Stay-Away, protecting sensitive applications from performance interference

Conference Paper · December 2014

DOI: 10.1145/2663165.2663327

Navaneeth Rameshan
Universitat Politècnica de Catalunya
16 PUBLICATIONS **86** CITATIONS
SEE PROFILE

Leandro Navarro
Universitat Politècnica de Catalunya
230 PUBLICATIONS **1,138** CITATIONS
SEE PROFILE

Vladimir Vlassov
KTH Royal Institute of Technology
122 PUBLICATIONS **772** CITATIONS
SEE PROFILE

Stay-Away, protecting sensitive applications from performance interference

Navaneeth Rameshan
Universitat Politècnica de Catalunya
rameshan@ac.upc.edu

Leandro Navarro
Universitat Politècnica de Catalunya
leandro@ac.upc.edu

Enric Monte
Universitat Politècnica de Catalunya
enric.monte@upc.edu

Vladimir Vlassov
KTH Royal Institute of Technology
vladv@kth.se

ABSTRACT

While co-locating virtual machines improves utilization in resource shared environments, the resulting performance interference between VMs is difficult to model or predict. QoS sensitive applications can suffer from resource co-location with other less short-term resource sensitive or batch applications. The common practice of overprovisioning resources helps to avoid performance interference and guarantee QoS but leads to low machine utilization. Recent work that relies on static approaches suffer from practical limitations due to assumptions such as a priori knowledge of application behaviour and workload.

To address these limitations, we present Stay-Away, a generic and adaptive mechanism to mitigate the detrimental effects of performance interference on sensitive applications when co-located with batch applications. Our mechanism complements the allocation decisions of resource schedulers by continuously learning the favourable and unfavourable states of co-execution and mapping them to a state-space representation. Trajectories in this representation are used to predict and prevent any transition towards interference of sensitive applications by proactively throttling the execution of batch applications. The representation also doubles as a template to prevent violations in the future execution of the repeatable sensitive application when co-located with other batch applications. Experimental results with realistic applications show that it is possible to guarantee a high level of QoS for latency sensitive applications while also improving machine utilization.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Management, Measurement, Performance, Experimentation

Keywords

Performance interference, Interference mitigation, Performance sensitivity, Quality of Service, Virtualization

1. INTRODUCTION

In recent years, there has been an increasing amount of large scale distributed computing infrastructures such as the cloud and large scale experimental testbeds like Planetlab[8] to run large-scale network services. Any of these infrastructures must satisfy a common goal of application isolation and resource efficiency. This requires servers to be shared among multiple users and at the same time guarantee operational isolation of applications. Virtualization has become the defacto standard to achieve these goals. While virtualization guarantees application isolation, better resource utilization and lower operational costs, it comes at the price of application slow down and inter VM performance interference. Inter-VM performance interference happens when behaviour of one VM adversely affect the performance of another due to contention in the shared use of resources. This can happen at any level: memory, cache, I/O buffer, CPU, etc.

Difficulties to model and predict performance interference has resulted in the heavy handed approach of disallowing any co-locations with performance sensitive applications, a major contributor to low machine utilization in data centers [6]. Recent work has seen proposals to predict interference and minimise QoS degradations by relying on static approaches based on prior profiling of applications [36, 19]. Even with an ideal profiling technique, it is impossible to fully characterize an application before runtime inorder to prevent interference and improve utilisation. This is because applications can have varying and sometimes unpredictable inputs/workloads during runtime and services may run for extended periods making it infeasible to profile. Addressing these challenges to mitigate performance interference requires analysis during the runtime to deal with long running jobs and varying workloads.

In this paper, we present Stay-Away, a generic and adaptive mechanism capable of performing a runtime analysis

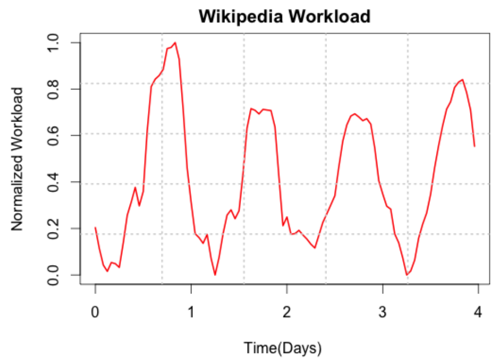


Figure 1: Total Workload variation of Wikipedia during the period 1/1/2011 to 5/1/2011

to execute best-effort batch applications co-located with latency sensitive applications without sacrificing the QoS of latency sensitive applications. Applications typically don't use all the requested resources during the life cycle of their execution because of changes in workload intensity and inherent phase changes. A phase change is defined as a change in the major share of resource consumed by an application. For example, an application can be mostly CPU intensive for a certain period and be I/O intensive at other times. As a result, not all the resources are used at all times. Apart from these phase changes, varying workloads often result in periods of low utilisation. Figure 1 shows the total read workload for Wikipedia obtained from trace [5]. The workload follows a diurnal pattern with clear periods of low workload intensity. Stay-Away identifies and leverages on these periods on low workload intensity and phase changes to improve utilisation by executing best-effort batch applications without sacrificing the QoS of latency sensitive applications.

Stay-Away periodically monitors the resource usage metrics of every Virtual Machine in the host, yielding a time series of measurement vectors. These vectors are then mapped onto a two dimensional space such that similar measurement vectors group together. QoS violations manifest during executions when VMs contend for a resource. This results in measurement vectors that deviate significantly from the measurement vectors of normal executions. As a result measurement vectors of QoS violations are mapped farther away from the group of normal executions. Once this mapping is done, Stay-Away then predicts any progression towards a QoS violation by performing a continuous spatial and temporal analysis of the two dimensional space to identify transitions, their rate and direction. Upon detection of any transition towards a QoS violation, Stay-Away throttles the batch application to avoid a contention before it occurs.

To summarize, the specific contributions in this work are:

- We present the design of Stay-Away, generic and adaptive mechanism to mitigate the detrimental effects of performance interference on sensitive applications when co-located with batch applications.

- We design a methodology to model real-time transitions of the VM states to be aware of dynamic changes across the environment in order to prevent known QoS violations before they occur.

- Additionally, we discuss how this methodology helps visualise co-located execution and serve as a template for repeatable experiments.

We experiment with VLC streaming server and a Web-service, co-located with different set of batch applications. Our results indicate that using Stay-Away, we are able to guarantee a high level of QoS, and are able to increase the machine untilization by 10%-70%, depending on the type of co-located batch application.

2. BACKGROUND

In this section, we highlight the differences between scheduling and dynamic reconfiguration, and provide the necessary background for understanding Multi Dimensional Scaling (MDS), a key component of our approach.

2.1 VM Placement

The problem of mitigating inter-VM interference can be seen from two different perspectives: VM placement and dynamic reconfiguration. The VMs can be scheduled to co-locate in a manner such that they minimize performance interference between each other. For instance, co-locating a memory intensive application with a CPU intensive application is much better than co-locating 2 memory intensive applications together. Papers like Bubble-up [19] and Paragon [11] solve the problem of interference by deciding which VMs to co-locate. In other words, these systems leverage the freedom to co-locate VMs such that there is minimal interference. Their techniques rely on application characterization and fails to address the challenges of dynamic workload and long running applications.

Alternatively, interference can be alleviated even after VMs are co-located. This can be achieved through dynamic reconfiguration. Dynamic reconfiguration may include resource scaling (dynamically increasing the amount of resources allocated to a VM), VM migration or relaxing the guarantees on some VMs. For example, a VM might be sharing a socket with other VMs leading to an interference at cache level, but by dynamically reconfiguring the VM to use the entire socket in an isolated way, the processing power increases and also VMs do not interfere at cache level anymore. This is possible only if additional resources are available for scaling. Cost of dynamic reconfiguration is an important concern. An adaptation is feasible only if the benefits accrued by reconfiguration outweighs the cost involved. VM migration is slow and involves a high cost.Yet another dynamic reconfiguration technique is to throttle the VMs that cause interference and don't need strict guarantees with performance. This does not incur a high cost and is instantaneous.

For these reasons, Stay-Away relies on throttling VMs when resource contentions are about to happen. This affects the performance of the throttled VMs and require them to be best-effort. We introduce the constraint that either best-effort batch applications are scheduled with latency sensitive applications or multiple sensitive applications are scheduled with the notion of priorities. With this constraint in place, we are able to achieve a high level of QoS and improved utilisation. Stay-Away is not a scheduler. It relies on dynamic reconfiguration and can complement from schedulers like Choosy[13] that allows scheduling with constraints. While we throttle batch-applications in our prototype implementation, Stay-Away imposes no limitation on the action that can be taken and if multiple sensitive applications are co-scheduled Stay-Away can choose to migrate or scale resources of the lower priority sensitive application.

2.2 Multi-Dimensional Scaling

Multi-Dimensional Scaling (MDS)[10] has the property of being able to represent a high dimensional space $\mathbf{R^m}$ in a lower dimensional one, for instance $\mathbf{R^2}$, preserving the relative distances, i.e. the absolute value of the distances is lost, but the points are rearranged in a 2D space so that the relative distances between points in the plane correspond to the relative distances in the high dimensional space. Each object or event is represented by a point in a 2D space. The points are arranged in this space so that the distances between pairs of points have the strongest possible relation to the similarities among the pairs of objects. That is, two similar objects are represented by two points that are close together, and two dissimilar objects are represented by two points that are far apart. Unlike a projection operator such as PCA [31] or a manifold discovery algorithm [28], which gives superposition in the direction of projection, MDS creates a new representation based on the distances between points.

The algorithm for assigning the points in a lower dimensional space preserving the relative distances between points is based on the idea of stress majorization, which assigns the coordinates by minimizing a loss function based on the weighted sum of the differences between the euclidean distances on the original space and the distances on the representation plane. The loss function is defined as $Loss(X) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\left(Dist(x_i,x_j)-\delta_{i,j}\right)^2$, where $x_i, x_j \in \mathbf{R^m}$, the matrix X consist of the concatenation of the state vectors x_i and $\delta_{i,j}$ corresponds to the relative distance of the represented points i, j on the plane $\mathbf{R^2}$. The loss function can be minimized by using Scaling by majorizing a convex function (SMACOF) algorithm, which minimizes a quadratic form iteratively.

3. STAY-AWAY MECHANISM

The key insight of Stay-Away is that, the closer the resource usage resembles a contention that was previously responsible for QoS degradation, the more the sensitive application progresses towards a QoS violation. The Stay-Away runtime is a middleware between the VMs and the underlying resource, and runs on each host periodically following a three step mechanism: Mapping, Prediction and Action as shown in figure 2. All the three steps are performed in each period.

3.1 Mapping

Stay-Away employs a set of continuous VM-state resource usage snapshots to capture patterns of VM behaviour during application runs. Specifically, the runtime-learning phase begins by periodically measuring a set of VM metrics such as CPU, memory, I/O, network traffic for all VMs in the physical host as a vector of measurements M(t) = <VM$_i$-CPU, VM$_i$-Memory, VM$_i$-I/O, VM$_i$-network> for all VMs at time t. Stay-Away does not impose any limitation on the choice of metrics to be used. Ideally, the right metrics to use are those that characterize the load on the resource subsystem we are interested in. For example, performance counters for each VM can be used to characterize the load on the memory bus. These measurement vectors are then mapped into a lower dimension using MDS. This maintains the topological properties (relative distances) of the high dimensional space and similar measurement vectors remain close to each other, while dissimilar measurement vectors are placed farther apart. The mapped measurement vector in the lower dimensional space is called a *mapped-state*.

The mapping of each measurement vector over time captures the temporal behaviour of the execution. The path traced by the mapped-state is the trajectory of execution. Stay-Away relies on the application to report whenever a QoS violation happens in order to label the mapped state corresponding to the QoS violation. Alternatively, using IPC to detect QoS violation is explored in other works[34]. The *mapped-state* corresponding to a QoS violation is called a *violation-state*.

The measurement vectors can be mapped on to any lower dimension, we specifically selected a 2D representation for the following reasons:

- *Interpretability*: As MDS preserves the relative distances between measurement vectors, it helps understand the patterns and behaviours during the temporal evolution of the co-located VM execution. The metric preservation property of MDS also maintains the angles and the directions of the trajectories, which means that a prediction in a plane (y, x), gives a reliable representation of the behaviour on the high dimensional space.

- *Parameter Estimation*: A natural technique for forecasting in high dimensions is Vector Autoregressive Models (VAR)[31]. In high dimensional spaces, the number of samples needed for a reliable estimation of parameters by means of histograms (explained in section 3.2.3) increases exponentially with the dimensionality and also the domain of values for the parameters increases with increasing dimensions, leading to unreliable parameter estimation. A 2D representation of the trajectories gives prediction models with two parameters, which can be estimated reliably from a small sample.

3.2 Prediction

The second step of Stay-Away is to predict the future state of the execution based on the observations so far. The mapping phase produces a two-dimensional map which is then used by the predictor to forecast the transition of the execution behaviour. Specifically, we are interested only in knowing if the execution progresses towards a *violation-state*. Once it is fairly certain that a QoS violation is likely in the future, Stay-Away can then steer away from QoS violation by throttling the batch application. The predictor has 2 goals:

- To prevent an impending violation
- To allow the system to progressively learn about new violation states

These are conflicting goals, since, letting the applications to execute till a QoS violation happens in order to learn implies that a preventive action should not be taken. This is overly conservative on the batch applications and degrades the performance of sensitive applications. However,throttling the batch application based on incomplete information is overly

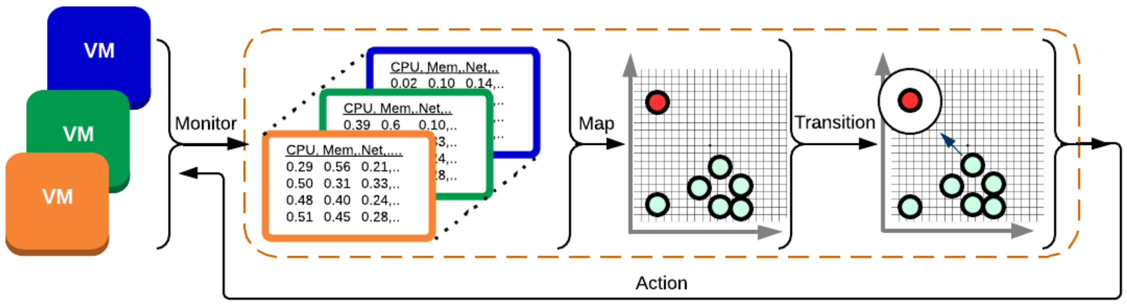


Figure 2: Stay-Away mechanism. This figure illustrates the 3 steps: Mapping, Predicting Transition and Action. The darker circle on the top of the map represents a QoS violation.

aggressive on the batch applications and restricts state-space exploration. The predictor needs to strike a balance between the two to consistently anneal to the correct QoS value without being overly aggressive nor overly conservative. In the following subsections, we explain how the predictor strikes a balance between the two and how a future *mapped-state* is estimated.

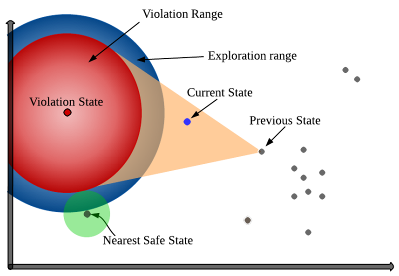


Figure 3: State-space exploration

3.2.1 *Should a prediction rely only on known QoS violations?*

When observing a resource contention causing a QoS violation, the system state is mapped on to the state-space and marked as *violation-state*. During every period, the predictor tries to estimate the position of the future *mapped-state*. If the estimated *mapped-state* overlaps with the *violated-state*, then a QoS violation is likely.

Any *mapped-state* or *violation-state* represent only the observed system behaviour. The *violation-states* also correspond to specific measurement vectors, and is marked as a point in the 2D space. If throttling of batch applications is done only based on exact overlap of the estimated *mapped-state* with *violation-state*, it limits the prediction to only seen states of violation. It is highly likely that the nearby neighbouring states around the *violation-state* also correspond to a QoS violation as they are separated only by minor deviations. For example, in a QoS violation corresponding to a particular VM consuming 70% of memory bandwidth, minor deviations such as 72% of memory bandwidth would still cause a QoS violation even though the exact value was not seen. If the predictor can take advantage by extending the range to account for unseen violations, it can prevent impending violations without having to capture the violation explicitly.

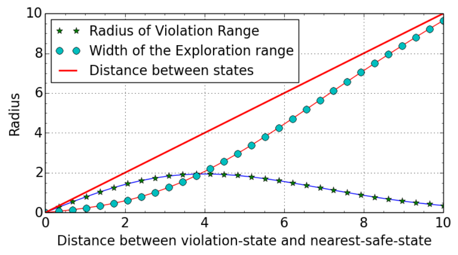


Figure 4: Variation of the radius of *violation-range* as distance between the *violation-state* and nearest *safe-state* varies

The unexplored neighbourhood area around the *violation-state* is called the *violation-range*, marked as a circle and is shown in figure 3. The *violation-range* is an approximation and corresponds to that area in the state space which the system hasn't seen yet but deems as the neighbourhood that would contain *violation-states* if a state were to be mapped in that range. Consequently, if an estimated *mapped-state* falls within a violation range, the batch application is throttled. Thus, a *violation-range* with a big radius would lead to aggresively throttling batch applications and a *violation-range* with a very small radius could lead to multiple QoS violations. In the next subsection, we explain how Stay-Away progressively attains accuracy and strikes a balance.The *exploration-range* is that neighbourhood which the system assumes safe. The predictor relies on a heuristic to predict the *violation-range* and progressively aims to attain accuracy. The area of the *violation-range* depends on the nearest known *safe-state* in the state-space. *Mapped-states* that do not correspond to a violation are called *safe-states*.

3.2.2 *How Far to explore?*

The radius of the *violation-range* is modelled as opposing forces (repulsion) between the *violation-state* and the nearest *safe-state*. The choice is intuitive: the closer there is a known *safe-state*, the lesser is the area of the *violation-range*. Initially the range is an approximation and as more states are explored, the representation gets more accurate. The radius of the *violation-range* is defined as the distance between *violation-state* and the nearest *safe-state* scaled by a Rayleigh distribution. It is important not to define the entire distance between a *violation-state* and the nearest *safe-state* as the radius of *violation-range* as it would prevent the system from exploring new states closer to the *violation-range*. A Rayleigh distribution is used to allow for the exploration range to adapt depending on the distance between the nearest *safe-state* and the *violation-state*. Ideally, the size of the exploration range should fade as the distance between the nearest *safe-state* and the *violation-state* gets closer. The radius of the violation range is given by:

$$R = de^{\frac{-d^2}{2c^2}}$$

where d is the distance between the nearest *safe-state* and the *violation-state*. c is defined as the median of the coordinate range of the mapped space. It follows from the observations that as the distance between these states increase, it is safer to increase the size of the *exploration-range* as we are more likely to be farther away from any unseen *violation-states*. However, as the distance between these states get closer, the exploration range needs to fade. Figure 4 shows the variation of the radius of violation range and the size of exploration range as the distance between these states vary.

3.2.3 When to act?

During the period of co-located execution, the system transitions through different states determined by the extent and type of resources being used by the VMs. In order to minimize the effect of performance interference, Stay-Away needs to predict any transition towards a violation and take a preventive action. The state transitions are specific to the applications running on the VM and can be:

- Gradual transitions, marked by resource consumptions such as memory usage where the memory allocated for the application typically varies gradually over time and as a result presents a consistent transition. Transitions are not marked by one application alone but instead by the combination of resource usage from all applications. As a result, the vector measurements in combination may or may not strictly follow a linear movement. However, the rate and pattern in transition becomes more apparent if the co-located application experiences minimal phase transition during its period of execution.

- Instantaneous transitions, marked by resource types such as CPU usage that could vary as instantaneous spikes. These sudden changes contribute to state transition in quick successions reducing the reaction time for any preventive action. For example, consider a *violation-state* characterised by a measurement vector with a contention at the level of CPU. If in future both applications contend for CPU, the transition to violated state is very quick giving almost no time for the system to react.

We have seen that the extent of *safe-states* are defined by the *exploration-range* and Stay-Away tries to avoid entering the *violation-range* to avoid violations. While these ranges act as a demarcation in space, the rate and direction of transition measure the temporal evolution or the progression over time.

State transitions are the result of complicated responses to an applications internal behaviour, and interactions between the co-located applications. It is not immediately obvious how a continuous movement process should be modelled and parameterized, since movement is multi-dimensional, combining both spatial and a temporal dimension. Because it is impossible to accurately model all of these interactions, they need to be modelled stochastically i.e. with intrinsically random velocities and orientations that can be summarized by well-defined probability densities and associated parameters. However, we observed that the accuracy of the prediction model suffers severely when all the state transitions are modelled using a single model. This is because every application has a characteristic behaviour and sometimes repetitive phases [26]. At any point in time, one of these 4 execution modes hold true:

- No application is running
- Batch application runs alone
- Latency-sensitive application runs alone
- Co-located execution of both batch application and latency-sensitive application

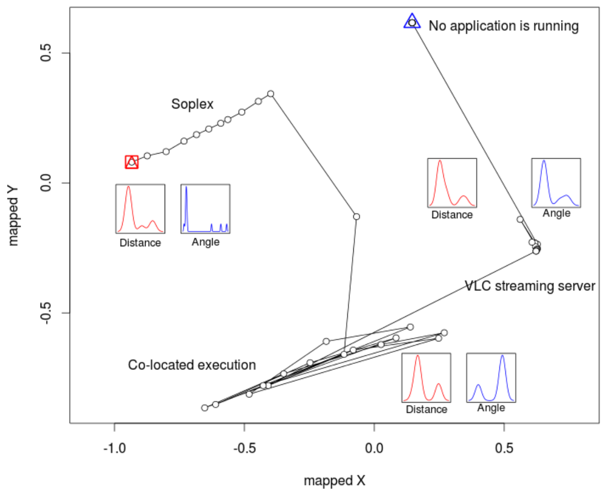


Figure 5: All 4 execution modes when VLC streaming is co-located with Soplex from SPEC CPU 2006

Figure 5 shows the state transition for an execution lifecycle comprising of VLC streaming server and Soplex from the SPEC CPU 2006 benchmark suite. The state transitions begins with no application running, followed by executing VLC streaming. Shortly after, the batch application is scheduled to execute and the states transition to co-located execution. Finally, VLC streaming finishes its execution and the batch application executes in an isolated fashion until it finishes. We can clearly see that each execution mode forms clusters and has a different pattern for trajectory. While VLC streaming is characterised by short bursts of correlated movement, Soplex follows a linear trajectory with a consistent orientation and slightly varying step length. Co-located execution on the other hand experiences an oscillating trajectory with bigger step lengths. As a result, modelling all the different execution modes using a single model fails to capture the inherent patterns and sequence specific to each execution mode. The trajectory pattern experienced in each of these execution modes is different. We experimented with numerous other co-locations of applications and observe that the life cycle of an execution steps through different modes and the trajectory pattern has a high dependence on the current execution mode. Our prediction model stems from this observation and as such no single prediction model can accurately model all the state transitions. The state transitions are broadly categorized into these 4 distinct execution modes, each with a different prediction model. Since Stay-Away runtime is a middleware managing the VMs, it can any time determine the current execution mode the system is in.

Marsh et al. [20] noted that a good description of the trajectory is achieved when the measured parameters and the relationships between them are sufficient to reconstruct characteristic tracks without loosing any of their significant properties while relying on a minimum set of relatively easily measured parameters. Based on a literature review [29], we have noticed that the trajectory can be fairly accurately modelled by the following parameters:

- Distance: The distance d between successive positions
- Absolute angle: The absolute angle α_i between the x direction and the step built by transitions from positions i and i+1

No static trajectory model can be assumed even within an execution mode for different sets of co-locations as each application has a different characteristic behaviour of its own. For a particular combination of batch application and latency sensitive application, co-located execution mode may show characteristics of a Biased Random Walk [9] whereas for a different combination, the execution mode may follow the trajectory model of levy flight [27]. Levy flight trajectories were observed for applications that experiences sudden phase changes. Because of these differences, we cannot rely on a static model and have to learn the behaviour during the execution.

To characterize the trajectories, we capture the behaviour of each execution mode by the probability density function (*pdf*) of the parameters: distance d and absolute angle α_i. The underlying measurement is a histogram. In Figure 5, we plot the smoothed version of the histogram using kernel density estimation and show the corresponding *pdf* of both distance and angles for different execution modes. The skew in the distribution indicates that the trajectory is biased and not random (with equal probabilities for all step lengths and angles). The bias indicates that the likelihood of certain step lengths and angles are higher than the others and this helps model the prediction with high accuracy. We experimented with many different set of applications and co-locations and always observe a bias in the trajectory. Therefore, after a few observations have been made, a first approximation of the *pdf*s for both parameters can be derived. A random set of samples are then generated following the histogram using the inverse transform method, which computes a mapping from a uniform distribution to an arbitrary distribution (i.e. the distribution from the histogram) [25]. This allows us to predict a set of new states around the current state and models the uncertainty in the likely position of the future state. Because of the inherent behavioural patterns in applications that cause a bias, with 5 samples to model uncertainty, we are able to achieve more than 90% accuracy on average for all the different co-locations we experimented with in section 7. Once the uncertainty is modelled, the generated states are examined to see if they fall within a violation range. Whenever a majority of the generated sample set fall within a violation range, Stay-Away takes an action to prevent degradation.

3.3 What Action to take and When to Stop?

To throttle the execution of the batch application, Stay-Away sends a SIGSTOP signal to pause the batch application and SIGCONT to resume its execution. Once paused, the system does not resume the batch application until the system believes that resuming the batch application will not cause a performance degradation for the sensitive application.This belief is based on the distance between the consecutive states of isolated execution of the sensitive application. Upon throttling, the system moves to a different execution mode. Only the sensitive application executes and prediction model is adapted to the new execution mode. Note that, it is impossible to have a violation in this execution mode as there is no interference. This also conforms in the state-space representation. If the performance-sensitive application continues to remain in the same phase or continues with the same workload after the batch application is paused, the states that follow roughly map to the same vicinity in the 2D space. An increase in the distance indicates a phase change or change in workload intensity of the sensitive application and is likely to have transitioned from contending for the bottleneck resource. Stay-Away has a learning parameter β, which is the maximum allowed distance between the states before resuming the batch application. Initially β is set to 0.01. Once the distance exceeds β, the system resumes the batch application. However, if resuming the batch application immediately leads to a violation, it indicates that the phase change of the performance-sensitive application was not enough to avoid degradation and the system increments β by a small amount. Over time, β attains accuracy. It is possible that the sensitive application does not experience any phase transition and in such scenarios, the batch application would starve indefinitely. To account for this, Stay-Away uses a random factor to resume the execution of the batch application when the distance falls below β for a long time. This is done in hope that the batch application may experience a phase transition and as such avoid degradation. However, if the batch application continues to degrade performance of the sensitive application, it is paused again.

4. OPTIMISATIONS AND OVERHEAD

In order to achieve efficiency, we need to address a few preprocessing problems before feeding the measurement vectors to MDS. Depending on the metrics chosen, the range of values for each metric may vary significantly. For example, while CPU usage ranges between 0 and 100, memory usage does not have a fixed upper limit as each VM could be assigned different amounts of memory. This variation causes higher values to introduce a bias that can affect the accuracy

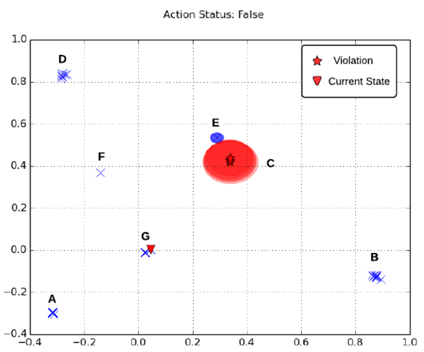


Figure 6: Snapshot of instantaneous transition of states when VLC transcoding is co-located with CPUBomb in the mapped space. Action status:False indicates that Stay-Away was not throttling the batch application during the snapshot

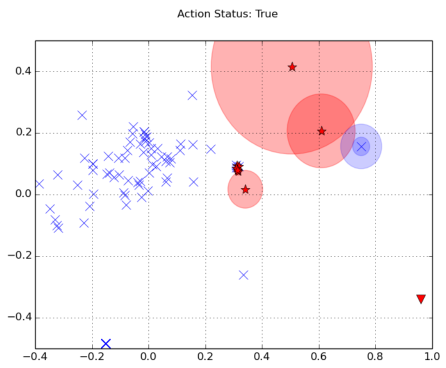


Figure 7: Snapshot of gradual transition of states when VLC streaming is co-located with Twitter-Analysis in the mapped space. Action status:True indicates that the batch application was being throttled during the snapshot

of MDS mapping. The problem is overcome by normalizing all the metric values between [0,1]. Normalisation also helps to cluster metrics with slight variations in the form of noise around the same neighbourhood forming visible clusters.

The SMACOF algorithm used to represent the high dimensional state to a lower dimension solves a quadratic form iteratively and can become computationally expensive as the number of samples increase. The cost of the algorithm is quadratic and we significantly reduce this overhead by choosing one representative sample from the set of samples that are very close to each other (Eucledian distance) and discarding other similar samples. We noticed that this optimization significantly reduces the computation time as it reduces the size of the observation matrix, while preserving the relative position of the different states, the temporal trajectories followed by the evolution of the execution, and their relative position with respect to the violation state. Alternatively, there is existing work in the literature that is capable of doing incremental MDS with high performance and very low overhead[32, 35]. The induced overhead by Stay-Away in terms of resource consumption is very minimal and corresponds to an average 2% CPU usage and negligible memory consumption.

5. SCALABILITY

When the number of dimensions increase, finding an optimal configuration of points in 2-dimensional space can become difficult. The best possible configuration in two dimensions may be a poor, highly distorted, representation of the data. This distortion will be reflected in a high stress value. When this happens, the only possible way to find an optimal configuration is to increase the number of dimensions in the mapped space. In our experiments, we found that the representation in a 2-dimensional space is always optimal with low stress value when there are 2 co-locations of VMs. It is not feasible to assume no more than one batch application for co-location. This can, however, be easily circumvented by considering all the batch applications as one logical VM. The monitored metrics of all the batch application are aggregated together to model their collective behaviour as a single logical VM. Since resources are shared between all the batch applications, contention can be accurately represented by a linear composition of resource usage values. However, identifying the specific batch application responsible for contention becomes difficult by considering their collective behaviour. Apart from the difficulty and the computation involved in identifying the specific interfering batch application, it is also possible that a single batch application alone does not cause a QoS degradation, but a set of batch applications cause a contention. While one batch application may cause a contention for CPU, another application may contend at the level of memory subsystem. The overhead involved in identifying the specific batch applications responsible for contention exceeds the benefits gained. For this reason, upon detecting a transition towards QoS violation, batch applications consuming a majority share of resources are collectively throttled. Alternatively, groups of batch applications can be iteratively throttled till QoS is guaranteed and each violation state labelled to identify groups of batch application responsible for that specific type of violation. In our current implementation, we collectively throttle the batch applications consuming a major share of resources.

6. TEMPLATE PROPERTIES

In case of repeatable latency sensitive applications, the *violation-states* in the generated map from a previous execution can be used as a starting point and is a valid map for a new execution with a different batch application. The state representation for a performance sensitive application is independent of the specific batch applications running on the co-located virtual machines. Although the generated map from the co-located execution depends on the specific batch application that was co-located, the *mapped-states* themselves are representative of load at the resource level. As a result, the captured states for a performance sensitive application doubles as a template for the latency sensitive application that can be used for future executions alongside a different set of application co-locations. For example, consider a latency sensitive application(L) co-located with a batch application(B_A). Their co-located execution generates a map(map-A) with safe states and violated states. The

Workload Name	Combination
Batch-1	Twitter-Analysis+Soplex
Batch-2	Twitter-Analysis+MemoryBomb

Table 1: Combination of Batch Applications

violated-states represent QoS violations. If we execute the same latency sensitive application (L) with another batch application (B_B), the *violated-states* from map-A would still correspond to a valid *violation-state* for the new execution. The batch application (B_B) may never map a state in that *violation-state*, but if the co-located execution were to map a state, it will be a *violation-state*.

7. EVALUATION

We conducted our experiments with our Stay-Away prototype on a 3.2 GHz dual-socket Intel Core i5 CPU with 4 cores. Each core has a 32KB L1 private data cache, a 32KB L1 private instruction cache, a 256 KB L2 cache and a shared 4 MB L3 cache. The OS is Ubuntu with GNU/Linux kernel version 3.5.0-22.

7.1 Experimental Setup

We chose Linux Containers (LXC)[3] because of its ability to provide near-native performance for applications even though it is highly susceptible to performance interference [33]. To evaluate Stay-Away, we use two different types of latency-sensitive applications and show that LXC combined with Stay-Away can achieve a high degree of QoS while improving machine utilisation. We conducted experiments using the VLC[4] media player for video streaming and a Webservice with a memory intensive, CPU intensive and a mix of both CPU and memory intensive workload as the performance sensitive application. Soplex from SPEC CPU 2006 benchmark [15], Twitter influence ranking from the Cloud Suite benchmark [1], CPUBomb from the isolation benchmark suite[21], VLC transcoding and a custom synthetic application that stresses the memory(Memory Bomb) were used as batch applications. Memory Bomb generates stress on the memory subsystem by allocating large chunks of memory and occasionally reading the allocated content. In order to evaluate the QoS and utilization with more than one co-location of a batch application, we setup two different combinations of batch applications shown in table 1. Each of the batch application were executed in a different LXC container. We instrumented the source code of VLC 2.0.5 to capture performance metrics when using VLC to stream a movie in real time to clients. The minimum transcoding rate required to provide real time viewing without any loss of frames at the server side is defined as the QoS threshold.

The Webservice is setup for analysing and serving data. It consists of a Memcached layer for in-memory data storage and performs analytics, if necessary, before serving the data. The data used for storage and analysis is the open dataset available from [2] and contains periodic network topology information and monitored host metrics of more than 80 nodes which are a part of the community-lab testbed [7]. The Webservice is capable of performing statistical analysis and aggregation of data for each monitored metric and to serve requested data for any specific period. The workload comprises of CPU intensive, Memory intensive and mix of CPU and memory intensive operations.

We begin by illustrating the state-space representation during different execution modes. To illustrate both instantaneous and gradual transitions, we first run two batch applications: transcoding a video with VLC in one LXC container and CPUBomb in another. We chose this co-location to simplify the illustration as both batch applications experience minimal phase transitions during isolated execution. In this contrived, yet representative example for illustrating state space transitions, a violation is said to have occurred when the rate of transcoding frames fall below a certain threshold. Figure6 shows a snapshot of the different states the system experiences during the execution life cycle. Darker points represents states with minor transitions that maps closer to each other. In figure 6, A corresponds to the state when only CPUBomb was executing. B corresponds to the state when VLC-transcoding runs alongside CPUBomb and C represents the violation state. When Stay-Away takes an action to prevent violation, the corresponding state experienced is represented by D. States E,F,G represent the states during the period of transition. Figure 7 shows the gradual transition observed when VLC was used as a streaming server (performance sensitive) and co-located with the Twitter-Analysis (batch).

7.2 QoS and Utilization

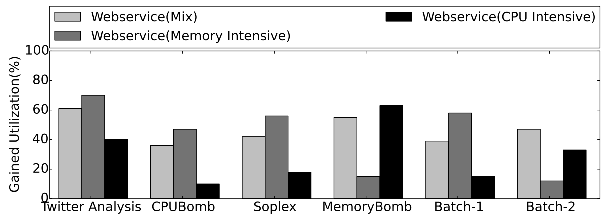


Figure 12: Gained Utilization when Webservice is co-located with different Batch Applications

We first evaluate the effectiveness of Stay-Away for enforcing targeted QoS without any prior profiling of the application. Figures 8 and 9 show the normalised QoS of VLC streaming server when co-located with CPUBomb and Twitter-Analysis respectively. The minimum transcoding rate for the VLC server to ensure uninterrupted delivery of frames is shown as QoS threshold. Whenever the rate of transcoding falls below this threshold the clients experience degradation in the quality of service. We can see from figure 8 and 9 that without any prevention the system experiences numerous violations as both these batch applications contend for resources with VLC streaming. The qualitative effect of Stay-Away on QoS becomes clear when reproducing the streamed video. Qualitatively, a choppy reproduction without Stay-Away transforms into a smooth playback when including Stay-Away. The QoS degradations are considerably reduced and most violations seen are in the early phase of execution. This is because the system is unaware of the states that correspond to a violation in the early phase and once seen, the system proactively prevents future violations. Other violations arise from factors such as: instantaneous jumps to violation states characterised by sudden increase in the use of CPU.

Figures 10 and 11 show the gain in machine utilisation from co-location. Gained utilisation is the gain in utilisation

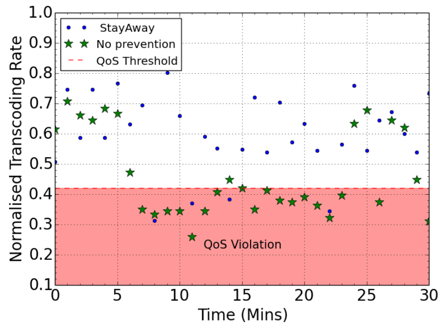


Figure 8: VLC with CPUBomb

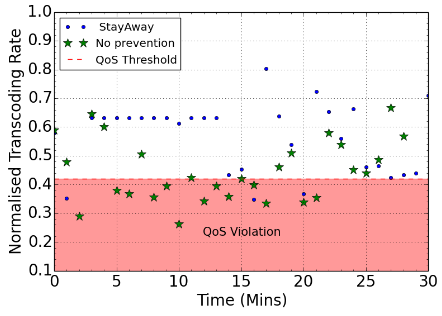


Figure 9: VLC with Twitter-Analysis

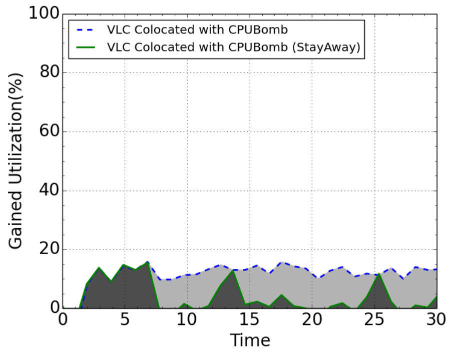


Figure 10: Gained Utilisation with CPUBomb

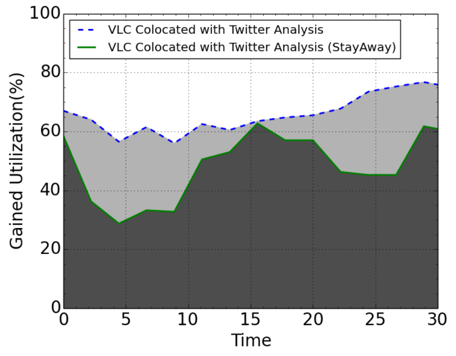
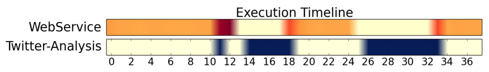
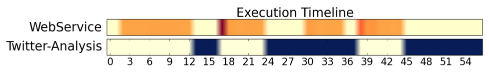


Figure 11: Gained Utilisation with Twitter-Analysis



(a) Webservice(CPU Intensive) co-located with Twitter-Analysis



(b) Webservice(mix) co-located with Twitter-Analysis

Figure 13: The colour gradient for Webservice is a measure of the stress on its performance. Darker colour indicates higher stress. Dark colour bands for Twitter-Analysis represents period of its execution and lighter colour bands represents the period it is throttled.

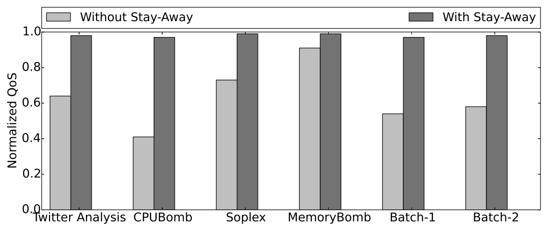


Figure 14: QoS of Webservice with a mix of CPU and Memory intensive workload when co-located with different Batch Applications

in comparison to executing VLC streaming service without any co-location. The upper band in the figure shows the maximum utilisation that can be gained by co-locating the batch application with VLC streaming service without any prevention from performance interference. With Stay-Away deployed, we are able to achieve a good balance in improving machine utilisation, shown in the lower band from figures 10 and 11, while still guaranteeing a high level of QoS. The gain in machine utilisation depends on the characteristics of the co-located batch application. In our setup, VLC streaming with Twitter-Analysis gains an average of 50% machine utilisation when compared to an isolated run of the VLC streaming server. The system gains substantial improvement in both utilisation and QoS. This is because Stay-Away throttles only when the system progresses toward resource contention. Co-location with CPUBomb as the batch application is the worst case scenario since the batch application constantly contends for CPU and does not experience any phase transition. As a result the gained machine utilisation, as shown in figure 10, is in spikes as Stay-Away throttles in an attempt to mitigate performance degradation. The gain in utilisation for CPUBomb is about 5% because CPUBomb constantly consumes CPU and it is impossible to execute both VLC streaming and CPUBomb without violating the QoS. However, with Stay-Away deployed, it learns this contention and guarantees a high level of QoS.

Figures 14, 15 and 16 show the QoS achieved when different batch applications are co-located with Webservice for different types of workload. We can see that with Stay-Away, a high level of QoS is guaranteed. Figure 12 shows the gained utilisation when Webservice is co-located with different batch applications. The gained utilisation is different for different types of workload and the gain is maximum when Twitter-Analysis is co-located with Webservice

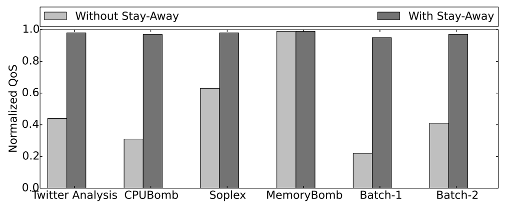


Figure 15: QoS of Webservice with CPU intensive workload when co-located with different Batch Applications

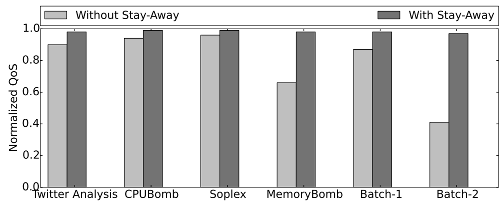


Figure 16: QoS of Webservice with Memory intensive workload when co-located with different Batch Applications

for a memory intensive workload. This is because Twitter-Analysis experiences a mix of both CPU and memory intensive phases, and is throttled only during its memory intensive phase. The effect of performance interference caused by Twitter-Analysis is seen only when its memory operation is intensive enough to force the OS to swap pages of Webservice to disk, causing a degradation in response time. As a result, Twitter-Analysis is throttled only when it performs extensive memory operations. The gained utilisation is relatively low when batch application is co-located with CPU intensive workload of Webservice because all batch applications except MemoryBomb are mostly CPU intensive and interfere negatively with the performance of Webservice.

Figure 13a shows the execution timeline when Web service with a CPU Intensive workload is co-located with Twitter-Analysis. We vary the intensity of workload to show that Stay-Away is capable of detecting and using such periods of low utilisation without violating QoS. The stress on Webservice is measured by monitoring the number of transactions completed per second and is done only to illustrate the functioning of our middleware. Stay-Away does not have any access to this information. Twitter-Analysis begins execution at timestamp 10. This causes a stress on Webservice and leads to a QoS violation shown by a dark band. Stay-Away learns this and immediately throttles Twitter-Analysis. Soon after this, there is a period of low workload, which Stay-Away detects and the execution of Twitter-Analysis is resumed. Subsection 3.3 explains how Stay-Away detects this. At timestamp 18, the workload of Webservice increases and the execution of Twitter-Analysis begins to cause a stress on its performance, but hasn't violated QoS yet. Stay-Away predicts this and throttles Twitter-Analysis before a QoS violation happens. Figure 13b shows the execution timeline when Web service with a mix of CPU and memory intensive workload is co-located with Twitter-Analysis. We introduce a period of change in the workload phase from timestamp 30 to 36. We can see from the figure that Twitter-Analysis executes in an uninterrupted manner during this period as Stay-Away identifies the period as a phase change and believes that executing Twitter-Analysis would cause no stress. Stay-Away detects this from the state space mapping as the change in phase maps the corresponding states to a different space in the map and farther away from the violation state.

7.3 Template Validation

This section validates how a map generated for a latency sensitive application can be reused for future executions with a different set of batch applications. We conduct an experiment by streaming a video file using VLC running alongside CPUBomb as the batch application. The Stay-Away component is active during the run, capturing the states and preventing violation. Figure 17 shows a snapshot of the states that characterises the VLC streaming service for a given video and is used as the template for future executions of VLC alongside a different batch application. In order to validate that the captured states correspond to the properties of VLC independent of the co-location with any specific batch application, we use the template as the initial state of VLC for streaming the same file alongside Soplex. We disable the Stay-Away component from taking any action to show that the states corresponding to violation in figure 17 continue to correspond to violation alongside Soplex. Figure 18 shows a snapshot of the state of VLC run alongside Twitter-Analysis. While new states are seen during the execution, we can see that there are more violations and they correspond to the area characterised by violations from figure17.

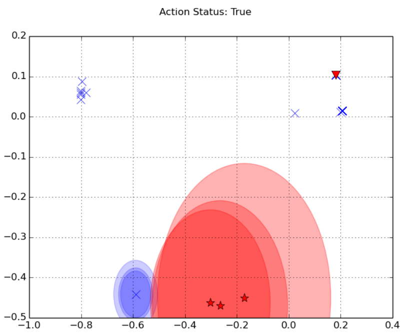


Figure 17: Template with CPUBomb

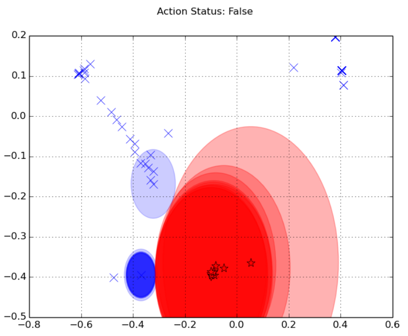


Figure 18: VLC with soplex

8. RELATED WORK

While there has been a lot of work on mitigating the performance interference due to resource contention, not much work consider run-time models that can adapt to dynamic changes. DejaVu [30] relies on a online-clustering algorithm to adapt to load variations by comparing the performance of a production VM and a replica of it that runs in a sandbox to detect interference. It mitigates interference by overprovisioning resources. Unfortunately, DejaVu has a high cost as it requires an additional sand-box for executing the replica. Stay-Away has no such overhead. A similar system, DeepDive [24], first relies on a warning system running in the VMM to conduct early interference analysis. When the system suspects that one or more VMs are subjected to interference, it clones the VM on-demand and executes it in a sandboxed environment to detect interference by comparing the differences in relevant measured metrics.If interference does exist, the most aggressive VM is migrated on to another physical machine. It incurs overhead in the form of cloning and migrating VMs. Migrating VMs is an expensive and time consuming operation. One advantage of Stay-Away is that it co-locates best effort batch applications with latency sensitive application. By introducing this constraint, the need for migrations are avoided since they can be throttled. Throttling involves very low overhead and has an immediate effect as opposed to migration.

Another class of work has also investigated providing QoS management for different applications on multicore [14, 22, 16]. While demonstrating promising results, resource partitioning typically requires changes to the hardware design, which is not feasible for existing systems.

Recent efforts [12, 37, 18] demonstrate that it is possible to accurately predict the degradation caused by interference by prior analysis of workload. However, in practice, applications are not available prior to their deployment and often run for a long time, so it is not feasible to perform this analysis. In [19] the application is profiled statically to predict interference and identify safe co-locations for VMs. It mainly focuses on predicting which applications can be co-run with a given application without degrading its QoS beyond a certain threshold. The limitation of static profiling introduces a lack of ability to adapt to changes in application dynamic behaviour. Paragon [16] tries to overcome the problem of complete static profiling by profiling only a part of the application and relies on a recommendation system, based on the knowledge of previous execution, to identify the best placement for applications with respect to interference. Since only a part of the application is profiled, dynamic behaviours such as phase changes and workload changes are not captured and can lead to a suboptimal schedule resulting in severe performance degradation. Our work is complementary to Paragon. If Paragon can be used with constraints for placing latency critical applications with batch applications, Stay-Away can be used to guarantee high level of QoS and utilization even if the schedule turns out to be suboptimal.

Bubble-Flex [34] is a runtime method for mitigating performance interference by phasing in and phasing out batch applications upon detecting QoS violations. Bubble-Flex is the closest work to our approach in the sense that it is a runtime method and categorizes application to latency sensitive and batch applications. It predicts QoS violations by generating load in the memory subsystem in bursts during runtime to generate a sensitivity curve. Their approach introduces additional stress on the memory subsystem, which can itself cause interference and batch applications are unable to fully exploit periods of low utilization. Stay-away guarantees QoS without itself causing any interference and also letting batch applications fully exploit periods of low utilisation.

Q-Clouds[23] is another system that aims to guarantee QoS in the face of interference. It achieves this by giving unallocated resources to an application to prevent falling below the QoS requirement.Q-Clouds improves performance as long as there is headroom available. If no headroom is available, it cannot guarantee QoS and the excess headroom when not used (Workload and phase changes) leads to lower machine utilisation. Stay-Away does not need any excess headroom and a high level of QoS is always guaranteed.

Koh et al [17]. propose a technique for predicting performance degradation of co-located applications based on their resource usage statistics. When a new application is hosted, its resource vector is compared with that of known applications and is mapped to the weighted average of one or more known applications whose resource vectors it closely resembles. Then the performance degradation of the new application is predicted based on already recorded performance degradation of the mapped/representative known applications. It requires the need to have an already profiled application that closely resembles an incoming application and does not capture the temporal properties of the application. Stay-Away takes into account both the spatial and temporal properties of the application.

9. CONCLUSION

This paper introduces Stay-Away, a generic and adaptive mechanism to mitigate the detrimental effects of performance interference on sensitive applications when co-located with other batch-like applications and improve resource utilization. Unlike earlier previous work that requires apriori knowledge and static models, Stay-Away continuously learns and maps to a state-space representation the favourable and unfavourable states of execution among multiple VMs. The representation allows to visualise and interpret co-located VM execution. This is used to predict real-time transitions of the co-located VM states continuously and prevent performance degradation in selected VMs. Additionally, we discuss how this mechanism doubles as a template engine for repeatable experiments or services.

The evaluation of a proof-of-concept prototype of Stay-Away with Linux Containers and several batch and interactive applications confirm the expected effect, validity and stability of the mechanism in general.

10. ACKNOWLEDGEMENTS

This work was partially supported by the European Community through the projects Community Networks Testbed for the Future Internet (CONFINE): FP7-288535, A Community Networking Cloud in a Box (CLOMMUNITY): FP7-317879, partially financed by SpeechTech4All (TEC2012-38939-C03-02), by Spanish government under contract TIN2013-47245-C2-1-R and in part by the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC) funded by the Education, Audiovisual and Culture Executive Agency (EACEA) of the European Commission under the FPA 2012-0030..

References

[1] Cloud Suite Benchmark. `http://parsa.epfl.ch/cloudsuite/`.

[2] Confine Open Dataset. `https://wiki.confine-project.eu/experiments:datasets`.

[3] Linux Containers. `http://lxc.sourceforge.net`.

[4] VLC. `http://www.videolan.org/vlc/index.html`.

[5] Wikipedia Trace Data. `https://aws.amazon.com/datasets/6025882142118545`.

[6] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE computer*, 40(12):33–37, 2007.

[7] B. Braem, C. Blondia, C. Barz, H. Rogge, F. Freitag, L. Navarro, J. Bonicioli, S. Papathanasiou, P. Escrich, R. Baig Viñas, et al. A case for research with and on community networks. *ACM SIGCOMM Computer Communication Review*, 43(3):68–73, 2013.

[8] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[9] E. A. Codling. *Biased random walks in biology.* PhD thesis, The University of Leeds, 2003.

[10] T. F. Cox and M. A. Cox. *Multidimensional scaling.* CRC Press, 2010.

[11] C. Delimitrou and C. Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems*, pages 77–88. ACM, 2013.

[12] M. Dobrescu, K. Argyraki, and S. Ratnasamy. Toward predictable performance in software packet-processing platforms. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 11–11. USENIX Association, 2012.

[13] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica. Choosy: max-min fair sharing for datacenter jobs with constraints. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 365–378. ACM, 2013.

[14] F. Guo, Y. Solihin, L. Zhao, and R. Iyer. A framework for providing quality of service in chip multi-processors. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 343–355. IEEE Computer Society, 2007.

[15] J. L. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[16] R. Iyer, L. Zhao, F. Guo, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. Hsu, and S. Reinhardt. Qos policies and architecture for cache/memory in cmp platforms. In *ACM SIGMETRICS Performance Evaluation Review*, volume 35, pages 25–36. ACM, 2007.

[17] Y. Koh, R. C. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu. An analysis of performance interference effects in virtual environments. In *ISPASS*, pages 200–209, 2007.

[18] J. Machina and A. Sodan. Predicting cache needs and cache sensitivity for applications in cloud computing on cmp servers with configurable caches. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE, 2009.

[19] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 248–259. ACM, 2011.

[20] L. Marsh and R. Jones. The form and consequences of random walk movement models. *Journal of Theoretical Biology*, 133(1):113–131, 1988.

[21] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens. Quantifying the performance isolation properties of virtualization systems. In *Proceedings of the 2007 workshop on Experimental computer science*, page 6. ACM, 2007.

[22] M. Moreto, F. J. Cazorla, A. Ramirez, R. Sakellariou, and M. Valero. Flexdcp: a qos framework for cmp architectures. *ACM SIGOPS Operating Systems Review*, 43(2):86–96, 2009.

[23] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European conference on Computer systems*, pages 237–250. ACM, 2010.

[24] S. Novakovic, D. Novakovic, R. Bianchini, D. Kostic, and N. Vasic. Deepdive: Transparently identifying and managing performance interference in virtualized environments. Technical report, 2013.

[25] A. P. Probability. Random variables and stochastic processes. *McGrow, Hill Series Elastical Eng, NY*, 1984.

[26] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder. Discovering and exploiting program phases. *Micro, IEEE*, 23(6):84–93, 2003.

[27] M. F. Shlesinger and J. Klafter. Lévy walks versus lévy flights. In *On growth and form*, pages 279–283. Springer, 1986.

[28] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[29] F. Urbano, F. Cagnacci, C. Calenge, H. Dettki, A. Cameron, and M. Neteler. Wildlife tracking data management: a new vision. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 365(1550):2177–2185, 2010.

[30] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini. Dejavu: accelerating resource allocation in virtualized environments. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 423–436. ACM, 2012.

[31] D. Williams and D. Williams. *Weighing the odds: a course in probability and statistics*, volume 548. Springer, 2001.

[32] M. Williams and T. Munzner. Steerable, progressive multidimensional scaling. In *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, pages 57–64. IEEE, 2004.

[33] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pages 233–240. IEEE, 2013.

[34] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 607–618, New York, NY, USA, 2013. ACM.

[35] T. Yang, J. Liu, L. McMillan, and W. Wang. A fast approximation to multidimensional scaling. In *Proceedings of the ECCV Workshop on Computation Intensive Methods for Computer Vision (CIMCV)*, pages 354–359, 2006.

[36] J. Zhang and R. J. Figueiredo. Application classification through monitoring and learning of resource consumption patterns. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10–pp. IEEE, 2006.

[37] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 129–142. ACM, 2010.