation w.r.t running time.

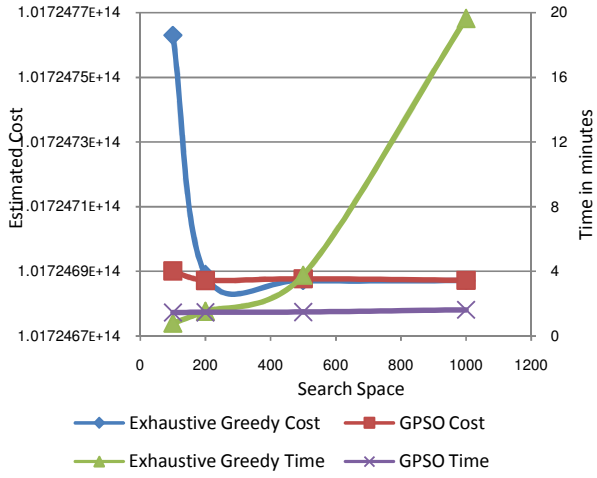

Fig. 16 The runnimg time of the exhaustive greedy algorithm grows much faster than GPSO with increasing search space size.

VII. Conclusions and Future Work

This work builds on and improves the Virtual Design Advisor, a framework for resource partitioning of physical machine resources among guest virtual machines running different database workloads. Our contribution is a search algorithm, namely GPSO, which is a hybrid between particle swarm optimization and greedy search. The proposed GPSO is motivated by the effect of the share parameter of the greedy algorithm on the feasibility and speed of convergence to an optimal configuration. Thus, the Exhaustive Greedy search algorithm studies the effectiveness of tuning the allocation of the shared resources and the effect of the share values of resources on the feasibility and speed of reaching an optimal solution. In other words, EG search algorithm benefits from using a share parameter value that is large, and thus allows for fast convergence, and at the same time gives a result that is as good as with smaller and slower-converging shares.

Using experiments with TPC-H benchmark on PostgreSQL database, we found that our GPSO algorithm was able to escape from local minima that otherwise trapped the greedy algorithm. Also, EG search algorithm was faster than the GPSO algorithm when the search space of the share values grows.

This work can be extended on many fronts: by considering other resources, such as I/O and network bandwidth, by incorporating QoS to provide more flexibility and control (e.g., integrating a penalty to reflect the cost of SLA violation), and by upgrading the PSO fitness function with a weighted factor of cost and time to strike a flexible balance between performance and speed.

References

[1] L. C. Qi Zhang, Raouf Boutaba, "Cloud computing: state-of-the-art and research challenges " *Journal of Internet Services and Applications,* vol. 1, No. 1, pp. 7-18, May 2010.

[2] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath, "Deploying Database Appliances in the Cloud.," *IEEE Data Eng. Bull.,* vol. 32, No. 1, pp. 13-20, 2009.

[3] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: a MapReduce framework on graphics processors," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 260-269.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, Bolton Landing, NY, USA, 2003, pp. 164-177.

[5] C. Computing. (2010). *Handbook of Cloud Computing*. Available: http://www.springerlink.com/index/10.1007/978-1-4419-6524-0

[6] A. A. Soror, A. Aboulnaga, and K. Salem, "Database Virtualization: A New Frontier for Database Tuning and Physical Design," in *Proceedings of ICDE Workshops (SMDB 2007)*, 2007, pp. 388-394.

[7] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath, "Automatic virtual machine configuration for database workloads," in *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, Vancouver, Canada, 2008, pp. 953-966.

[8] R. Sahal, S. M. Khattab, and F. A. Omara, "Automatic calibration of database cost model in cloud computing," in *Informatics and Systems (INFOS), 2012 8th International Conference on*, 2012, pp. CC-25-CC-34.

[9] S. M. K. Radhya Sahal, Fatma A. Omara, "GPSO: An improved search algorithm for resource allocation in cloud databases," in *Computer Systems and Applications (AICCSA), 2013 ACS International Conference on*, 2013, pp. 1-8.

[10] F. A. Omara, S. M. Khattab, and R. Sahal, "Optimum Resource Allocation of Database in Cloud Computing," *Egyptian Informatics Journal,* vol. 15, pp. 1-12, 2014.

[11] ([Apr.16, 2012 6:30 PM]). *PostgreSQL 8.3.18 Documentation*. Available: http://www.postgresql.org/docs/8.3/static/plpgsql.html

[12] (2001, 9-11-2011 1:29AM). *TPC-H Homepage*. Available: http://www.tpc.org/tpch/

[13] (2012, 7-11-2012). *What is Cloud Bursting*. Available: http://searchcloudcomputing.techtarget.com/definition/cloud-bursting

[14] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "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, pp. 289-302.

[15] P. a. L. Mitran, Long and Rosenberg, Catherine and Girard, André, "Resource Allocation for Downlink Spectrum Sharing in Cognitive Radio Networks," in *Proceedings of the VTC Fall*, 2008, pp. 1-5.

[16] G. Soundararajan, D. Lupei, S. Ghanbari, A. D. Popescu, J. Chen, and C. Amza, "Dynamic resource allocation for database servers running on virtual storage," in *Proccedings of the 7th conference on File and storage technologies*, San Francisco, California, 2009, pp. 71-84.

[17] D. Narayanan, E. Thereska, and A. Ailamaki, "Challenges in building a DBMS Resource Advisor," *IEEE Data Eng. Bull.,* vol. 29, pp. 40-46, 2006.

[18] A. Skelley, "DB2 Advisor: An Optimizer Smart Enough to Recommend its own Indexes," in *Proceedings of the 16th International Conference on Data Engineering*, 2000, p. 101.

[19] T. C. Air 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.

[20] S. S. u. Weihua Sheng, Maximilian Odendahl, Rainer Leupers, Gerd Ascheid, "Automatic calibration of streaming applications for software mapping exploration," in *Proceedings of the International Symposium on System-on-Chip (SoC)*, 2011, pp. 136 - 142.

[21] J. Rogers, O. Papaemmanouil, and U. C. W. Dietrich, "A generic auto-provisioning framework for cloud databases," in *Proceedings of the ICDE Workshops*, 2010, pp. 63-68.

[22] H. G. a. M. Pedram, "Maximizing Profit in Cloud Computing System via Resource Allocation," *Distributed Computing Systems Workshops, International Conference* vol. 0, pp. 1-6, 2011.

[23] H. G. a. M. Pedram, "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, pp. 324-331.

[24] S. Kong, Y. Li, and L. Feng, "Cost-Performance Driven Resource Configuration for Database Applications in IaaS Cloud Environments," in *Cloud Computing and Services Science*, I. Ivanov, M. van Sinderen, and B. Shishkov, Eds., ed: Springer New York, 2012, pp. 111-129.

[25] J. K. a. R. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks*, 1995, pp. 1942-1948.

[26] C. L. a. S. Yang, "Fast multi-swarm optimization for dynamic optimization problems," in *Proceedings of the Fourth International Conference on Natural Computation* 2008, pp. 624-628.

[27] A. Carlisle and G. Dozier, "Adapting Particle Swarm Optimization to Dynamic Environments," in *Proceedings of the International Conference on Artificial Intelligence (ICAI),* , 2000, pp. 429-434.

[28] F. v. d. Bergh and A. P. Engelbrecht, "A Cooperative approach to particle swarm optimization," *Trans. Evol. Comp,* vol. 8, pp. 225-239, 2004.

[29] T. Blackwell, "Particle swarm optimization in dynamic environments," in *Evolutionary Computation in Dynamic and Uncertain Environments*, S. Yang, Y.-S. Ong, and Y. Jin, Eds., ed: Springer Berlin Heidelberg, 2007, pp. 29-49.

[30] Y. S. a. R. Eberhart, "A modified particle swarm optimizer.," in *Proceedings of IEEE International Conference on Evolutionary Computation,*, 1998, pp. 69-73.

[31] Y. LIU, Z. QIN, and X. HE, "Supervisor-student model in particle swarm optimization," *IEEE Congress on Evolutionary Computation (CEC),* vol. 1, pp. 542-547, 2004.

[32] C. Jun-yi and C. Bing-gang, "Design of Fractional Order Controllers Based on Particle Swarm Optimization," in *Proceedings of the Industrial Electronics and Applications, 2006 1ST IEEE Conference*, 2006, pp. 1-6.

[33] X. Tao, W. Jun, and L. Xiaofeng, "An improved particle swarm optimizer with momentum," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007)*, 2007, pp. 3341-3345.

[34] R. Jinxia and Y. Shuai, "A Particle Swarm Optimization Algorithm with Momentum Factor," in *Proceedings of the Computational Intelligence and Design (ISCID), 2011 Fourth International Symposium on*, 2011, pp. 19-21.

[35] R. Rose, *Survey of System Virtualization Techniques*. Lisbon, Portugal: Theses (Electrical Engineering and Computer Science) MS non-thesis Research Papers (EECS), 2004.

[36] D. E. Williams, *Virtualization with Xen: Including Xenenterprise, Xenserver, and Xenexpress*: Syngress Publishing, 2007.

[37] S. W. Dietrich, M. Brown, E. Cortes-Rello, and S. Wunderlin, "A practitioner's introduction to database performance benchmarks and measurements," *Comput. J.,* vol. 35, pp. 322-331, 1992.