SOPRAN: Integrative Workload Modeling and Proactive Reoptimization for Virtual Machine Management

JIAN ZHOU, LEI SHI and KIAN-LEE TAN
National University of Singapore

For a data center to operate effectively (i.e., meeting customers' Service Level Agreements (SLAs)) and efficiently (i.e., maximizing resource utilization), the virtual machines (VMs) must be carefully managed. In particular, as the resource demands of VMs change, the assignment of VMs to physical machines becomes sub-optimal. VM replication and migration provide a solution for dealing with dynamic workloads. However, as migrations are costly, an effective control policy is critical to avoid frequent migrations. Moreover, an agile decision making component is also important to reduce feedback latencies. In this paper we propose SOPRAN, a virtual machine management system leveraging an integrative workload model for the data center, which can dynamically adapt the assignment of VMs to physical machines to minimize resource consumption without sacrificing the SLAs. Different from existing trace-based methods for this problem, SOPRAN characterizes the dynamic workloads in the system using an integrative risk cube model, and approximates the workload demands with a *representative state set*. The optimal plan for each representative state is incrementally generated, forming the *switchable plan set*. At runtime, a two-phase re-optimization strategy matches the current system demand to the closest representative state and actuates the corresponding plan in the switchable plan set. At the same time, online monitors profile the actual demands and refine the risk cube to guarantee the model's accuracy. This modeling technique and optimization procedure based on it brings the great savings in optimization cost and migration frequency, and enables the high scalability of SOPRAN. We evaluated SOPRAN against the state-of-the-art IBM MFR algorithm. The results show that, with comparable resource consumptions, SOPRAN can achieve more stable SLA violation rate of no more than 4%, 80% lower migration rate, and save up to 90% optimization overhead.

Keywords: Data Center, Service Level Agreement, Virtual Machine Management

1. INTRODUCTION

Cloud computing is increasingly being deployed in commercial systems and research projects [Garfinkel et al. 2003]. Virtualization, as the underlying platform of cloud infrastructures, has significantly contributed to the advantages brought by cloud service [Ng et al. 2003]. For example, by letting multiple virtual machines (VMs) hosting applications run on the same physical machine (PM), the physical resource utilization can be increased, and the data center capacity can be subsequently increased; the power consumption, cooling cost and maintenance fee can be saved with better investment return [Jung et al. 2009; Kansal et al. 2010]; workloads running on the same PM can be safely isolated; and a programmable control layer provided by Virtual Machine Monitors (VMM) such as Xen [Barham et al. 2003] and VMware [Haletky 2007] enables performance surveillance and online automatic control of VM reconfiguration and migrations across PMs.

In particular, the VM migration function enabled by VMMs provides the flexibility of handling dynamic workloads with minimized resource consumption. However, VM migrations, even in the live mode, are not free of cost. A typical live migration operates in two steps: in the "pre-copy" step, the memory pages are copied to the destination machine while the VM is still operating; in the "stop-and-copy" step, the VM is stopped and the remaining pages are copied, after which the VM is completely migrated, and the service interruption happens in the "stop-and-copy" stage.

Authors' addresses: Jian Zhou, Kian-Lee Tan, National University of Singapore, NUS Graduate School for Integrative Sciences and Engineering, Centre for Life Sciences (CeLS), 05-01, 28 Medical Drive, Singapore; Lei Shi, Kian-Lee Tan, National University of Singapore, School of Computing, Computing 1, 13 Computing Drive, Singapore.

Although the service interruption (i.e., downtime in the second step) is only 60 ms as claimed in [Clark et al. 2005], the lengthy "pre-copy" stage can increase the application's response time by 50% to 200%, which causes significant impact on both the migrated VM and the co-located VMs on the same server [Verma et al. 2010; Jung et al. 2009].

To fully exploit the aforementioned benefits brought by virtualization while avoiding unnecessary performance degradation, it is crucial to develop proper automatic control mechanisms for virtual machine management (placement, communication and migration, etc.) in data centers [Bu et al. 2009; Wang et al. 2007]. Meanwhile, since the workload demands are always changing, it is very important to take into account the Service Level Agreement (SLA) fulfillment under such dynamic demands in the automatic control mechanisms [Abdelzaher et al. 2002]. For instance, if some workload's CPU demand experiences peak time and valley time periodically, the control mechanism must be able to detect the demand changes and react as soon as possible, so that when the demand increases, more CPU fraction can be allocated to this VM hosting the workload to satisfy the needs, and when the demand decreases, excess CPU fraction can be taken away from this VM to save the cost.

1.1 Objectives

Therefore, the objectives for a virtualized data center is to achieve an optimal system configuration so that it can "do more out of less". Firstly, the VMs must be dynamically reconfigured with sufficient resources, and be assigned to proper physical hosts (a.k.a. VM-PM allocation) to achieve the stipulated SLA, (which has the highest priority in a commercial data center [Fito et al. 2010]). Secondly, the system also needs to adaptively (re)optimize the hosts of VMs so that the dynamic online resource demands can be catered with minimized cost. And lastly it must also balance the optimization and migration overhead against the cost savings, so that frequent VM migrations will not happen, to avoid frequent service interruptions. [Kimbrel et al. 2004; Sapuntzakis et al. 2002].

To quantify the above objectives in evaluating VM management mechanisms, we use the following performance metrics in our experiments as the performance indicators, in order of decreasing importance from the perspective of data centers.

(1) Average SLA violation rate during runtime. This reflects the performance of a scheme in terms of meeting customers' SLA requirement;
(2) Average migration rate during runtime. This indicates a scheme's ability to provide system stability;
(3) Total re-optimization overhead during runtime. This captures the overhead of a scheme, which is important for the online performance and system reliability; and
(4) Average number of physical machines used during runtime. This gives an indication of the operational cost of the data center. In particular, machines that are not used can be powered down or set to stand-by mode to make the system more energy efficient.

These metrics will be introduced in the experiment section again.

1.2 Challenges

To achieve the above objectives, it is very helpful to understand the workloads' demand patterns, especially for long running ones. Various demand patterns can incur quite different optimization results. For example, some workloads are very stable in CPU demand over the time, while some others may display periodicity. Some may have high demand on CPU but relatively low on memory. And some comprehensive studies also point out that in real-world workloads the CPU demand changes more rapidly than memory resource [Gmach et al. 2008]. If these characteristics are taken into account to design the system, more effective and reliable deployment plans can be achieved by decoupling the complicated relationships in multiple resources allocation.

Next, since the system is expected to achieve automatic management with satisfactory performance, it is important for the control system to properly reconfigure VMs for the workloads with

dynamic demands during the runtime, which is to decide the fractions of physical resources (e.g. CPU, memory, I/O bandwidth) to be allocated to the VMs. Note that the amount of allocated resources should handle possible fluctuations in the near future to avoid scenarios where resources are repeatedly being added to or remove from a VM.

Moreover, to ensure an optimal workload consolidation solution for the purpose of minimized total resource consumption servers with changing demands, it may be necessary to re-assign the VMs' hosts at runtime. In our context, we call the residence-host relationship of all VMs and PMs currently in the data center a **plan**. While an exhaustive enumeration of all candidate plans will end up with an "optimal" plan, the optimization overhead would grow exponentially with increasing number of VMs. As a result, the overall effect may harm the online agility of the control system.

During runtime, the performance monitor will detect SLA violations of workloads and feedback to the system coordinator. Once the violation rate exceeds a certain predetermined threshold, reoptimization will be triggered. Intuitively, a tighter threshold means more aggressive reoptimizaion and higher migration rate, which may incur more service interruption; while a more relaxed threshold may cause higher SLA violation rate. Therefore, a *robust* plan is preferred in order to trade-off between performance and optimization overhead, and to guarantee that the deployment we make can be effective enough in the foreseeable future.

1.3 Existing Methods

Most of existing solutions for this virtual machine management problem follow a "modeling-optimizing-actuating" framework, yet certain strategies are employed for specific workloads such as web application [Karve et al. 2006; Abdelzaher et al. 2002], database workloads [Seltzsam et al. 2006; Soror et al. 2008] or multi-tier workloads [Padala et al. 2009; Jung et al. 2009]. A more detailed literature review is presented in Section 2.

1.4 Contribution

In this paper, we propose SOPRAN, a trace-based virtual machine management system using an integrative workload model for the data center. SOPRAN has the following distinguishing features:

(1) SOPRAN discretizes and approximates each workload into a small set of coarse-grained resource demand (e.g., low, moderate, high). In this way, a *risk (hyper-)cube model* (in a hypercube space) can be derived to capture the integrative system load. Essentially, each point in the model corresponds to a resource demand state of the data center where the ith dimension captures the resource demand of the ith workload.

(2) Our risk cube model is compact without complex formulas and parameters, and can easily handle a much broader range of dynamic workloads. This is in contrast to some modeling methods which require complex parameter inference and refinement processes. Thus, SOPRAN greatly simplifies the optimization process, and so improves the online performance in terms of system latency.

(3) SOPRAN pro-actively maintains a set of *switchable plans*, one for each point in the risk cube (which is defined in section 5.2). Each such plan is essentially the optimal VM-PM allocation plan with regard to the given system load corresponding to that point. Benefiting from the reduced dimension of the workload model, the plans in this switchable plan set are effective and robust in handling a set of conditions represented by the corresponding states in the model.

(4) SOPRAN adopts a two-phase re-optimization strategy. In the first phase, it monitors the actual runtime workload and matches it to the corresponding point in the risk cube. In the second phase, it determines the best allocation strategy to use. When the current system load falls within that captured by the risk cube, an optimal plan is readily available for reuse. This keeps the optimization overhead low. When the actual load is not captured by the risk cube, it has to be refined to reflect the current system load more accurately (see the next point).

(5) As workload demand changes with time, there may be a need to refine the risk cube and re-optimize the corresponding switchable plan set. Based on the workload profile, SOPRAN periodically recomputes a *tentative* risk cube. This tentative risk cube will replace the existing risk cube once SOPRAN detects that the discrepancy between them is sufficiently significant. This mechanism combines both proactive and reactive advantages, to maintain the accuracy of the risk cube model, so as to ensure SLA fulfillment with minimized cost.

(6) The switchable plan sets are generated incrementally during runtime when the corresponding representative states are matched up, to amortize the optimization overhead and further reduce the system latency. Also, the low optimization overhead and small migration rate keep the entire system more stable without frequent service interruption, guaranteeing a good system scalability.

1.5 Paper Organization

The rest of this paper is organized as follows. Section 2 presents an overview of related works in comparison with our SOPRAN. In Section 3 the background of this paper is summarized, and in Section 4 we identify the optimization problem. Section 5 gives the system architecture and detailed description of SOPRAN. Section 6 reports results of a performance study. And we conclude this paper in Section 7.

2. RELATED WORK

The workload deployment strategies can be roughly divided into two categories: Under *static deployment*, each VM is assigned a fixed amount of resources (typically the peak demand) and a pre-determined VM-PM allocation. This system configuration will be in use for a long period, e.g. several months. Static deployment simplifies the system design. However, the resources are typically not well utilized and the system cannot cope well with changing workloads. In *dynamic deployment*, with the help of VM migration function, the system periodically re-optimizes the VM-PM mapping according to the runtime feedback to achieve optimal performance. However, it may incur high optimization overhead when the system scales up.

2.1 Static Deployment

Static deployment schemes pre-determine the configuration and placement of the VMs. This allocation will then be used for a long period (e.g. several months) without changes. This decision is often done offline [Urgaonkar et al. 2002]. Static deployment makes the system easy to manage and scale up, but VMs are very likely to be over-provisioned in order to satisfy peak time demands. Obviously, for those workloads who rarely reach their peak times, the corresponding VMs are severely under-utilized even when other VMs in the same cluster are hungry for resources.

2.2 Dynamic Deployment

For a more effective solution to the automatic virtual machine management/scheduling problem, dynamic deployment strategies are preferred. Dynamic deployment has been studied by both academia [Karve et al. 2006; Abdelzaher et al. 2002; Seltzsam et al. 2006; Soror et al. 2008; Padala et al. 2009; Jung et al. 2009; Bobroff et al. 2007; Fito et al. 2010; Wang et al. 2007; McNett et al. 2007; Xu et al. 2007; Verma et al. 2008; Gmach et al. 2008] and industry (such as HP Service Integration Environment (SIE) and IBM System Director). And most of the existing solutions follow a "modeling-optimizing-actuating" framework, which corresponds to the main challenges in automatic virtual machine management as discussed in section 1.2.

2.3 Dynamic Deployment with Workload Differentiation

Within this dynamic deployment scope, different strategies are employed for specific workloads such as web application [Karve et al. 2006; Abdelzaher et al. 2002], database workloads [Seltzsam et al. 2006; Soror et al. 2008] or multi-tier workloads [Padala et al. 2009; Jung et al. 2009]. For

example, Soror et al. discussed how virtualized platform can benefit database workloads [Soror et al. 2008]. They discussed the tuning problems caused by running database systems inside the VMs, and proposed to reconfigure VMs at runtime for database workloads. They modeled the performance of a VM with a certain configuration using a "what-if" model based on the database query optimizer, and identified this problem as an optimization of resource allocation among several VMs with some performance constraints. This solution works well for database workloads, however, it cannot be generalized to non-database workloads, since the model relies on the query optimizer in database system.

2.4 Dynamic Deployment with Resource Differentiation

Besides the differentiations in workloads, some proposed methods emphasize on specific resources, such as CPU, memory, I/O bandwidth, etc. For example, the authors in [Bobroff et al. 2007] focused on CPU resource allocation during online reconfiguration, and some other groups studied the I/O performance in virtual machine scheduling [Kim et al. 2009; Ongaro et al. 2008; Gulati et al. 2009]. Since the workloads' requirements for different resources can be correlated, the overhead relationship between different resources, e.g. CPU vs. I/O and memory, has also been investigated to provide guidelines for building cost models [Jang et al. 2011; Cherkasova and Gardner 2005].

Our proposed SOPRAN system is applicable to multiple resource types, although we use CPU resource demands in our risk cube model as an illustration. The rationale is that in real-world workloads the CPU demand changes more frequently than memory resource [Gmach et al. 2008], and memory resizing is not as agile as CPU reconfiguration.

2.5 Dynamic Deployment: Trace-based Modeling Methods

For methods that focused on CPU resource reconfiguration in shared-resource environment, many of the existing methods adopt a trace-based modeling mechanism [Bobroff et al. 2007; Gmach et al. 2008; Rolia et al. 2004; Rolia et al. 2005; Seltzsam et al. 2006]. Workload traces have been shown to be useful especially when the workloads display repetitive patterns in resource demands over a long period. Therefore workload trace can reflect future demands well. In SOPRAN the risk cube model is also built on the basis of workload traces which are kept up-to-date.

Generally the trace-based methods model the workload demands using the historical traces of the workload, and predict the future demands according to the model, which will be used for resource resizing and allocation in the optimization phase. For example, the MFR system [Bobroff et al. 2007] models the workload's CPU demands into autoregressive time series first, and then predicts the future demand for certain time interval as the input of plan optimization. This algorithm fully exploits the characteristics of time series models. However, it is not generally applicable to workloads that cannot be accurately modeled with time series.

Three findings of the workload modeling phase are: (1) as the model's granularity increases, the model's accuracy increases, but the modeling overhead such as parameter inference and refinement overhead also increase. After the model's granularity reaches certain level, the gain in model accuracy may not worth the increased overhead; (2) as the model's granularity increases, the optimization complexity also increases, because the dimension of the searching space in the optimization algorithm grows exponentially with the cardinality of the model. And similarly, the model's granularity should not be too fine to make up the optimization overhead; (3) most of the aforementioned modeling methods model the individual workloads running in the system, yet they fail to capture the overall workload pattern in the system. This is fine for systems with few machines, but when the system grows to 1000 VMs, the scalability of the control system may be degraded due to the trivial model technique. Based on these findings, in SOPRAN we adopt a coarse-grained model for each workload, and maintain an integrative system workloads model called the "risk cube model" that improves the model's abstraction ability for the entire system's workload pattern, to enable a coarse-grained online adjustment for VM reconfiguration and reallocation.

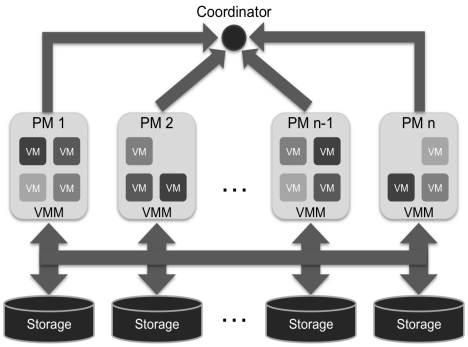


Figure. 1: A data center with virtualized platform

There are also non-trace-based academic systems such as Sandpiper [Wood et al. 2007]. These systems can also work well in its proposed environment.

2.6 SLA-aware Dynamic Deployment

The objectives of virtual machine management in a data center are not only to achieve a minimum resource consumption but also the fulfillment of Service Level Agreements. In particular, for the Infrastructure-as-a-Service framework, the service provision level is critical to users, and can impact the service providers' standpoint in the cloud market. Therefore, SLA-aware virtual machine scheduling has also been investigated by some groups of researchers [Bobroff et al. 2007; Fito et al. 2010].

In SOPRAN, we also use online performance feedback as an indicator of SLA fulfillment, and to adjust the resource allocation or VM placement whenever the VMs' performance becomes degraded, in order to provide users with high quality of service.

2.7 Energy-aware Dynamic Deployment

Besides performance objectives, the energy and power consumption concerns are becoming increasingly important.

To ensure energy efficiency, energy and power aware virtual machine management strategies have also been proposed [Kansal et al. 2010; Verma et al. 2008].

In particular, since virtual machine migration is costly, migration-aware virtual machine management in data centers are being pursued by several groups of researchers. For example, Akshat Verma et al. proposed *pMapper* as a trade-off between power and migration cost and application performance in virtualzed systems [Verma et al. 2008]. Some other works also sought to minimize migration opportunities during runtime [Kimbrel et al. 2004; Sapuntzakis et al. 2002; Wood et al. 2007; Xu et al. 2007]. In SOPRAN, the compact system workload model uses representative states to cover similar system conditions, and render a reduced solution space for the optimization problem, so the probability for VM migrations has been significantly reduced.

3. BACKGROUND

Currently, many commercial vendors adopt the hardware architecture depicted in Figure 1 for data centers [Brantner et al. 2008].

There are two types of physical nodes: computing nodes and storage nodes. Computing nodes are mainly for providing computation power and do not hold data, and VMs can only be hosted by computing nodes. When migration is invoked, VMs do not need to carry data with them. All the data and logs are stored in the storage nodes. Some of the computing nodes may be set to standby mode or even powered down, while storage nodes are active most of the time.

All computing nodes are interconnected as a cluster with a centralized coordinator, and all storage nodes are interconnected to build an "infinite" storage pool, which can be expanded

Symbol	Meaning
$P =< P_1, ..., P_M >$	Physical machines
$V =< V_1, ..., V_N >$	Virtual machines
$W =< W_1, ..., W_N >$	Workloads
$R =< R_1, ..., R_M >$	PMs' resource capacity
$D(t) =< D_1(t), ..., D_N(t) >$	Resource demands at t
$S(t) =< S_1(t), ..., S_N(t) >$	Resource allocated at t
τ	Re-optimization interval
S	A time section of $n \times \tau$
RSS	Representative State Set
SPS	Switchable Plan Set

Table I: Notations of the system model

when necessary by adding more storage nodes without being perceived by the VMs. On top of each computing node there is a VMM layer between the hardware and the operating systems, which handles VM creation, resource allocation, VM migration, performance monitoring, etc. Mature VMMs such as Xen and VMware can achieve high control accuracy and low overhead. Multiple VMs with their (fractional) resource allocation (CPU, memory, bandwidth, etc.) can be created by VMM, and each guest OS can run as if it is assigned dedicated hardware.

The computing cluster is connected to the storage pool using a fast communication bus. Every time the computing nodes need to exchange data with the storage, they will issue I/O requests which are queued and processed accordingly. The coordinator can communicate with all VMMs to coordinate the tasks and schedule the I/O requests.

4. PROBLEM IDENTIFICATION

SOPRAN is designed based on the above data center architecture. Suppose there are M physical computing nodes with a data storage pool, and N VMs with workloads running on the M PMs. The notations of this system are listed in Table I.

The objectives of SOPRAN are to make sure that the VMs must be allocated sufficient physical resources to achieve the stipulated SLA under dynamic workload demand, and also to adaptively reconfigure VMs and possibly dynamically change the host of each VM, so that the resource consumption can be minimized through consolidation.

At the same time, the adjustment granularity of resource allocation during the optimization must be traded off against the gains from this dynamic consolidation, so that there will not be too frequent VM migrations across PMs, which itself consumes resources and interrupts the applications hosted on the VMs. If the migration is too frequent, the system stability is affected, and may render a poor scalability due to the associated overhead in each migration operation.

The optimization problem of VM resource allocation and placement problem can be identified as a constrained programming problem, in which the workload demands are the inputs and the VM-PM mapping plan is the output.

4.1 Control Variables

We use a matrix m_{MN}, called the *plan matrix*, to represent the multiple-to-one mapping between N VMs and M PMs, which indicates the resident-host relationship between them. If VM V_j resides on PM P_i, the i^{th}-row and j^{th}-column element m_{ij} in the plan matrix equals to 1; otherwise, $m_{ij} = 0$. Intuitively, if some PM P_m is not hosting any VM, the elements in the m^{th}-row of this matrix will be all zero, and we call this row a "zero line". The number of non-zero lines in the matrix is the number of used PMs in the system. All the elements in this matrix

constitute the control variables of the optimization problem.

$$m_{MN} = \begin{bmatrix} m_{11} & m_{12} & ... & m_{1N} \\ m_{21} & m_{22} & ... & m_{2N} \\ ... & ... & ... & ... \\ m_{M1} & m_{M2} & ... & m_{MN} \end{bmatrix} \quad (1)$$

If V_j resides on P_i, $m_{ij} = 1$; else, $m_{ij} = 0$.

4.2 Constraints

There are some constraints in the VM resource allocation and placement optimization problem. For example, the amount of resource allocated to each VM in a plan should be greater or equal to the demand of this workload at time t, as shown in Formula (2):

$$D_j(t) \leq S_j(t), 1 \leq j \leq N \quad (2)$$

The summation of all the VMs' resource S_j on the same PM P_i should not exceed the capacity of this host PM, as shown in Formula (3):

$$\sum_{j=1}^{N} m_{ij} \cdot S_j(t) \leq R_i, 1 \leq i \leq M \quad (3)$$

In each plan, one VM can only be hosted by one PM at a time, meaning that there must be strictly a single "1" in each column of plan matrix, as represented by Formula (4).

$$\sum_{i=1}^{M} m_{ij} = 1, 1 \leq i \leq N \quad (4)$$

4.3 Objective Function in Optimization

The objective of this optimization problem is to minimize the average number of PMs used during runtime (every PM holds as many VMs as possible), subject to the demands of each VM with respect to the above constraints (Formulas 2), (3) and (4). Therefore, we have

$$cost(m_{MN}) = rank(m_{MN}) \quad (5)$$

where m_{MN} is the matrix of a plan that is to be evaluated, and the *rank* of a matrix A is the number of linearly independent rows or columns of A. Here, since every plan is a matrix, the number of PMs used equals to the rank of the matrix by definition.

4.4 A 0-1 Optimization Problem

The optimization problem has now been packaged into a 0-1 programming model. We use first-fit heuristic algorithm to solve it and obtain the best solution. Experiments have shown that the best solution from first-fit heuristic is very close to the optimal one from dynamic programming algorithm while the latter would cost much longer time to solve.

5. DESIGN OF SOPRAN

In Section 2.5, we have introduced the IBM MFR method which establishes a deterministic model for the workloads, uses the inner components of the model to predict future performance degradations, and proactively makes adjustment. This model is accurate, yet the computational cost and optimization cost may be significant.

Instead, SOPRAN discretizes and approximates each workload into a small set of coarse-grained resource demand (e.g., low, moderate, high). In this way, a *risk (hyper-)cube model* (in a hyper-cube space) can be derived to capture the system load without any tedious parameter inference efforts. In addition, the online checking of the risk cube model's deviation drives the system reoptimization according to online feedbacks. In particular, the compact model simplifies

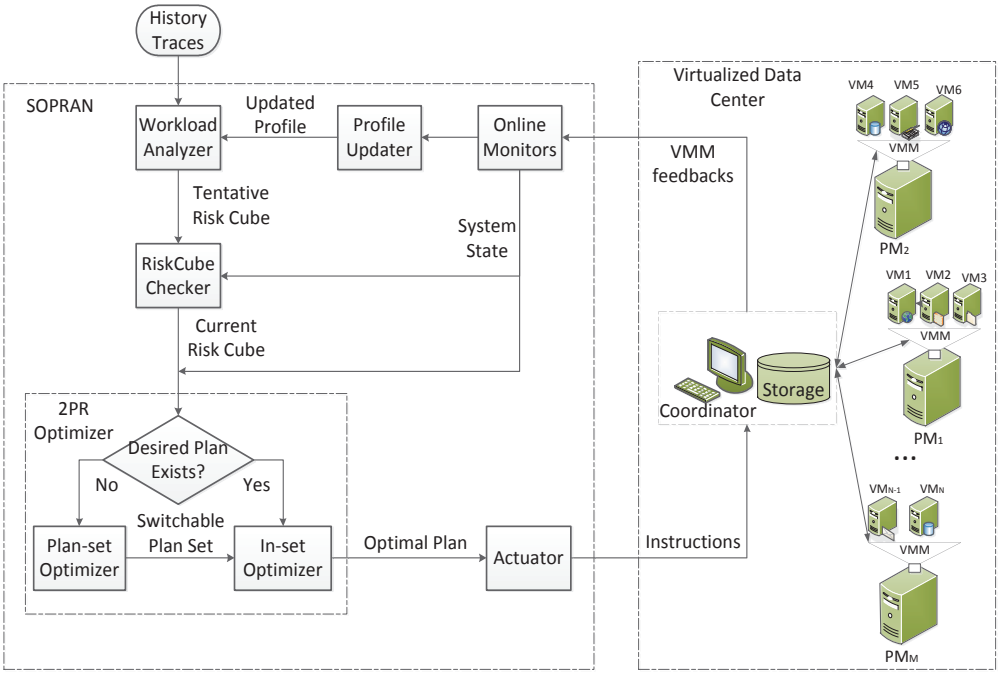


Figure. 2: SOPRAN system architecture

the optimization process, and makes the overall optimization overhead grow at a much slower speed than precise modeling methods as the number of VM increases.

5.1 System Architecture

Figure 2 shows the architecture of SOPRAN. It consists of several key components:

— The **Workload Analyzer** examines the (historical) workload traces to build a model that represents the possible states of the system load. For N distinct workloads, the model is essentially a N-dimensional hyper-cube which we called *risk cube*. At runtime, the **Profile Updater** gathers runtime system demand (from the **Online Monitor**) and updates the workload profiles. These updated profiles are periodically processed by the workload analyzer to generate a *tentative* risk cube that more accurately reflects the recent workload.

— The **Two-Phase Re-optimizer (2PR optimizer)** consists of two sub-components - the **Plan-set Optimizer** and the **In-set Optimizer**. After the risk cube has been updated, a new set of representative states (see the definition in section 5.3) are derived from the new risk cube, and the previously maintained plan set (which we call *switchable plan set*) must be updated. The plan-set optimizer is the one that incrementally generates one optimal plan for each representative state and maintains the complete set of plans for all representative states, forming the *switchable plan set*. At runtime, the in-set optimizer matches the current system load state to the closest state in the representative states set, and selects the corresponding plan in the switchable plan set, which is actually the pre-computed optimal plan for this representative state by the plan-set optimizer. As long as the system load stays within the current risk cube, the best plan can be easily obtained from the switchable plan set without further reoptimization. However, when the current switchable plan set no longer gives acceptable performance, the in-set optimizer will request an update of the risk cube.

— The **Online Monitor** on each VMM samples the actual demands of VMs and reports to the **Profile Updater**. It also detects SLA violation of the running workloads. If the average SLA violation rate exceeds a predtermined threshold (say, 5%), 2PR optimizer will be triggered.

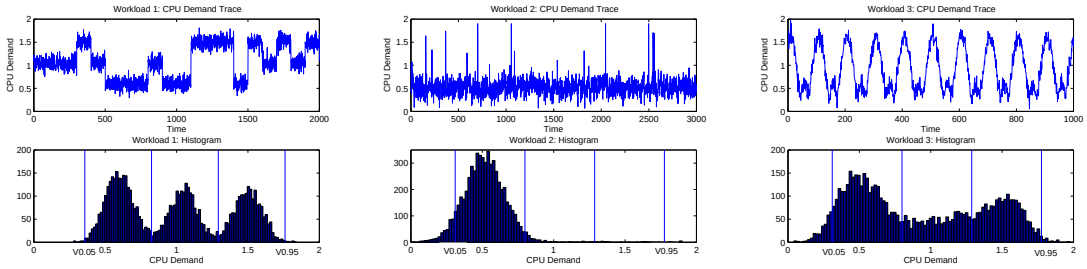


Figure. 3: Three categories of workload patterns with histograms

The SLA violation for each workload is obtained by comparing the assigned amount of resource with the monitored actual demand within the time window. If the ratio between the assigned demand and the actual demand is below the pre-defined threshold, this is counted as a violation. If over the time window 5% of the time the SLA is violated for this workload, then the SLA violation rate is considered to have exceeded the threshold.

— The **RiskCube Checker** checks the accuracy of the current risk cube. Once the discrepancy between the tentative risk cube and the current one exceeds a certain pre-determined threshold, the riskCube checker will replace the current risk cube with the tentative one, and notify the plan-set optimizer to clear the current switchable plan set.

— The **Actuator** is responsible for actuating the plan that is chosen by the optimizer. It analyzes the plan matrix and translates the matrix into the corresponding residence relationship between the VMs and PMs. Then it compares this chosen plan with the current plan in the system, and figures out the discrepancies, e.g. whether the host of each VM in the currently deployed plan has been changed in the newly chosen plan, and if changed, what is the IP address and related information of the new host for this VM. Thus an instruction that calls the virtual machine monitor on the current host to suspend and migrate the particular VM to the destination host is pushed into the instruction list, which is a series of instructions that migrate the affected VMs.

The design of SOPRAN has three novel features:

— **Feature 1: Risk Cube Model** of workloads contributes to SOPRAN's supporting of various workload patterns, simpler model and lower migration rate;

— **Feature 2: Adaptive Two-Phase Re-optimization** contributes to better SLA fulfillment and less online re-optimization overhead; and

— **Feature 3: Incremental Plan Generation** contributes to the further reduction of re-optimization overhead, especially for large scales of VMs, so as to guarantee good system scalability.

The three features are the three main steps in SOPRAN, which we shall discuss in depth in the following three subsections.

5.2 Risk Cube Model

5.2.1 *Workloads Characterization.* The characteristics of the dynamic workload demands can provide useful information to (a) better explore the optimization opportunities, and (b) avoid frequent re-optimization and migration, so as to improve system performance and guarantee SLA fulfillment.

For simplicity, we assume that each workload's resource demand is represented by its CPU demand trace; generalizing to other types of resources will be considered in future work. According to the survey of [Bobroff et al. 2007] on a large number of CPU usage traces from production servers, the workloads can be roughly divided into three categories, as shown in Figure 3. The

first type represents servers with consecutive tasks executed, such as application servers. The second type represents servers with random density of CPU loads, and sometimes have sudden burst, such as web servers and mail servers. And the third type has clear repetitive patterns in CPU demands with peak time and valley time. Application servers and database servers used by banking system often display such workload patterns – during office hours the servers are at peak load, but in the night the systems become idle or lightly loaded.

To clearly illustrate our proposed scheme, in this paper we reasonably assume that all the workloads running in the data center display the three patterns of resource demands without lost of generality.

5.2.2 *Risk Cube and Representative Value Vector.* Because the three types of workloads present different patterns, we assign each pattern a *risk score* based on the demand fluctuation, to measure the degree of uncertainty. The risk score can be labeled either using the statistical parameter of historical demand traces, or values assigned by experts (based on their experience) when the historical traces are not available. Besides, for each workload trace a *risk interval* can be formed, indicating that the future demand values are most likely to be within this value range. The calculation procedure of risk intervals from historical traces will be discussed later.

Risk intervals make the optimizer aware of the dynamic pattern of each workload, so that it can switch plans accordingly or trigger re-optimization if necessary. More importantly, such plans are *robust* enough to be effective for the interval. In this way, the system is more stable system with fewer re-optimizations.

With the historical trace of workload j, we calculate the risk interval as follows: make a histogram of the trace and take it as an approximation of the probability density distribution of the demand values, and identify the 5 and 95 (or any other numbers defined by the system administrator, such as 2.5 and 97.5) percentile values of this histogram, denoted as $v_{0.05}$ and $v_{0.95}$. Thus the value of risk score $risk_j$ for workload j is assigned as:

$$risk_j = scale \times (v_{0.95} - v_{0.05}) \tag{6}$$

The scale factor, *scale*, is an ad-hoc number that identifies the sensitivity of this risk score. It is also defined by the system administrator and should be between 0 and 1, such as 0.5, 0.6. And therefore the risk interval is $[v_{0.05}, v_{0.95}]$, meaning that the probability of the next value falling into this range is approximately 90%.

With the risk interval computed, a d-dimensional *representative value vector* for this histogram can be drawn. By evenly dividing the risk interval into d subintervals, each subinterval's upper value is picked as a representative value for the subinterval. The dimension of the representative value vector is set to be 3 by default, as our experiments show that 3 representative values are adequate to approximate the demand curve with satisfying performance and low overhead. Three-dimensional representative value set is illustrated by the three vertical lines (exclude the left-most one) in each of the histogram in Figure 3.

Since there are three types of workloads in the system, if we put each workload's demand value along one dimension in a coordinate system, the three representative value vectors will form a *risk cube* as shown in Figure 4. If at time t we observe all three workloads' demand values as a vector, the point in the coordinate space corresponding to this vector at time t will most likely arrive inside the risk cube.

In the procedure of resource allocation, for each workload we get the actual amount of CPU resource demand, and pick the value which is immediately greater than it from the three representative values of this workload type, thus forming the staircase approximation of this actual trace along the time axis, as shown in Figure 6. Note that the three representative values will be periodically refined according to online statistics.

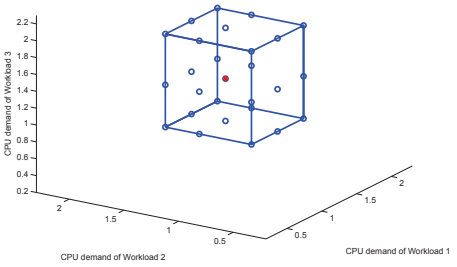


Figure. 4: Three-dimensional risk cube

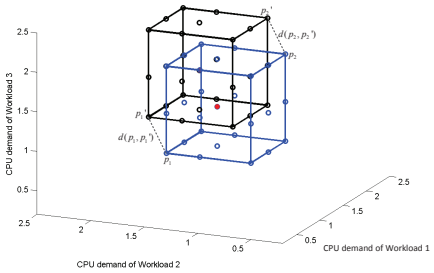


Figure. 5: Risk cube discrepancy

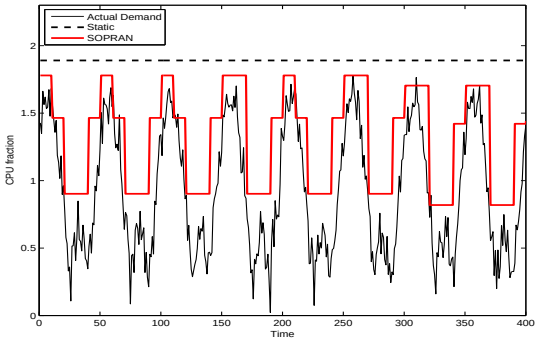


Figure. 6: Staircase approximation of workload

5.3 Two-Phase Re-optimization

Before introducing SOPRAN's 2PR optimizer, let us define some terms used in the following description:

DEFINITION 1. *At any time point t_0, the CPU demands observed from all N VMs form a N-dimensional vector, $D(t_0) =< D_1(t_0), D_2(t_0), ..., D_N(t_0) >$, called the* ***System State*** *at time t_0.*

DEFINITION 2. *Each sample value of workload demand has its representative value which has three possible values (details in Section 5.2), and there are three types of workloads in this system. So there are a set of possible combinations, called the* ***Representative States Set*** *(RSS), which will be effective until the risk cube has been refined. Note that RSS can be directly formed from risk cube.*

DEFINITION 3. *For a given system state, the plan that has the minimum value for the objective function (5) is called the* ***Optimal Plan*** *at this system state.*

DEFINITION 4. *For each representative state set, if we find every optimal plan at all representative states and form a plan set, it is called the* ***Switchable Plans Set*** *corresponding to this representative states set.*

DEFINITION 5. *The risk cube that is active in the system, and to which the current switchable plan set corresponds, is called the* ***Current Risk Cube****.*

DEFINITION 6. *Each time the profile has been updated, a new risk cube, called the* ***Tentative Risk Cube****, will be computed by the workload analyzer.*

The tentative risk cube will be compared against the current risk cube, and the decision on whether to replace the current risk cube with the tentative one is made during runtime.

In SOPRAN, we adopt an enhanced version of the solution proposed in Section 4 to achieve a more robust performance as well as a lower optimization overhead.

Initially, the workload analyzer studies the historical resource demand traces of workloads and characterizes the workloads using the *representative states*. From the resultant combined *risk cube* model for all workloads, the *representative state set* can then be derived from the risk cube. The starting state of the system is measured and an initial plan is deployed. During runtime, the online monitor performs two tasks. First it measures the actual system state and uses it as an estimate for the workload demands for the next time window τ. Second, it estimates the SLA violation rate under the current resource allocation. If the detected or anticipated SLA violation rate is below the pre-determined threshold, the system will carry on with the current plan. Otherwise, it will trigger 2PR optimizer.

The 2PR optimizer works in two phases: in the first phase, it compares the system state with the current representative state set. If the system state still stays within the current risk cube, the 2PR optimizer can proceed to the second phase. In the second phase the in-set optimizer adjusts the plan among the switchable plans. This ensures that as few changes as possible would be made to the current placement. For example, if the current system state changes from closest to representative state 1 to closest to representative state 2, then the system can just jump from plan 1 to plan 2 in the switchable plans set, without exhausting the huge plan space.

However, if the system state falls outside the risk cube so that none of the representative state in RSS can be used, then risk cube has to be refined and the switchable plan set has to be reset, as discussed in the next paragraph.

Besides the reactive refinement of risk cube, SOPRAN will also proactively refines it if necessary. The profile updater will gather the actual demands from the online monitor and add into the historical traces, forming the updated profile. The workload analyzer will periodically compute the tentative risk cube from the updated profile. The RiskCube Checker continuously compares the tentative risk cube with the current one. Once the discrepancy between the two risk cubes is big enough, or in the two-phase re-optimization loop the system state lies far outside the risk cube, the tentative risk cube will become the current risk cube, and the switchable plan set will be regenerated.

The discrepancy of the two risk cubes is measured by the average distance between the corresponding matching points from the two cubes, as shown in Figure 5. Each representative state in the RSS is a point in the risk cube, denoted as $p_1, p_2, ..., p_l$, where l is the dimension of RSS and SPS. Let $d(p_i, p_i')$ be the Euclidean distance of p and p' in the d-dimensional coordinate system. The discrepancy of two risk cubes is therefore $dif(riskCube, terisk Cube) = \frac{1}{l}\sum_{i=1}^{l}(d(p_i, p_i'))$. To decide whether the discrepancy is big enough or not, a threshold is set as the guideline. Intuitively, a stricter threshold means more aggressive re-optimization and higher migration rate, which may incur more service interruption and migration failure. On the other hand, a less strict threshold may cause higher SLA violation rate. In our experiment we simply choose a value that can make the SLA violation rate just below 5%, in order to trade-off between performance and optimization overhead, and to guarantee that the deployment we make can be effective enough in the foreseeable future.

The 2PR optimizer can effectively reduce VM migration rate and service interruption by reusing switchable plans as much as possible. Algorithm 1 gives an algorithmic description of the main idea of the two-phase re-optimization.

5.4 Incremental Plan Generation

The third feature can further reduce or spread the online re-optimization overhead: once the representative states are changed, instead of generating the l switchable plans all at a time, SOPRAN incrementally optimizes them when the corresponding representative state is encountered, and keeps the plan rather than drop it once the state has expired. In such case, as long as the current representative states set is still effective, for any representative state that has been encountered

before and is picked again, SOPRAN can simply reuse the pre-computed plan in the SPS instead of optimizing again. Not until the risk cube has been refined will the switchable plans set be cleared.

This feature is extremely helpful when the risk-cube checking interval is long, and the system has a large scale. It not only avoids repeated computations, but also reduces SOPRAN optimizer's online latency.

Algorithm 1: Two-Phase Re-optimization

input : history traces $D(t)_{hist}, t \in [T_0 - S, T_0]$ and realtime traces $D(t), t \in [T_0, T_{end}]$
output: Optimal Plan $OptPlan$

compute $RiskCube$ and RSS from $D(t)_{hist}$;
$OptPlan = initialPlan$;
Tentative risk cube $TeRiskCube = RiskCube$;
/*realtime loop*/
while $T_0 \leq t_0 \leq T_{end}$ **do**
  update $RiskCube$' threshold $thres$;
  window number $w = 0$;
  closet representative state id $sid = 0$;
  foreach $\tau \in S$ **do**
    w ++;
    update S_w;
    $OptPlan = Plan_{sid}$ in SPS;
    $PMused_w = rank(OptPlan)$;
    get monitored D_w with length τ;
    /*record SLA violation in this time window*/
    $SLAvio_w = compare(S_w, D_w, SLA)$;
    $S_w^{current} = S_{window}$;
    $sid^{current} = sid$;
    $< sid, S_w >= InsetOptimizer(D_w, RSS)$;
    /*First phase: reactively refine $RiskCube$*/
    if *system state not close enough to any Representative state* **then**
      $RiskCube = TeRiskCube$;
      derive RSS from $RiskCube$;
      clear SPS;
      $< sid, S_w >= InsetOptimizer(D_w, RSS)$;
    /*Second phase: state matching and plan selection*/
    if $S_w^{current} \neq S_w$ **then**
      if $Plan_{sid}$ *not exists* **then**
        $OptPlan = PlansetOptimizer(S_w, R)$;
        accumulate $OptimizCost$;
        add $OptPlan$ to SPS as $Plan_{sid}$;
        else
          $OptPlan = Plan_{sid}$ in SPS;
      deploy $Plan_{sid}$ in SPS;
    compute $TeRiskCube$ from updated profile;
    $shift = compare(RiskCube, TeRiskCube)$;
    /*proactively refine $RiskCube$ and RSS*/
    if $shift \geq thres$ **then**
      $RiskCube = TeRiskCube$;
      derive RSS from $RiskCube$;
      clear SPS;
    $t_0 = t_0 + S$;

6. EXPERIMENTAL EVALUATION

To fully understand the system performance of SOPRAN, we have conducted two groups of experiments. The first evaluates the benefits that can be derived from the three features in SOPRAN. The second compares the overall performance of SOPRAN to existing methods, namely, the IBM MFR system [Bobroff et al. 2007] and a static deployment method.

6.1 Experiment Setups

In our simulation experiments, we adopt the data center topology as described in Section 5.1.

The number of VMs is varied from 25 to 1000 to see the scalability performance. We assume that there are always sufficient PMs in the data center for up to 1000 VMs with the given SLA. The PMs have identical configurations with 4 CPU cores and 4G memory. The data are stored in shared disk pool, and VM migration has no effect on data transfer.

To focus on the performance evaluation of SOPRAN on system modeling and reoptmization, we only implemented the Workload Analyzer, Profile Updater, RiskCube Checker and the 2PR Optimizer. The workloads running on the VMs are emulated to generate resource demands as the input to our experiment testbed.

In each round of experiments, the number of VMs is fixed at a certain number N, and the workload traces are generated randomly at the beginning of each round. The three algorithms are applied separately for a simulated time duration of 2500 hours with the same set of workloads. After each round, the values of the performance indicators (described in section 6.3) are calculated, and the results presented in sections 6.4 and 6.5 are the average level of 5 rounds for each case.

6.2 Workloads Emulation

To mimic the data center workloads, we generate three types of CPU usage traces as the workload demand traces during runtime, with the patterns described in Section 5.2.

The first pattern is approximated using step functions, together with white noise being added, which is quite common in real world machine's CPU usage conditions. In our experiments, this type of workload trace is realized with random parameters drawn from a pre-defined range.

The second pattern is approximated using noised AR(2) time series functions, which is also the model adopted in [Bobroff et al. 2007]. Each realization of this type's trace is generated with random AR coefficients and randomized interpolations of sudden bursts.

The third type is approximated using noised sine function, and each realization will have randomized amplitude and phase. Figure 3 illustrates these workload types.

The generated traces are used as input to each algorithm to evaluate the online performance. To ensure fairness, each round of comparison experiments will use the same group of traces.

6.3 Performance Metrics

The following performance metrics are used to evaluate the methods in our experiments, in order of decreasing importance from the perspective of data centers.

(1) Average SLA violation rate during runtime. This reflects the performance of a scheme in terms of meeting customers' SLA requirement. It is very critical to data centers to ensure customers are satisfied;

(2) Average migration rate during runtime. This indicates a scheme's ability to provide system stability. Frequent migration increases the failure rate and overhead;

(3) Total re-optimization overhead during runtime. This captures the overhead of a scheme. It is important for the online performance and system reliability; and

(4) Average number of physical machines used during runtime. This gives an indication of the operational cost of the data center. In particular, machines that are not used can be powered down or set to stand-by mode to make the system more energy efficient.

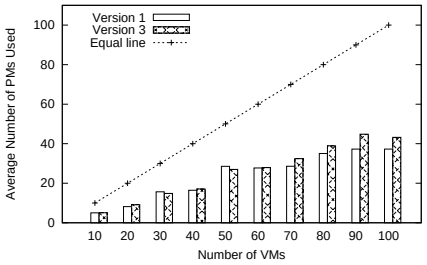


Figure. 7: Version 1 vs Version 3: average number of PMs used during runtime

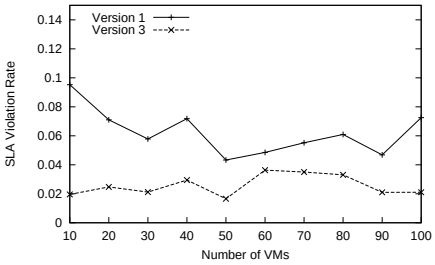


Figure. 8: Version 1 vs Version 3: average SLA violation rate during runtime

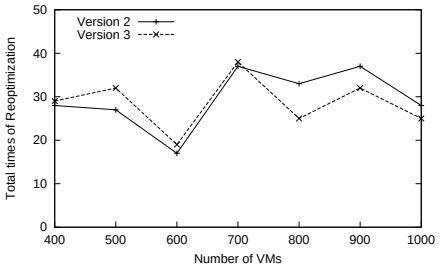


Figure. 9: Version 2 vs Version 3: total number of re-optimizations during runtime

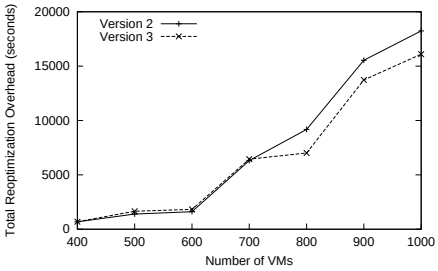


Figure. 10: Version 2 vs Version 3: total re-optimization overhead during runtime

6.4 SOPRAN Feature Evaluation

To evaluate the contributions derived from each of the three features in SOPRAN, we compare the performance of three versions of this algorithm constructed by deactivating different features in SOPRAN.

— Version 1: Only Feature 1 is activated and the other two are deactivated, combined with a regular optimization algorithm used in [Bobroff et al. 2007]. Therefore, it uses the representative value set to form a staircase approximation of the workload demands, and re-optimize the plan once SLA violation is anticipated to happen.

— Version 2: Feature 1 and Feature 2 are activated, and Feature 3 is deactivated, therefore it uses a staircase approximation of the workload demands, and has the risk cube (or equally representative states) which will also be refined online. Each time when the current representative state is expired it will drop the plan instead of keeping it for possible further usage.

— Version 3: All three features are activated, forming the full-functional SOPRAN system.

First, we compare Version 1 and Version 3 under the same conditions, and the results are shown in Figure 7 and Figure 8. From the figures we can see that even though Version 3 has slightly higher (-1% to 8% more) resource consumption than version 1, it can achieve much lower (2-4 times less) and more stable SLA violation rate. Since SLA fulfillment is the most important objective for data center, we prefer Version 3 over Version 1.

Next we compare Version 2 and Version 3 under the same conditions. The results are presented in Figure 9 and Figure 10. As shown in the figures, as the system scales up to 1000 VMs, Version 3 can save up to 12% of the online re-optimization overhead, which means that the **Incremental Plan Generation** feature can actually help to reduce substantial overhead and improve the system scalability.

Based on the above two comparison experiments, it is clear that all the three features in SOPRAN are indispensable for better performance, in terms of fulfilling SLA, reducing overhead and improving system scalability.

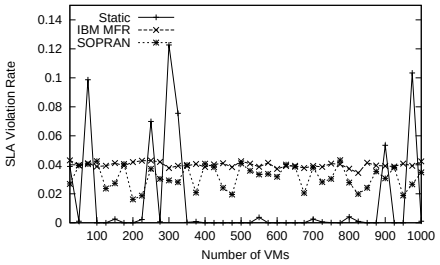


Figure. 11: Average SLA violation rate during runtime

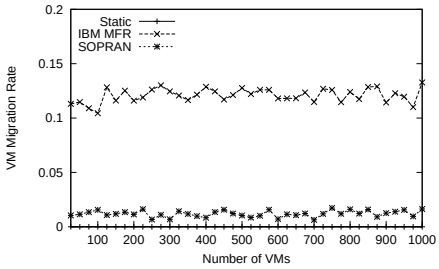


Figure. 12: Average VM migration rate during runtime

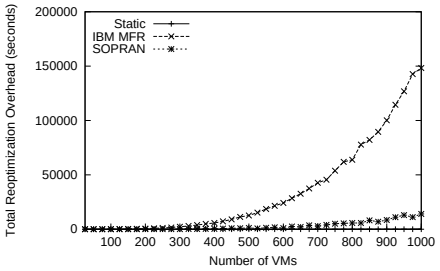


Figure. 13: Total re-optimization overhead during runtime

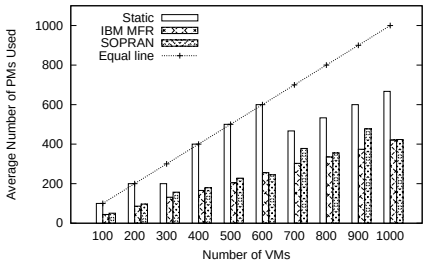


Figure. 14: Average number of PMs used during runtime

In the next group of experiments that compare SOPRAN with other methods, we activate all three features in SOPRAN.

6.5 Comparison Experiments

The IBM MFR is a state-of-the-art workload modeling based method that reconfigures VM online with SLA constraints (see Section 2). However, it only applies to the second workload pattern in Section 5.2 (see Figure 3). For other two patterns, MFR can neither formulate their model nor infer the parameters. Therefore, we set MFR algorithm to have full knowledge of the future demand for the next time window τ, while SOPRAN can only estimate from past traces. In other words, the experiments are biased towards MFR scheme. For static deployment, our static scheme gets the profile at the beginning of deployment based on the history traces, allocates the maximum amount of past demand to the VMs, and maintains this configuration for a long time. By letting N varies from 25 to 1000 with the step of 25, we run 40 experiments in total.

6.5.1 *Average SLA Violation Rate.* Under the same experiment settings, the average SLA violation rates of the three algorithms are illustrated in Figure 11. First of all, we observe that the SLA fulfillment under static deployment is very uncertain. When the history traces happened to be perfectly representative of the online load, the static deployment is very effective - it can meet most of the SLA requirements (i.e., violation rate below 1%). However, on occasions where there are sudden bursts of workload or when unexpected patterns happen, the static deployment may fail to handle them. Hence, the violation rate can even be as high as 13%, which would be totally unacceptable to customers.

On the contrary, both MFR and SOPRAN can adapt to the dynamic changes in the workload demands. Even if the changes were not reflected in the history traces, both the two algorithms can make adjustments based on model prediction or online monitoring. Since we assume MFR can perfectly predict the future demands, in the experiments MFR can keep the SLA violation rate below 5% when the number of VMs scales from 25 to 1000. SOPRAN can also response effectively to the sudden burst or unknown patterns by monitoring the actual demands and

refining the risk cube. Therefore SOPRAN can successfully control the SLA violation rate to be around 2% to 4%.

6.5.2 *Average Migration Rate.* Next we compare the three algorithms in terms of the VM migration rates. The results are shown in Figure 12. Since the static scheme does not perform any re-optimization, it will not trigger any migrations. While both MFR and SOPRAN allow live migrations, they react differently: MFR is more aggressive than SOPRAN in performing re-optimization, and exploits more opportunities of migration and re-optimization to achieve better SLA fulfillment and lower resource consumption (see Figure 14). Therefore the average migration rate under MFR is quite large - above 12%. SOPRAN, on the other hand, can save up to 6 times the migration rate of MFR - it has a migration rate of lower than 2%. However, SOPRAN incurs slightly higher resource consumption. We argue that this cost is worth paying since migration itself also consumes resources, such as CPU time, memory, network and I/O bandwidth. More importantly, the gain in terms of lower VM migration rate will give rise to shorter service interruption and lower failure rate.

6.5.3 *Total Re-optimization Overhead.* Static deployment also does not incur any re-optimization overhead even though its performance may be poor at times.

As illustrated in Figure 13, SOPRAN saves as much as 90% of the re-optimization overhead compared to MFR, especially when there is a large number of VMs. We can see that the overhead of MFR grows dramatically, while SOPRAN's grows at a much slower speed.

For the MFR algorithm, a VM should be allocated exactly or slightly higher amount of resources than its demand. This is to ensure that it can fulfill the SLA while keeping the cost low. As such, it has to re-optimize nearly every time the workload demands change. Therefore the re-optimization overhead grows exponentially as the number of VMs scales up, given that the search space grows dramatically.

While SOPRAN has a coarser re-optimization scheme, and it only searches among limited combinations of the state space when the system scales up, therefore the re-optimization cost each time is much lower. Moreover, each time when SOPRAN is making a decision for the next τ, it will anticipate the future demands based on the history. Thus, the next plan to be actuated is likely to be effective in the foreseeable future. This characteristic reduces the re-optimization times, increases the system reliability, and earns the much lower migration rate as explained in Section 6.5.2. Furthermore, less optimization overhead also avoids slowing down the whole system.

From the above perspectives, we can see that SOPRAN is especially efficient for large scale systems.

6.5.4 *Average Number of Physical Machines Used.* Although SLA is the most important concern of both customers and data centers, the resource utilization is also important for the data centers. This is especially so in a cloud environment where the data centers want to make higher profit and at the same time compete with opponents with lower price.

From Figure 14 we can see that both SOPAN and MFR can save substantial amount of average number of PMs used: MFR uses 51.9% of the number of PMs used by static deployment, while SOPRAN uses 58.6%, which is 6.6% higher than MFR.

To summarize, we can see that with the same settings and workloads, both SOPRAN and MFR are much more efficient and flexible than static deployment. While SOPRAN consumes slightly more resources, it compromises the cost for better SLA fulfillment, much lower migration rate and re-optimization overhead and better scalability.

7. CONCLUSIONS

In this paper, we have addressed the problem of online virtual machine management with dynamic workloads. Our proposed SOPRAN system represents the possible states of the system load in the form of a risk (hyper)cube. By maintaining VM-PM allocation for each of these states,

the best allocation can be quickly determined as long as the actual workload stays within the risk cube space. By monitoring the system workload, the risk cube is continuously refined and kept up-to-date to reflect the changing workload demands. Our experimental study shows that SOPRAN, in comparison with a state-of-the-art MFR scheme, is applicable to a broader range of workload patterns, can fulfill SLA requirements, achieves much less overhead and migration rate using slightly higher resource consumption. These advantages make SOPRAN more suitable and reliable for online virtual machines management in large scale data centers.

REFERENCES

Abdelzaher, T. F., Shin, K. G., and Bhatti, N. 2002. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems 13*, 80–96.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. 2003. Xen and the art of virtualization. In *Proceedings of the 19th ACM symposium on Operating systems principles(SOSP)*.

Bobroff, N., Kochut, A., and Beaty, K. 2007. Dynamic placement of virtual machines for managing sla violations. In *10th IFIP/IEEE International Symposium on Integrated Network Management*. IEEE.

Brantner, M., Florescu, D., Graf, D., Kossmann, D., and Kraska, T. 2008. Building a database on s3. In *Proceedings of ACM SIGMOD international conference on Management of data(SIGMOD)*.

Bu, X., Rao, J., and Xu, C.-Z. 2009. A reinforcement learning approach to online web systems auto-configuration. In *Proceedings of 29th IEEE International Conference on Distributed Computing Systems(ICDCS)*.

Cherkasova, L. and Gardner, R. 2005. Measuring cpu overhead for i/o processing in the xen virtual machine monitor. In *Proceedings of the annual conference on USENIX Annual Technical Conference(ATEC)*.

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. 2005. Live migration of virtual machines. In *Proceedings of conference on Symposium on Networked Systems Design & Implementation(NSDI)*.

Fito, J., Goiri, I., and Guitart, J. 2010. Sla-driven elastic cloud hosting provider. In *Proceedings of 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*.

Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., and Boneh, D. 2003. Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM symposium on Operating systems principles(SOSP)*.

Gmach, D., Rolia, J., Cherkasova, L., Belrose, G., Turicchi, T., and Kemper, A. 2008. An integrated approach to resource pool management: Policies, efficiency and quality metrics. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*. 326 –335.

Gulati, A., Merchant, A., Uysal, M., Padala, P., and Varman, P. 2009. Efficient and adaptive proportional share i/o scheduling. *ACM SIGMETRICS Performance Evaluation Review 37,* 2.

Haletky, E. L. 2007. *VMware ESX Server in the Enterprise: Planning and Securing Virtualization Servers*. Prentice Hall.

Jang, J.-W., Jeon, M., Kim, H.-S., Jo, H., Kim, J.-S., and Maeng, S. 2011. Energy reduction in consolidated servers through memory-aware virtual machine scheduling. *IEEE Transactions on Computers 60*, 552–564.

Jung, G., Joshi, K. R., Hiltunen, M. A., Schlichting, R. D., and Pu, C. 2009. A cost-sensitive adaptation engine for server consolidation of multitier applications. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*. Middleware '09. Springer-Verlag New York, Inc., New York, NY, USA, 9:1–9:20.

Kansal, A., Zhao, F., Liu, J., Kothari, N., and Bhattacharya, A. A. 2010. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM symposium on Cloud computing*. SoCC '10. ACM, New York, NY, USA, 39–50.

Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., and Tantawi, A. 2006. Dynamic placement for clustered web applications. In *Proceedings of the international conference on World Wide Web(WWW)*.

Kim, H., Lim, H., Jeong, J., Jo, H., and Lee, J. 2009. Task-aware virtual machine scheduling for i/o performance. In *Proceedings of ACM SIGPLAN/SIGOPS international conference on Virtual execution environments(VEE)*. ACM, New York, NY, USA.

Kimbrel, T., Schieber, B., and Sviridenko, M. 2004. Minimizing migrations in fair multiprocessor scheduling of persistent tasks. In *Proceedings of the 15th annual ACM-SIAM symposium on Discrete algorithms(SODA)*.

McNett, M., Gupta, D., Vahdat, A., and Voelker, G. M. 2007. Usher: an extensible framework for managing custers of virtual machines. In *Proceedings of the 21st conference on Large Installation System Administration Conference(LISA)*.

Ng, C., Parkes, D. C., and Seltzer, M. 2003. Virtual worlds: fast and strategyproof auctions for dynamic resource allocation. In *Proceedings of the 4th ACM conference on Electronic commerce(EC)*.

Ongaro, D., Cox, A. L., and Rixner, S. 2008. Scheduling i/o in virtual machine monitors. In *Proceedings of ACM SIGPLAN/SIGOPS international conference on Virtual execution environments(VEE)*.

Padala, P., Hou, K.-Y., Shin, K. G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., and Merchant, A. 2009. Automated control of multiple virtualized resources. In *Proceedings of ACM European conference on Computer systems(EuroSys)*.

Rolia, J., Cherkasova, L., Arlitt, M., and Andrzejak, A. 2005. A capacity management service for resource pools. In *Proceedings of the 5th international workshop on Software and performance*. WOSP '05. ACM, New York, NY, USA, 229–237.

Rolia, J., Zhu, X., Arlitt, M., and Andrzejak, A. 2004. Statistical service assurances for applications in utility grid environments. *Perform. Eval. 58*, 319–339.

Sapuntzakis, C. P., Chandra, R., Pfaff, B., Chow, J., Lam, M. S., and Rosenblum, M. 2002. Optimizing the migration of virtual computers. In *Proceedings of the Symposium on Operating Systems Design and Implementation(OSDI)*. 377–390.

Seltzsam, S., Gmach, D., Krompass, S., Kemper, A., and Mnchen, T. U. 2006. Autoglobe: An automatic administration concept for service-oriented database applications. In *Proc. of the 22nd Intl. Conference on Data Engineering (ICDE2006), Industrial Track*.

Soror, A. A., Minhas, U. F., Aboulnaga, A., Salem, K., Kokosielis, P., and Kamath, S. 2008. Automatic virtual machine configuration for database workloads. In *SIGMOD*.

Urgaonkar, B., Shenoy, P., and Roscoe, T. 2002. Resource overbooking and application profiling in shared hosting platforms. In *OSDI*.

Verma, A., Ahuja, P., and Neogi, A. 2008. pmapper: Power and migration cost aware application placement in virtualized systems. In *Proceedings of the ACM/IFIP/USENIX 9th International Middleware Conference*. Springer-Verlag, Berlin, Heidelberg, 243–264.

Verma, A., Kumar, G., and Koller, R. 2010. The cost of reconfiguration in a cloud. In *Proceedings of the 11th International Middleware Conference Industrial track*. Middleware Industrial Track '10. ACM, New York, NY, USA, 11–16.

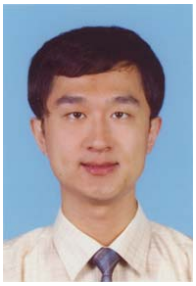
Wang, X., Lan, D., Wang, G., Fang, X., Ye, M., Chen, Y., and Wang, Q. 2007. Appliance-based autonomic provisioning framework for virtualized outsourcing data center. In *Proceedings of the 4th International Conference on Autonomic Computing(ICAC)*.

Wood, T., Shenoy, P., Venkataramani, A., and Yousif, M. 2007. Abstract black-box and gray-box strategies for virtual machine migration. In *Proceedings of USENIX Symposium on Networked Systems Design & Implementation(NSDI)*.

Xu, J., Zhao, M., Fortes, J., Carpenter, R., and Yousif, M. 2007. On the use of fuzzy modeling in virtualized data center management. In *Proceedings of the Fourth International Conference on Autonomic Computing*. IEEE Computer Society, Washington, DC, USA.

Jian Zhou is a Ph.D. student at the National University of Singapore in the Graduate School for Integrative Sciences and Engineering. Her research interests include virtual machine management and database system in cloud computing. She received her B.S. degree in Control Science and Engineering at the Huazhong University of Science and Technology in 2009, and received the NGS Scholarship from NUS to pursue a Ph.D degree under the supervision of Prof Kian-Lee Tan from School of Computing, National University of Singapore.

Lei Shi received his bachelor degree in College of Information Science and Engineering, Northeastern University (NEU), China in 2008. Currently he is a Ph.D candidate in School of Computing, National University of Singapore. His research interests include cloud computing infrastructure, parallel and distributed database theories and applications.



Dr. Kian-Lee Tan received his B.Sc (Hons) and Ph.D. in computer science from the National University of Singapore, in 1989 and 1994 respectively. He is currently a Professor in the Department of Computer Science, National University of Singapore. His major research interests include query processing and optimization, database security and database performance. He has published over 250 conference/journal papers in international conferences and journals. He has also co-authored three books. Kian-Lee is a member of the Association of Computing Machinery (ACM).