2012 IEEE Fifth International Conference on Cloud Computing

Remediating Overload in Over-subscribed Computing Environments

Long Wang, Rafah A. Hosn, Chunqiang Tang
Thomas J. Watson Research Center
IBM Corporation
Hawthorne, NY, USA
{wanglo, rhosn, ctang}@us.ibm.com

***Abstract*— Resource oversubscription brings the risk of resource overload. This paper proposes a mechanism to remediate overload without assuming there is always resource available for migration. A *work value* notion is introduced to compare importance of VMs, and the overload remediation problem is formulated as a variant of Removable Online Multi-Knapsack Problem. An algorithm is proposed to solve this optimization problem. The mechanism is implemented in a large commercial Cloud environment. Experiments and model-based studies demonstrate the effectiveness of the proposed mechanism in remediating overload and its performance in maximizing work values provided by computing environments (27% higher work values than the baseline algorithm in our study).**

Keywords- over-subscription; over-commitment; overload; quiesce; optimization; VM placement

I. INTRODUCTION

Resource oversubscription in Computing Cloud Systems is getting popular for IT services. Virtual Machines (VMs) are usually provisioned to host separate services in the Cloud systems. Traditionally these VMs are provisioned with all their allocated resources (e.g. memory and CPU) guaranteed in the physical resources. Resource allocation more than the amount that can be guaranteed by the physical resources, i.e. resource oversubscription, allows for more VMs to be hosted on a hypervisor [13]. As statistically VMs do not consume all the allocated resources, over-subscription increases resource utilization, and as a result, reduces cost of service provision.

However, oversubscription brings the risk of resource overload in the hypervisor before hosted VMs use up their allocated resources, and the resource overload, especially memory overload, largely degrades service performance and needs to be remediated.

An example is given by our experimental results illustrated in Figure 1. Eight 16GB-memory VMs run on a hypervisor with 64GB memory in this experiment. Each VM is a DayTrader server (a service of stock trading). We started a memory-consuming program in these VMs and continuously increased the memory consumption of this program. From Figure 1 we can see that, when each VM uses memory less than 8GB and the total memory used by these VMs is no more than 60.2GB (94% of physical memory), the performance of these VMs degrades slowly with the increase of the used memory. When the used total memory exceeds 94%, the performance has a big drop (53% worse from 430 reqs/min to around 200 reqs/min). The 102% total RSS sizes in Figure 1 is reported because certain memory shared between VMs via Kernel Samepage Merging (KSM) is double-counted in RSS.

A common approach to the resource overload problem is to migrate a couple of VMs from the overloaded hypervisor to other hypervisors with sufficient resources [4]. This approach assumes that *there are hypervisors with available resources in the Cloud system.*

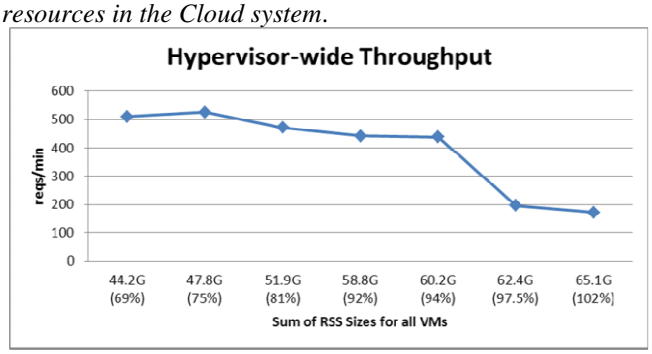

Figure 1: Throughput of VMs on an overloaded hypervisor

This paper studies overload remediation with the assumption of "resource-available hypervisors exist" relaxed. When no resource-available hypervisors exist, we have to discard certain computation temporarily, i.e. quiesce one or more *less important* VMs, to remediate the ongoing overload so that the rest *more important* VMs run normally; after a set of VMs complete their jobs and terminate normally or release resources, resources become available and quiesced VMs are resumed to finish their jobs. *Specifically, this paper proposes an approach that determines what operations (migration, quiesce or resume) should be taken for which VMs, in order to remediate resource overload in over-subscribed computing systems which may not have sufficient resources, while maximize the values of running services.*

Several essential challenges need to be addressed:

1) How to determine which jobs/services/VMs are more important compared with others?
2) How to formulate the problem?
3) How to solve the problem?

This paper first defines a notion *work value* to compute and compare the importance of different jobs/services/VMs. Based on the work value notion we formulate the problem as a variant of Multiple Knapsack Problem (actually it is a variant of Removable Online Multiple Knapsack Problem, ROMKP [20]).

This problem is NP hard. As there is not a solution to this variant of MKP problem yet in the literature, we propose an approximate algorithm ROWM (Remediating Overload with Work value Maximized) to select certain VMs to quiesce or

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.53

migrate for remediating overload while maximizing the work value provided by the computing environment.

Our solution is implemented and tested in a large commercial Cloud platform in IBM. Experimental results show that our solution effectively remediates overload and improves service performance by more than 40%. The results also show that during overload remediation important VMs remain running with a low-value VM being quiesced to release resource for the important VMs. Moreover, model-based studies are conducted to evaluate the performance of ROWM against a baseline algorithm and investigate ROWM's performance in different situations. The studies demonstrate that, ROWM achieves 27% higher work values than the baseline algorithm in overloaded scenarios. In non-overloaded scenarios ROWM does not demonstrate such large advantages over the baseline. Furthermore, higher frequency of checking whether quiesced VMs should be resumed improves the ROWM's performance in optimizing total work values.

In summary, this paper has the following contributions, in which ii) and iii) study the particular problem in a more general research background:

i) We study overload remediation without assuming there is always resource available for migration. As a result, quiesce and resume of VMs are considered to be options besides migration.

ii) This paper introduces the notion of *work value* to compare importance of VMs. Based on this notion, remediating overload while maximizing work values provided by computing environments is formulated as a variant of ROMKP.

iii) The ROWM algorithm is proposed to solve the formulated problem. This algorithm quiesces low-value VMs to free up resources for use of high-value VMs when resource is insufficient.

iv) This paper describes the implementation of the proposed solution in the real world, and presents the evaluation of the solution from experiments and model-based studies.

II. SYSTEM OVERVIEW

Figure 2 illustrates the basic architecture of the overload-remediation system in a Cloud. A number of hypervisors are interconnected. Images of VMs are placed on the storage shared across all hypervisors. So a VM can be started or resumed on any hypervisor. A monitor installed in each hypervisor periodically measures the resource usage of all the VMs on the hypervisor and reports the resource measurement to the *remediation center* component. When an overload on a hypervisor is detected by the monitor in the hypervisor, the remediation center has the knowledge on resource usages of all the hypervisors in the Cloud, and makes decisions of which VMs to be migrated or quiesced or resumed. Then the remediation center sends commands to corresponding hypervisors to issue operations. The message transmissions between the monitors and the remediation center are illustrated as the dashed lines in Figure 2.

As resource usage of a VM changes with time, and VM provision and deprovision occur from time to time, resource usage in the Cloud changes all the time. So the remediation center periodically checks whether sufficient resources become available for any quiesced VMs to resume.

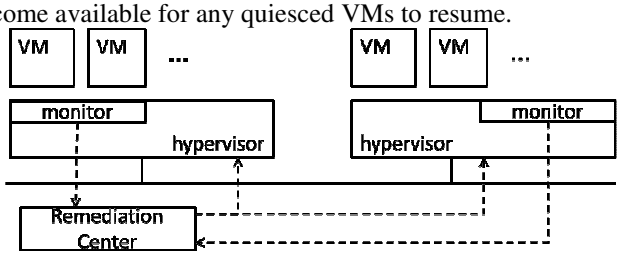

Figure 2: The basic overload-remediation system

III. WORK VALUE

Traditionally migration is used for remediating overload. When there is not a server with sufficient resources to host the migrated VMs, an additional server is powered on to host them [2]. In these solutions there is no or little discontinuation of the IT services provided by the migrated VMs.

However, when there are insufficient physical resources in computing environments (this situation may occur due to resource over-subscription), discontinuation of services provided by certain VMs is inevitable. One challenge to be addressed in these situations is to determine/evaluate which VMs are doing or will do less important tasks/services and can be quiesced to release resources for other VMs' normal execution.

We introduce a notion *work value* to define the importance of a VM. A number of VM placement algorithms in existing literature aim at optimizing utilities like performance, number of hosts, power, etc [1][3][5]. Our *work value* notion is different from such utilities in that work value aims at measuring i) potential loss of the services/tasks on a VM caused by the VM's being quiesced, and/or ii) impacts of a VM's being quiesced on execution of other VMs. These values are not measured as utility in traditional VM placement techniques because they do not quiesce VMs. For example, work value includes the expected work to be done by a quiesced VM if it were not quiesced, and the utilities in these techniques do not count this.

Potential loss of services/tasks. An intuitive embodiment of work value is revenue, or price. If price is not used as the work value, the work value may be indicated by service types. When a VM providing critical services like email and NFS is quiesced, it can cause great loss. We can estimate the potential loss directly from knowledge about a VM's services, or by leveraging analytical models if we do not have such direct knowledge. For example, if a VM provides backup service for a number of machines and performs the backup service regularly once a week, then the analysis of time series with seasonality [21] can be applied to detect the regularity of this VM's service and predict the criticality of this VM at specific time. Workload patterns can also be predicted [8][10] and help determine importance of a VM's work at specific time.

Besides using the knowledge about services, analytical models, and workload patterns to estimate the work value, administrators can specify their own computation of the

work value of VMs at any time t. An example of computing the work value is given below (the parameters V_1, V_2, V_3, and the function V_pattern() need to be properly set). Note that we just give example criteria here, which are not a complete set of all possible criteria for work values. Other specific criteria can be used for computing work values as administrators see proper.

estimate_work_value(a_vm, t) {
if a_vm is known that it provides critical services all the time
work_value(a_vm, t) = V_1 for any time;
else if a_vm is estimated to be running regularly-scheduled critical services
work_value(a_vm, t)=V_2 for scheduled time;
work_value(a_vm, t)=V_3 for other time;
else if the workload pattern on a_vm is predicted
work_value(a_vm, t)=V_pattern (workload_pattern, t) // with the workload pattern known, we can get its work value at time t
else
work_value(a_vm, t)=average amount of load in a_vm in its past history
}

Impacts on other VMs. Besides the loss of the quiesced VM's service, the quiesced VM may also impact other VMs which depend on the quiesced VM. For example, a stock trading service (*DayTrader*) in a WebSphere Application Server depends on a DB2 server, and the WAS and DB2 servers exist in two separate VMs. When the DB2 VM is quiesced, the WAS VM does not do any useful work. Due to the constraints of space, dependency between VMs is not discussed or addressed in this paper.

IV. PROBLEM FORMULATION AND SOLVING

This section discusses how to optimize the total work values of running VMs in dependency-free scenarios when we select certain VMs to quiesce, migrate, or resume. In dependency-free scenarios, a VM's work value is independent of other VMs' work values. Then the problem is a variant of multiple-knapsack problem (MKP), or more specifically, a variant of Removable Online MKP, because VMs are to be provisioned, to be removed from hypervisors, and to be relocated from one hypervisor to another, and VMs have their resource usages and work values changing.

Besides work values, costs of VM quiescing and migration are also considered when we handle this problem.

A Cloud computing environment has m hypervisors H_1, H_2, …, H_m. H_i has the amount of resource R_i (for simplicity of our analysis we assume there is one type of resource; however, our analysis can be extended to support multiple types of resources). There are n VMs VM_1, VM_2, …, VM_n.

- u_j *work value to be completed by VM_j*
- r_j *resource being used by VM_j*
- R_i *resource in H_i*
- $x_{i,j}$ *1: VM_j running on H_i; 0: otherwise*
- cq_j *cost of quiescing VM_j (e.g. …)*
- cr_j *cost of resuming VM_j (e.g. …)*
- cm_j *cost of migrating VM_j (e.g. …)*
- μ *threshold percentage for overload detection*

u_j, r_j, and $x_{i,j}$ change with time. So we denote them as a function of time t: $u_j(t)$, $r_j(t)$, and $x_{i,j}(t)$.

Formally, we are given

$$\begin{cases} \sum_{i=1}^{m} x_{i,j}(t) \le 1, for\ 1 \le j \le n \\ x_{i,j}(t) \in \{0,1\}, for\ all\ 1 \le j \le n, 1 \le i \le m \end{cases}$$

and the system is experiencing resource overload (when we remediate the overload) or not (when we resume quiesced VMs), i.e.,

$$\exists i, \sum_{j=1}^{n} r_j(t) * x_{i,j}(t) \ge R_i * \mu, 1 \le i \le m$$

,

or $\sum_{j=1}^{n} r_j(t) * x_{i,j}(t) \le R_i * \mu, for\ all\ 1 \le i \le m$.

We try to get $max(\sum_{i=1}^{m}\sum_{j=1}^{n}(u_j(t+1) * x_{i,j}(t+1)))$ by selecting $x_{i,j}(t+1)$ subject to,

$$\begin{cases} \sum_{j=1}^{n} r_j(t+1) * x_{i,j}(t+1) \le R_i * \mu, for\ 1 \le i \le m \\ \sum_{i=1}^{m} x_{i,j}(t+1) \le 1, for\ 1 \le j \le n \\ x_{i,j}(t+1) \in \{0,1\}, for\ all\ 1 \le j \le n, 1 \le i \le m \end{cases}$$

and,

$$\begin{cases} quiesce\ cost = \sum_{k=1}^{|YQ|}(cq_{yq_k} * \sum_{i=1}^{m} x_{i,yq_k}(t+1)) \le thrd_q, yq \in YQ \\ resume\ cost = \sum_{k=1}^{|YR|}(cr_{yr_k} * \sum_{i=1}^{m} x_{i,yr_k}(t+1)) \le thrd_r, yr \in YR \\ migration\ cost = \sum_{k=1}^{|YM|}\left(cm_{ym_k} * \sum_{i=1}^{m} x_{i,ym_k}(t+1)\right) \le thrd_m, ym \in YM \\ YQ = \{j | VM_j\ is\ to\ be\ quiesced\} \\ YR = \{j | VM_j\ is\ to\ be\ resumed\} \\ YM = \{j | VM_j\ is\ to\ be\ migrated\} \end{cases}$$

$thrd_q$, $thrd_r$, and $thrd_m$ are the thresholds for the costs of quiesce, resume and migration, respectively. Their values are determined by the system or specified by administrators.

Whether VM_j is to be quiesced, resumed, migrated, or unchanged is formally expressed by the relationship between $x_{i,j}(t)$ and $x_{i,j}(t+1)$, as shown below:

A= $\sum_{i=1}^{m}[x_{i,j}(t+1) - x_{i,j}(t)]$; B= $\sum_{i=1}^{m}[x_{i,j}(t+1) - x_{i,j}(t)]^2$

(1) VM_j is to be quiesced <=> A=-1, B=1
(2) VM_j is to be resumed <=> A=1, B=1
(3) VM_j is to be migrated <=> A=0, B=2
(4) VM_j is to be unchanged <=> A=0, B=0

A. Solving the problem

This variant of ROMKP problem is NP hard. There is no standard/classical solver for the problem in the literature. So we design the ROWM approximate algorithm, listed in Figure 3, to solve the problem.

There are two parts in the algorithm: the *overload-remediation* part that decides which VMs to quiesce/migrate when an overload occurs to a hypervisor, and the *periodic-resume* part that decides which quiesced VMs to resume

when resources become sufficient. Here is a brief description of the algorithm.

overload-remediation:

1. Q={the VMs planned to be quiesced by this algorithm; initially empty}; C = { the VMs consuming most resources on the overloaded hypervisor such that the rest VMs consume less than a threshold thrd2, e.g. 85%, of physical resource } (O(nlogn)) (n denotes number of VMs, k=|C|)

2. Sort all running VMs on all hypervisors according to their u/r ratio in the increasing order (O(nlogn))

3. Test if the VMs in C can be migrated to other hypervisors (VMs in Q considered to be quiesced), without causing resource usage more than thrd2 (O(km)) and with acceptable migration cost

- *A best-fit MKP heuristic algorithm applied during the test*
- *If successful, then the work is done and goto step 5; otherwise, goto step 4*

4. Q=Q+{the VM with the next smallest u/r value and with acceptable quiesce cost}; goto step 3

- *Complexity of steps 3 and 4: (O(nkm)) (m denotes number of hypervisors)*

5. Now the migration plan is successfully created as M. Remove from Q the VMs unnecessarily added into Q by step 4. <M, Q> is the final result. (O(n))

The algorithm complexity is (O(nlogn+nkm))

periodic-resume:

Periodically run the following steps:

1. Q={the VMs planned to be quiesced by this algorithm; initially empty}; C = {the quiesced VMs} (O(k)) (k=|C|)

2. Sort all VMs, both running and quiesced, on all hypervisors according to their u/r ratio in the increasing order (O(nlogn)) (n denotes number of VMs)

3. Test if the VMs in C can be resumed on the hypervisors (VMs in Q considered to be quiesced), without causing resource usage more than thrd2 (O(km)) and with acceptable resume cost

- *A best-fit MKP heuristic algorithm applied during the test*
- *If successful, then the work is done and goto step 5; otherwise, goto step 4*

4. Q=Q+{the VM with the next smallest u/r value and with acceptable quiesce cost}; goto step 3

- *Complexity of steps 3 and 4: (O(nkm)) (m denotes number of hypervisors)*

5. Now the resume plan is successfully created as M. Remove from Q the VMs unnecessarily added into Q by step 4. <M, Q> is the final result. (O(n))

The algorithm complexity is (O(nlogn+nkm)).

Figure 3: The ROWM algorithm for overload remediation

Upon an overload (i.e. resource usage exceeding a threshold *thrd1*), we identify a set of VMs on the overloaded hypervisor that have to be migrated or quiesced to remediate the overload (the set C in step 1 in Figure 3). Then we test if all the VMs in C can be migrated to other hypervisors by applying a heuristic algorithm for MKP. The test will fail when there is insufficient resource. Then we plan to quiesce the VM with the least work value-to-resource usage ratio in the entire Cloud (i.e. add the VM to Q in step 4), and try the test again. We continue this procedure until the test passes. Then the plan-to-be-quiesced VMs in Q are checked to pick out those VMs that have been unnecessarily planned to be quiesced (step 5 in Figure 3). After all these steps the final plan of migration and quiesce is obtained. Note that Figure 3 only presents the brief outline of the algorithm and does not cover all corner cases. The steps of *periodic-resume* part are similar to the description of the *overload-remediation* part above, except that we test whether all quiesced VMs in C can be resumed in step 3.

This algorithm has the time complexity of O(nlogn+nkm), where n is the number of VMs in the Cloud, k is the number of VMs to be migrated/resumed, and m is the number of hypervisors. It is polynomial time overhead, and is efficient for medium-size Cloud (hundreds of hypervisors and thousands of VMs).

V. IMPLEMENTATION AND EXPERIMENTS

Our solution is implemented as a MAPE (Monitor-Analyze-Plan-Execution) loop on a large commercial Cloud platform in IBM. Figure 4 depicts the architecture of our implementation.

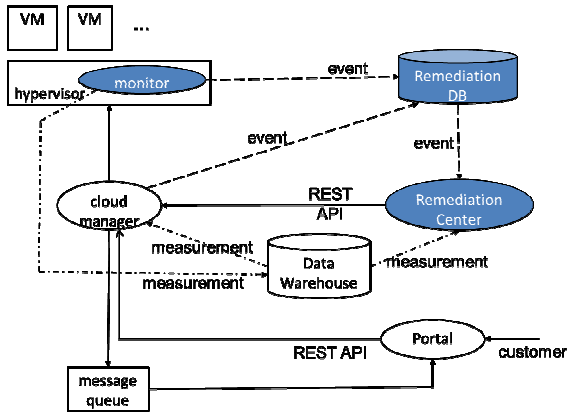

Figure 4: Implementation of the overload remediation system in the target Cloud

The shaded parts in Figure 4 are those components added to the target Cloud for supporting overload remediation. The rest parts in Figure 4 are components of the Cloud itself. The *cloud manager* is a software package, such as OpenStack [23], in charge of VM provision, deprovision, migration, and other management operations. The *data warehouse* stores measurement data which are periodically reported by the *monitors* in hypervisors. A customer can issue commands to the cloud manager for conducting desired operations via the portal. REST (REpresentation State Transfer) API is leveraged between the portal and the cloud manager for requesting specific operations in the cloud manager.

Compared with the basic overload remediation illustrated in Figure 2, we add a database, *remediation DB*, in the real implementation to support persistency. The database stores events generated by the hypervisor monitor (e.g. "resource overload" event, "overload remediated" event), status of the actions performed by the cloud manager (e.g. "success of migration", "failure of resuming a VM"), and the current states of all managed VMs (e.g. "normally running", "overloaded", "quiesced"). The remediation center is a process running on a separate computer. It periodically queries the remediation DB to process any new events received by the DB. When processing these events, the remediation center retrieves performance data from the data warehouse as these data are needed in the remediation algorithms (see Figure 4). When the decision of which VMs to be migrated/quiesced/resumed is made, commands are sent to the cloud manager to execute the decision.

In order to make the remediation center resilient to any crashes or other failures, we keep all states in the remediation DB. As a result, when a crash occurs to the remediation center, it is rebooted automatically and is able to continue the stopped processing successfully.

A. Experiments

We conducted experiments on the target Cloud platform after implementation of our overload remediation system. Figure 5 illustrates the results of an experiment in which quiescing of a VM mitigates the overload of the hypervisor and improves the throughput of all VMs on the hypervisor.

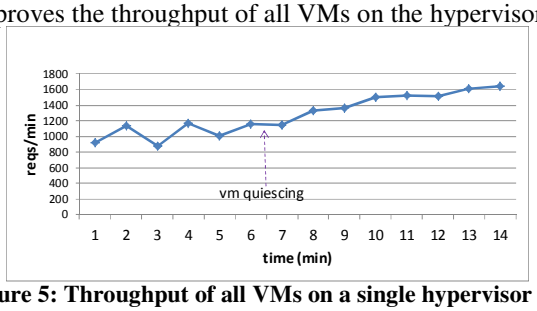

Figure 5: Throughput of all VMs on a single hypervisor

There is only one 64GB-memory hypervisor in this experiment. 18 VMs are on this hypervisor. Each VM is a stock-trading server (DayTrader) and uses memory around 3~4 GB. When a new VM is provisioned on this hypervisor, the total used memory becomes 62.3GB, exceeds 95% of the physical memory, and triggers our overload detection (overload is declared when the used memory keeps exceeding 95% for a number of consecutive minutes). As there is only one hypervisor in this experiment, two VMs are quiesced to bring the total memory below 54.4GB. Figure 5 records the total throughput of all the running VMs on the hypervisor, and clearly demonstrates that the total throughput gets much improved after the VM quiescing, from around 1150 reqs/min to around 1630 reqs/min (an improvement more than 40%).

We also conducted experiments on multiple hypervisors. The results show that a high-value VM on the overloaded hypervisor is migrated to another hypervisor while a low-value VM on the destination hypervisor is quiesced to release the resource for the high-value VM. This demonstrates the effectiveness of ROWM in optimizing work values. Due to the constraints of space in the paper, the details of the experiment are not presented here. More studies on the performance of ROWM in optimizing work values are in following sections.

VI. MODEL-BASED STUDY

Our overload remediation solution is implemented and tested in a large commercial Cloud system. In addition to real experiments in limited settings, we also evaluate our algorithms through model-based simulation (or study): a model is constructed to capture the behavior of the Cloud environment, different characteristics of workloads, and execution details of algorithms; simulation of the model allows us to perform systematic evaluations under a wide range of workload scenarios. Specifically, we are able to tune a number of model parameters in studying the effectiveness of our algorithm for optimizing the total work values of running VMs in over-subscribed computing environments.

The model in our study consists of the following sub-models:

- Sub-model for VM workloads, including i) the provision and deprovision of VMs, as well as ii) application load within these VMs.

The provision and deprovision of VMs are modeled by two random variables: T_{arr} as the inter-arrival time between two consecutive provisions of VMs, and L as the life of a VM (i.e. the time from its provision to its deprovision). As exponential distribution is popularly applied in modeling request arrivals and lives of sessions, we use exponential distribution for modeling T_{arr} and L with the exponential rates of λ_{arr} and λ_l, respectively.

The workload of a VM is characterized by two random variables: U as its resource usage and V as its work value. U and V are modeled in normal distributions with two parameters, *mean* and *variance*, for each random variable.

- Sub-model for the Cloud environment. This sub-model includes *N*, the number of hypervisors, and *R*, the amount of resources provided by each hypervisor, as parameters (identical hypervisors are assumed). To simplify the study, the costs of quiesce/migration/resume operations are not captured in the model.
- Sub-model for algorithm execution. To study the effectiveness and performance of our algorithm, we capture in this sub-model the algorithm steps described in Section IV. Moreover, we model a simple algorithm of VM selection for overload remediation as a baseline to compare with our algorithm in this study. The baseline algorithm is described below. There are three parameters in this sub-model for both our algorithm and the baseline algorithm: *P*, the period interval for checking whether any quiesced VM can be resumed, and the two thresholds used in the overload remediation algorithm, *thrd1* and *thrd2* (one for overload detection and the other for overload remediation; see Section IV for more details).

A. Baseline Algorithm

Like our algorithm in Section IV, the baseline algorithm consists of two parts: the *overload-remediation* part and the *periodic-resume* part. The baseline algorithm is quite straightforward and intuitive. When there is overload on a hypervisor, identify the set of VMs with most resource usage in this hypervisor such that, after migrating or quiescing these VMs the resource usage on this hypervisor is reduced to be under thrd2. Then for each VM in the selected set, migrate this VM to a target hypervisor when the migration is possible; otherwise, quiesce this VM. The details of the baseline algorithm are listed in Figure 6.

VII. STUDY RESULTS

Multiple studies are conducted based on the constructed model to evaluate the performance of ROWM in optimizing the work values of running VMs. Major results of these studies are summarized below before the details of the studies are presented:

overload-remediation:

1. C = {the VMs consuming most resources on the overloaded hypervisor such that the rest VMs consume less than a threshold thrd2, e.g. 85%, of physical resource}

2. Starting from the most resource-consuming VM in C, do the following for the VMs in C one by one:

select the hypervisor with the most available resource as the target;

if this VM can be migrated to the target hypervisor without causing overload (i.e. less than thrd2), then migrate it; else quiesce this VM

periodic-resume:

Periodically, run the following steps:

1. C = {the quiesced VMs which are sorted in the decreasing order of u/r}

2. Starting from the first VM in C, do the following for the VMs in C one by one:

Select the hypervisor with the most available resource as the target;

if this VM can be resumed on the target hypervisor without causing overload (i.e. less than thrd2), then resume it; else skip this VM for resume

Figure 6: The baseline algorithm for overload remediation

i) ROWM achieves much higher work values than the baseline algorithm in overloaded scenarios: in our simulation results, ROWM increases total work values of running VMs by 27% (115.09/90.96=127%).

ii) ROWM does not increase total work values of running VMs from the baseline in non-overloaded scenarios.

iii) The performance of ROWM in optimization improves as the period interval P decreases.

A. Parameters

Table 1 lists the values of the model parameters used in our studies, unless specific values are explicitly stated in the individual studies below.

Table 1: List of the model parameter values used in our studies

λ_{arr}	0.4	λ_l	0.01
U_{mean}	10 (GB)	$U_{variance}$	2.4
V_{mean}	8	$V_{variance}$	2.4
N	2	R	64 (GB)
P	5	thrd1	95%
thrd2	85%		

A concept of "time unit" is used in our model. The time unit may be a minute, an hour or another granularity of time. The time unit does not need to be explicitly specified for our model to work. λ_{arr} of 0.4 in Table 1 means that, the inter-arrival time between consecutive VM provision requests is in the exponential distribution with the mean value of the inter-arrival time being 1/0.4=2.5 time units. Similarly, the mean value of the VM life is 100 time units.

The work value of each VM is generated from a normal distribution with the V_{mean} and $V_{variance}$ specified in Table 1. The work value keeps unchanged throughout the life of each VM in our studies because i) we want to minimize variation in studied scenarios, and ii) certain embodiments of work value in the real world are constant, e.g. the unit-time price of VMs offered by IBM Smart Cloud Enterprise [22] does not change with time.

The resource usage of each VM at any time unit is generated from a normal distribution with the given values of U_{mean} and $U_{variance}$. Unlike the work value, the resource usage changes continuously during the life of the VM.

B. Performance of ROWM vs. performance of the baseline

The first study is to investigate the performance of our ROWM algorithm in maximizing work values of VMs running on the Cloud when there is resource overload. We simulate both the ROWM algorithm and the baseline algorithm, and compute the sum of the work values for all VMs running during the simulation. Figure 7 illustrates the sum along the timeline. Specifically, we do the sampling at each time unit, compute the average of 100 consecutive samples, and plot the average as a data point in Figure 7. So a unit of the horizontal axis in this figure is 100 time units.

In the same way we compute the average percentage of available resources in all the hypervisors during the simulation. The results are depicted in Figure 8.

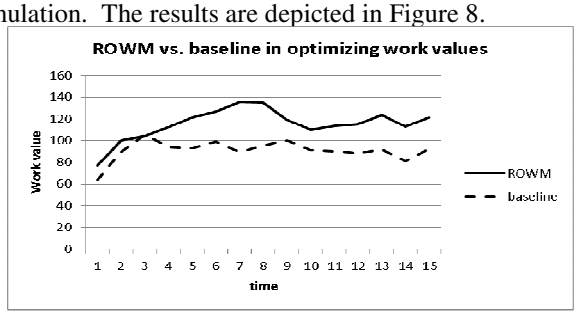

Figure 7: Performance of ROWM in optimizing work values in overloaded scenario (λ_{arr} =0.4)

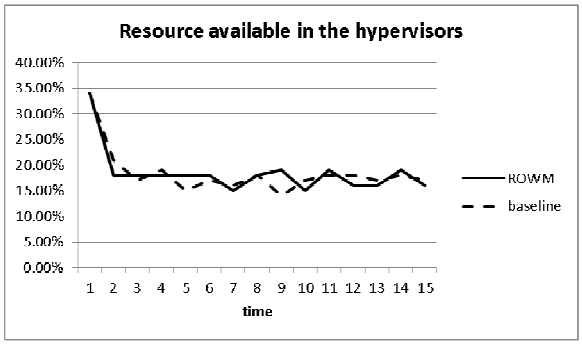

Figure 8: Available resources in the hypervisors in overloaded scenario (λ_{arr} =0.4)

The horizontal axis in Figure 7 and Figure 8 indicates the timeline from the beginning of the simulation, when there are no VMs on the Cloud yet, until a certain amount of time passes after the simulation results enter a steady state.

λ_{arr} is set as 0.4 in this simulation. The frequent requests of VM provision cause the hypervisors to be overloaded at most of time, as indicated by the curves in Figure 8 (the available resource percentage hits 15% and gets bounced back by the overload remediation as thrd2 is 85%).

Figure 7 demonstrates that ROWM achieves much higher work values than the baseline algorithm. The average of all data points in Figure 7 are 115.09 and 90.96, respectively, for ROWM and the baseline. ROWM increases the work values of running VMs by 27% (115.09/90.96=127%) in this scenario. When revenue is assigned as the work value, our algorithm brings 27% increase of revenue. This large performance improvement is achieved because ROWM optimizes the work values of running VMs by quiescing low-value VMs to release resource for usage of high-value VMs.

Non-overloaded scenarios. We have shown ROWM achieves a large performance improvement in overloaded scenarios. Figure 9 and Figure 10 demonstrate the work value and the available resource in non-overloaded scenario (λ_{arr} =0.1). The available resource is no lower than 20% in Figure 10, which indicates that there is no overload; as a comparison, Figure 8 shows the available resource is close to 15%, clearly indicating the overload as well as overload remediation.

From Figure 9 we see that ROWM does not increase the work values of running VMs from the baseline in non-overloaded scenario. Actually, the average of work values for the ROWM and baseline in Figure 9 are 61.49 and 61.31, respectively. This result is intuitive as there are few quiescings in non-overloaded scenarios.

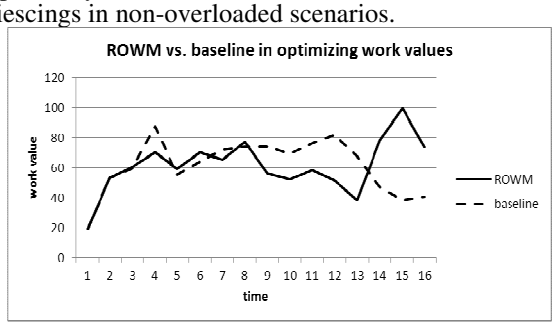

Figure 9: Performance of ROWM in optimizing work values in non-overloaded scenario (λ_{arr} =0.1)

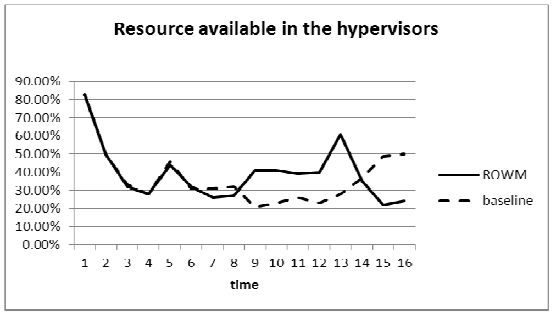

Figure 10: Available resources in the hypervisors in non-overloaded scenario (λ_{arr} =0.1)

C. Performance of ROWM with different P values

We studied how the selection of P, the period interval, affects the performance of the two overload remediation algorithms. Figure 11 shows the study results (the studied scenarios are overloaded ones as we aim to study the performance of overload remediation algorithms). We can see that the average work value increases as P decreases when ROWM is applied (from 110.60 for P=50 to 122.03 for P=1, around 10% increase in Figure 11). This is because more frequent checking for resuming quiesced VMs allows ROWM to have more chances to adjust which VMs to run/quiesce on the Cloud, and therefore, the result of the work values gets improved.

This result does not mean that the P value should be selected as small as possible in our implementation, because we still need to consider the high overhead incurred by the high frequency of invoking the periodic-resume procedure in the implementation. Therefore the selection of P requires a trade-off between the work value (such as revenue) and the overhead.

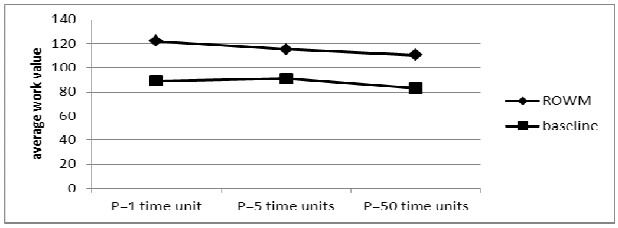

Figure 11: Impacts of the period interval (P) on the algorithms

VIII. RELATED WORK

This paper proposes a solution that remediates resource overload in over-subscribed computing environments, especially when there is not sufficient resource to host all running VMs.

Most of related work falls in the area of dynamic placement of VMs. Static placement of VMs, as a classical bin packing problem, focuses on computing the initial placement of VMs by using heuristic algorithms for global optimization. However, our core problem is to determine which VMs should be quiesced, migrated, or resumed at runtime.

Entropy [1] periodically runs algorithms for static placement of VMs and solves the global optimization problem to minimize the number of physical hosts and the number of migrations to reach the new placement. Bobroff et al. [2] also solve global optimization to provide guarantee to probabilistic SLAs. Other similar techniques solving optimization problems for dynamic placement of VMs can be found in the literature [3][5][17].

There are a number of techniques [4][12][11] that do not try to solve an optimization problem, but conduct greedy algorithms to compute the dynamic placement of VMs for different objectives. Black-box and gray-box [4] selects largest-resource-usage VMs on the most loaded hosts, and migrates them to the least loaded hosts to avoid SLA violation. Khanna et al. [12] identify the VMs with smallest resource usage on the host with SLA violated, and migrate them to the hosts with least available resource as long as SLA is not violated.

Besides the techniques that solve optimization problems or invoke greedy algorithms, there are also brutal-force techniques, such as [6], that conduct a first-fit search of all the combinations of possible placements.

All of these techniques assume there is sufficient physical resource in the Cloud. Whenever there is resource overload VMs can be migrated or swapped to remediate overload (new servers can be turned on as needed). We deal with situations when there are not sufficient resources: a number of VMs have to be selected and they are quiesced for a certain amount of time with their services/jobs unavailable during the quiesce.

We have to quiesce less important VMs to release resource for use of more important ones in resource-shortage scenarios. The notion *work value* introduced to indicate the importance of VMs is not considered by the prior techniques of dynamic VM placement. We try to optimize the total work values of running VMs. This poses a problem different from those addressed by the previous techniques, which do not quiesce any VMs and aim to optimize certain

utilities such as the number of physical servers, power consumption, SLA-associated performance, resource utilization, load balancing, etc.

Besides placement of VMs, researches on placement of application load are also related to our work. A method is proposed to dynamically start/stop application instances and adjust workload of these application instances on a cluster of servers in [7][9]. This work does not handle VMs, but handles application instances whose workloads can be flexibly adjusted across multiple hosts by a central dispatcher. Specifically, it optimizes resource utilization by matching the utility value of an application as its CPU-usage/memory-usage ratio, with the residual-CPU/residual-memory ratios of existing hosts. We see that the optimization problem and its solution in this related work are quite different from those discussed in this paper.

Other related work includes workload prediction techniques that are leveraged to proactively prevent or reactively remediate overload [2][8][10][14][18], as well as dynamic resource management in over-subscribed Cloud [15][16][19]. These techniques can be applied with our method as a complement in remediating overload.

IX. CONCLUSIONS

Resource oversubscription increases resource utilization. However, oversubscription also brings the risk of resource overload. This paper studies overload remediation without assuming there is always resource available for migration.

To decide which VMs to quiesce, we introduce the notion of *work value* to compare importance of VMs. Then we formulate the problem of remediating overload while maximizing work values provided by computing environments as a variant of ROMKP. The ROWM algorithm is proposed to solve the formulated problem.

We also implement the proposed mechanism in a real-world Cloud environment and evaluate the mechanism by means of experiments and model-based studies. Experiments show that our solution effectively remediates overload and improves service performance by more than 40%. Meanwhile, ROWM has good performance in optimizing total work values of the Cloud: it achieves 27% higher work values than the baseline algorithm in our study.

REFERENCES

[1] I. Hermenier, X. Lorca, J. Menaud, G. Muller, J. Lawall. Entropy: a consolidation manager for clusters, *Proc. Of ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments,* 2009.

[2] N. Bobroff, A. Kochut, K. Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations, *Proc. Of IFIP/IEEE Int'l Symp. On Integrated Network Management,* 2007.

[3] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting, C. Pu. Mistral: Dynamically managing power, performance, and adaptation cost in Cloud infrastructures, *Proc. Of IEEE Int'l Conf. on Distributed Computing Systems,* 2010.

[4] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif. Black-box and Gray-box Strategies for Virtual Machine Migration, *Proc. Of Int'l Symp. On Networked Systems Design and Implementation,* 2007.

[5] J. Xu, J. A. B. Fortes. A Multi-objective Approach to Virtual Machine Management in Datacenters, *Proc. Of ACM Int'l Conf. on Autonomic Computing,* 2011.

[6] Y. Xu, Y. Sekiya. Virtual Machine Migration Strategy in Federated Cloud, *IC2011*.

[7] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, A. Tantawi. Dynamic Placement for Clustered Web Applications, *Proc. Of Int'l Conf. on World Wide Web,* 2006.

[8] C. Isci, J. Hanson, I. Whalley, M. Steinder, J. Kephart. Predicting Virtual Machine CPU Demand for Effective Autonomous Resource and Power Management, *Proactive Problem Prediction, Avoidance and Diagnosis Conference,* 2009.

[9] D. Carrera, M. Steinder, J. Torres. Utility-based Placement of Dynamic Web Applications with Fairness Goals, *Proc. Of IEEE Network Operations and Management Symp.,* 2008.

[10] N. Roy, A. Dubey, A. Gokhale. Efficient autoscaling in the Cloud using Predictive Models for Workload Forecasting, *Proc. Of IEEE Int'l Conf. on Cloud Computing,* 2011.

[11] M. Andreolini, S. Casolari, M. Colajanni, M. Messori, Dynamic load management of virtual machines in a cloud architectures, *Proc. Of IEEE Int'l Conf. on Cloud Computing,* 2009.

[12] G. Khanna, K. Beaty, G. Kar, A. Kochut. Application Performance Management in Virtualized Server Environments, *Proc. Of IEEE/IFIP Network Operations and Management Symp.,* 2006.

[13] D. Williams, H. Weatherspoon, H. Jamjoom, Y-H. Liu. Overdriver: Handling Memory Overload in an Oversubscribed Cloud, *Proc. Of ACM SIGPLAN/ SIGOPS Int'l Conf. on Virtual Execution Environments*, 2011.

[14] S. Kounev, F. Brosig, N. Huber, R. Reussner. Towards self-aware performance and resource management in modern service-oriented systems, *Proc. Of IEEE Int'l Conf. on Services Computing,* 2010.

[15] J. Heo, X. Zhu, P. Padala, Z. Wang. Memory Overbooking and Dynamic Control of Xen Virtual Machines in Consolidated Environments, *Proc. Of IFIP/IEEE Int'l Symp. On Integrated Network Management,* 2009.

[16] A. Pokluda, G. Keller, H. Lutfiyya. Managing Dynamic Memory Allocations in a Cloud through Golondrina, *Int'l DMTF Academic Alliance Workshop on Systems and Virtualization Management*, 2010.

[17] H. N. Van, F. D. Tran. SLA-aware virtual resource management for cloud infrastructures, *Proc. Of IEEE Int'l Conf. on Computer and Information Technology*, 2009.

[18] N. Huber, F. Brosig, S. Kounev, Model-based Self-Adaptive Resource Allocation in Virtualized Environments, *Proc. Of Int'l Symp. On Software Engineering for Adaptive and Self-Managing Systems,* 2011.

[19] Memory Overcomittment Manager, https://github.com/aglitke/mom

[20] K. Iwama, S. Taketomi. Removable Online Knapsack Problems, *Automata, Languages and Programming, LNCS,* 2002, vol. 2380.

[21] J. D. Hamilton. Times Series Analysis, *Princeton University Press,* 1994.

[22] http://www.ibm.com/services/us/en/cloud-enterprise/

[23] http://openstack.org