MORPHOSYS: Efficient Colocation of QoS-Constrained Workloads in the Cloud

VATCHE ISHAKIAN   AZER BESTAVROS
visahak@bu.edu   best@bu.edu
Computer Science Department, Boston University
Boston, MA 02215, USA

***Abstract*—In hosting environments such as IaaS clouds, desirable application performance is usually guaranteed through the use of Service Level Agreements (SLAs), which specify minimal fractions of resource capacities that must be allocated for unencumbered use for proper operation. Arbitrary colocation of applications with different SLAs on a single host may result in inefficient utilization of the host's resources. In this paper, we propose that periodic resource allocation and consumption models – often used to characterize real-time workloads – be used for a more granular expression of SLAs. Our proposed SLA model has the salient feature that it exposes flexibilities that enable the infrastructure provider to safely transform SLAs from one form to another for the purpose of achieving more efficient colocation. Towards that goal, we present MORPHOSYS: a framework for a service that allows the manipulation of SLAs to enable efficient colocation of arbitrary workloads in a dynamic setting. We present results from extensive trace-driven simulations of colocated Video-on-Demand servers in a cloud setting. These results show that potentially-significant reduction in wasted resources (by as much as 60%) are possible using MORPHOSYS.**

I. INTRODUCTION

Motivation: *Cloud computing* in general and Infrastructure as a Service (IaaS) in particular have emerged as compelling paradigms for the deployment of distributed applications and services on the Internet due in large part to the maturity and wide adoption of virtualization [1].

From the perspective of an IaaS customer, this paradigm shift presents both an opportunity and a risk. On the one hand, deploying applications in the cloud is attractive because it enables efficiency through elastic scaling to match customer demand. On the other hand, deploying applications in the cloud implies relinquishing QoS monitoring and control to the cloud. Mitigating that risk requires the establishment of a "contract" – a Service Level Agreement (SLA) – between the provider and the customer, which spells out minimal resource allocations that the customer believes would satisfy desirable QoS constraints, while also being verifiable through measurement or auditing of allocated resources. Indeed, providing trustworthy accountability and auditing features have been cited as key attributes that would increase cloud adoption [2].

From the perspective of an IaaS provider, the cloud value proposition is highly dependent on efficient resource management [3], and hence reduced operational costs by capitalizing on energy savings [4] and on improved scalability [5]. Such efficiencies need to be achieved while satisfying the aforementioned contractually-binding customer SLAs. This necessitates that SLAs be spelled out in such a way so as to expose potential flexibilities that enable efficient mapping of physical resources to virtualized instances.

Given the wide range of applications currently supported in an IaaS setting – not to mention envisioned services, *e.g.*, in support of cyber-physical systems [6] – it would be impractical for an IaaS provider to support special-purpose SLAs that are tailor-made for each such application and service, and which can be efficiently audited. Rather, a more practical approach calls for the development of a common language for expressing SLAs – a language that would cater well to the widely different types of applications that are likely to be colocated on an IaaS infrastructure.

Currently, the de-facto language for expressing SLAs mirrors how virtual machines are provisioned – namely through the specification of resource capacities to be allocated on average, over fairly long time scales. While appropriate for many applications, such coarse SLAs do not cater well to the needs of applications that require resource allocations at a more granular scale, *e.g.*, through the specification of a worst-case periodic resource consumption in support of interactive applications.

To elaborate, recent studies have documented the often unacceptable performance of a number of application classes in a cloud setting. Examples include latency-sensitive, interactive, image acquisition applications, IP telephony and streaming applications [7], [8]. The main culprit for the degraded performance is the high variability associated with the time-scale of resource allocation in a virtualized environment. This suggests the need for finer-grain SLA specifications that enable applications to spell out their resource needs over arbitrary time scales, as well as any tolerable deviations thereof (flexibilities).

Recognizing this need, in this paper we propose an expressive periodic resource allocation model for the specification of SLAs – a model that on the one hand provides customers with a larger degree of control over the granularity of resource allocation, and on the other hand enables providers to leverage flexibilities in customers' SLAs for the efficient utilization of their infrastructures. Our SLA model is equally expressive for traditional cloud application as well as for the aforementioned QoS-constrained applications; it enables providers to cater to a wider customer base while providing them with the requisite measurement and auditing capabilities.[1]

This research was supported in part by NSF awards #0720604, #0735974, #0820138, #0952145, and #1012798.

[1]Virtualization is not only enabling IaaS offerings, but also it is enabling a cloud-like management of resources in traditional settings such as the "Single Chip" cloud computer [9]. Thus, while IaaS motivates the MORPHOSYS framework presented in this paper, we note that this framework is also applicable in traditional multiprocessing settings.

Scope and Contributions: Given a set of applications (workloads), each of which specified by minimal resource utilization requirements (SLAs), the problem we aim to address is that of colocating or mapping these workloads efficiently to physical resources. To achieve efficient mapping, we need to provide workloads with the ability to express potential flexibilities in satisfying their SLAs. For instance, an SLA spelled out as a fixed periodic allocation may not expose potential workload flexibililty over some minimal frequency of allocation. Thus, we recognize that it could be the case that there are multiple, yet functionally equivalent ways to express the resource requirements of a QoS-constrained workload. Towards that end, we propose a specific model for SLAs that makes it possible for providers to rewrite such SLAs as long as such rewriting is safe. By safety, we indicate that we can substitute the original SLA by the rewritten SLA without violating the original SLA, and that the resources allocations that satisfy the rewritten SLA would also provably satisfy the original SLA. The ability to make such safe SLA transformations enables providers to consider a wider range of colocation possibilities, and hence achieve better economies of scale. In that regard, we present MorphoSys:[2] the blueprints of a colocation service that demonstrates the premise of our proposed framework. Results from extensive trace-driven simulations of colocated Video-on-Demand (VOD) servers in a cloud setting show that potentially-significant reduction in wasted resources (by as much as 60%) are possible using MorphoSys.

Paper Overview: The remainder of this paper is organized as follows. In Section II we present some background and basic concepts underlying our SLA model. In Section III, we introduce our SLA model for QoS-constrained resource supply and demand (for hosts and workloads, respectively), along with necessary notation and basic definitions. In section IV, we present the basic elements of our MorphoSys framework. In Section V, we present experimental results that demonstrate the promise from using MorphoSys to manage colocated streaming servers in a dynamic environment. In Section VI, we review relevant related work, and we conclude in Section VII with closing remarks and future research directions.

II. SLA Model: Basics

Recall that an important consideration for efficient colocation is the ability of a provider to evaluate whether a given set of customers can be safely colocated. To do so, a provider must be able to decide whether the capacity of a given set of resources (*e.g.,* a host) can satisfy the aggregate needs of a set of customers Given our adopted periodic model for SLA specification, it follows that evaluating the feasibility of colocating a set of customer workloads on a given host can be viewed as a "schedulability" problem: given the capacity of a host, are a set of periodic real-time tasks schedulable?[3]

Different models and schedulability analysis techniques have been proposed in the vast real-time scheduling theory, including Earliest Deadline First and Rate Monotonic Analysis (RMA) [10], and Borrowed Virtual Time [11], among others. While similar in terms of their high-level periodic real-time task models, these approaches differ in terms of the trade-offs they expose vis-a-vis the complexity of the schedulability analysis, the nature of the underlying resource manager/scheduler, and the overall achievable system utilization. Without loss of generality, we assume that RMA [10] is the technique of choice when evaluating whether it is possible to co-locate a set of periodic workloads on a fixed-capacity resource.

Liu and Layland [10] provided the following classical result for the schedulability condition of n tasks (SLAs), each of which requiring the use of a resource for C_i out of every T_i units of time, under RMA:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(\sqrt[n]{2} - 1)$$

Follow-up work, by Lehoczky *et al* [12] showed that by grouping tasks in k clusters such that the periods of tasks in each cluster are multiples of each other (*i.e.*, harmonic), a tighter schedulability condition is possible – namely:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq k(\sqrt[k]{2} - 1)$$

As motivated above, there may be multiple yet functionally-equivalent ways to satisfy a given SLA. This flexibility could be leveraged by a provider for efficient colocation. In particular, given a set of periodic tasks (SLAs), it might be possible to obtain clusters of tasks with harmonic periods by manipulating the period C_i or the periodic allocation T_i of some of the tasks (SLAs) in the set. For such a transformation to be possible, we must establish that it is *safe* to do so.

In earlier work of ours [13], we provided a type-theoretic formalism for studying transformations of real-time workloads, In that work, we used a generalized type for the specification of real-time task SLAs, comprising a quadruple of natural numbers (C, T, D, W), $C \leq T$, $D \leq W$, and $W \geq 1$, where C denotes the resource capacity demanded in each allocation interval T, and D is the maximum number of times that the task could tolerate missing an allocation in a window consisting of W allocation intervals. We also showed that this model is expressive enough to represent different categories of real-time tasks, and established a set of provably safe transformations on real-time workloads to enable efficient multiprocessor scheduling.

In this work, we exploit a subset of theorems from [13] as a building block in our MorphoSys framework. These theorems provide us with the means to rewrite SLA $\mathcal{S}$ as SLA $\mathcal{S}'$ such that resource allocations that satisfy $\mathcal{S}'$ would provably also satisfy $\mathcal{S}$. Theorems 1 - 4 provide different ways of scaling up/down either or both of the period C and the periodic allocation T for a an SLA. In general, satisfying SLA $\mathcal{S}'$ would require more of the underlying resources than the original SLA $\mathcal{S}$. Nevertheless, such a transformation may be advantageous to the IaaS provider as it may result in a more efficient colocation of customer workloads (tasks) – *e.g.*, by

[2]MorphoSys can be seen as catalyzing the "morphosis" of a set of SLAs – namely, morphing SLAs to enable more efficient colocation.

[3]In this paper, we often use terms that hearkens back to CPU scheduling terminologies, noting that our work applies equally to computational as well as other consumable resources (*e.g.*, networking and I/O bandwidth).

making such workloads harmonic and hence subject to looser schedulability bounds [12].

In the following theorems from [13], we make the simplifying assumption that $D = 0$ and $W = 1$ – *i.e.*, SLAs are strict; applications do not tolerate missed allocations.

Theorem 1. *Given an SLA of type (C, T), a transformed SLA (C', T') satisfies (C, T) if one of the following conditions holds:*

1) $T' \leq T/2$ *and* $C' \geq C/(K - 1)$ *where* $K = \lfloor T/T' \rfloor$.
2) $T' > T$ *and* $C' \geq T' - (T - C)/2$.
3) $T/2 < T' \leq T$ *and* $T' - (T - C)/3 \leq C'$.

Theorem 2. *Let $\tau = (KC, KT)$ be an SLA type for some $K \in \mathbb{N}$, and $K \geq 1$ then the SLA of type (C, T) satisfies τ.*

Theorem 3. *Let $\tau = (C, T)$ be an SLA type, and $\tau' = (C', T')$ be another SLA type, where $T' = KT$ for some $K > 1$ then τ' satisfies τ if $(K - 1) * T + C \leq C' \leq T'$*

Theorem 4. *Let $\tau = (C, T)$ be an SLA type and $\tau' = (C, T')$ be an SLA type, where $(T + C)/2 < T' < T$ and $C \leq T'$. If $m = \mathrm{lcm}(T, T')/T$, and $n = \mathrm{lcm}(T, T')/T'$ where lcm is the least common multiple, then we can guarantee at least $s = n - m + 1$ satisfied intervals out of total m intervals. Therefore τ' satisfies τ if $s = m$.*

III. SLA Model: Fluidity

As we alluded before, we believe that a periodic resource allocation model is appropriate for expressing SLAs in an IaaS setting. Thus, in this section we extend the real-time task SLA model proposed in [13] for the purpose of expressing general SLAs of IaaS customer workloads – which may not be inherently “real time”. In particular, we extend the SLA model to allow for the modeling of “fluid” workloads.

A *fluid workload* is one that requires *predictable* periodic allocation of resources (*i.e.* not best effort), but has flexibility in terms of how such periodic allocations are disbursed (*i.e.* not real-time). For instance, a fluid workload may specify a periodic need for resources as long as the disbursement of these cycles is guaranteed over some acceptable range of periods. For example, a fluid workload may specify the need for 10K cycles per second as long as these cycles are disbursed over a fixed period in the range between 100msec and 10 secs. Thus, a disbursement of 1K cycles every 100 msecs is acceptable as is a disbursement of 100K cycles every 10 secs. But, a disbursement of 200K cycles every 20 secs would be unacceptable as it violates the upper bound imposed on the allocation period, and so would an allocation of 100 cycles every 10 msecs as it violates the lower bound.

It is important to highlight that our periodic allocation is less stringent than what a “real-time” workload may require, but more stringent than what typical IaaS virtualization technologies are able to offer. Unlike real-time systems, there is no notion of deadlines, but rather an expectation of resource allocations at a prescribed predictable rate.

Definition 1. *An SLA τ is defined as a tuple of natural numbers (C, T, T_l, T_u, D, W), such that $0 < C \leq T$, $T_l \leq T \leq T_u$, $D \leq W$, and $W \geq 1$, where C denotes the resource capacity supplied or demanded during each allocation interval T, T_l and T_u are lower and upper bounds on T, and D is the maximum number of times that the workload could tolerate missing an allocation in a window consisting of W allocation intervals.*

According to the above definition, an SLA of type (C, T, T_l, T_u, D, W) represents a fluid workload which requires an allocation of C every interval T, where T can vary between T_l and T_u as long as the ratio C/T is consistent with the original SLA type. The following are illustrative examples, which are meant to show the range of SLAs that we are able to express using this generalized model.

An SLA of type $(1, 2, 2, 2, 0, 1)$ can be used to represent a workload that requires a unit capacity $C = 1$ over an allocation period $T = 2$ and cannot tolerate any missed allocations. An SLA of type $(1, 2, 2, 2, 1, 5)$ is similar in its periodic demand profile except that it is able to tolerate missed allocations as long as there are no more than $D = 1$ such misses in any window of $W = 5$ consecutive allocation periods.

An SLA of type $(2, 4, 2, 8, 0, 1)$ represents a fluid workload that demands a capacity $C = 2$ every allocation interval $T = 4$, however the original SLA would still be satisfied if its gets a capacity $C' = 4$ every allocation interval $T' = 8$ since the ratio C'/T' is equal to C/T.

As we noted earlier, without loss of generality, in this paper, we restrict our attention to workloads for which $D = 0$ and $W = 1$. Allowing different D and W would not change our MorphoSys framework, other than providing us with *even more* flexibility for colocation (due to availability of additional set of rewriting rules [13]), and hence better performance.

Fluid Transformations: In addition to the transformations implied by Theorems 1-4 from [13], we introduce the following tranformation for fluid workloads (Proof in [14]).

Theorem 5. *Given a fluid SLA type $\tau = (C, T, T_l, T_u)$, if $T_l \leq T' \leq T_u$ and $C' = \lceil C * T'/T \rceil$ then the SLA type $\tau' = (C', T', T_l, T_u)$ satisfies τ.*

IV. MorphoSys: The Framework

We consider an IaaS setting consisting of any number of homogeneous instances (servers), to which we refer as “Physical Machines” (PM).[4] Each workload (served with a virtual machine instance) is characterized by an SLA that follows the definition above – namely $\tau = (C, T, T_l, T_u, D, W)$. The MorphoSys colocation framework consists of two major services: a Workload Assignment Service (WAS) and a Workload Repacking Service (WRS). WAS assigns workloads to PMs in an on-line fashion using a prescribed assignment policy. WRS performs workload redistribution across PMs to optimize the use of cloud resources.

[4]Again, we emphasize that while we present our framework in the context of computational supply and demand – using terminologies such as physical and virtual machines – MorphoSys is equally applicable to other types of resources.

A. Workload Assignment Service (WAS)

Figure 1 provides an overview of the main elements of WAS. WAS is invoked upon the arrival of a request for a Virtual Machine (VM) allocation, in support of a workload specified by an SLA. The WAS service uses one of two heuristics to select the PM that could potentially host the VM: First Fit (FF) and Best Fit (BF). FF assigns the VM to the first PM that can satisfy the VM's SLA, whereas BF assigns the VM to the fullest – most utilized – PM that can still satisfy the VM SLA.

If it is not possible for WAS to identify (using FF or BF) a PM (currently in use) that could host the newly-arriving VM, then WAS attempts to rewrite the SLA of the VM (safely) in the hopes that it would be possible to assign the VM (subject to the transformed SLA) to an existing PM. To do so, WAS proceeds by generating a safe SLA transformation and attempts to use either FF or BF to find an assignment. This process is repeated until either one of the safe SLA transformations results in a successful assignment of the VM to a PM, or WAS runs out of possible safe SLA transformations. In the latter case, WAS may invoke the WRS repacking service to repack already utilized hosts in an attempt to assign the workload, or alternatively WAS can simply instantiate a new PM to host the newly-arriving PM.

In the worst case, the complexity of WAS is $O(k*n)$ where k is the largest number of possible task transformations, and n is the number of hosts in the system. Based on experimental observations (inferred by traces from real-workloads), $k << n$, which implies that in practice, WAS scales linearly with the number of hosts.

B. Workload Repacking Service (WRS)

Repacking is an essential service that allows the remapping/reclustering of workloads. This service is needed because IaaS environments may be highly dynamic due to the churn caused by arrival and departure of VMs, and/or the need of customers to change their own resource reservations. Over time, such churn will result in under-utilized hosts which could be managed more efficiently if workloads are repacked.

Repacking Heuristic: Remapping a set of workloads to multiple hosts efficiently is the crux of the problem. We say "efficient" as opposed to optimal because multi-processor real-time scheduling has been shown to be NP-Hard [15] (and our problem by reduction is also NP-Hard), and thus we resort to heuristics. In particular, we implemented repacking heuristic algorithms that utilize Breadth First Search (BFS) and Depth First Search (DFS) techniques to explore the solution space.

Given the set of task transformations, our (BFS or DFS) heuristic proceeds by setting up a search space of all the alternative task sets that could be considered for colocation. It then proceeds to explore the search space with the aim of finding a feasible colocation. In the worst case, our heuristic may end up searching the entire solution space, which is obviously impractical. To manage the exponential nature of the search space, our heuristic utilizes two strategies, which proved highly effective.

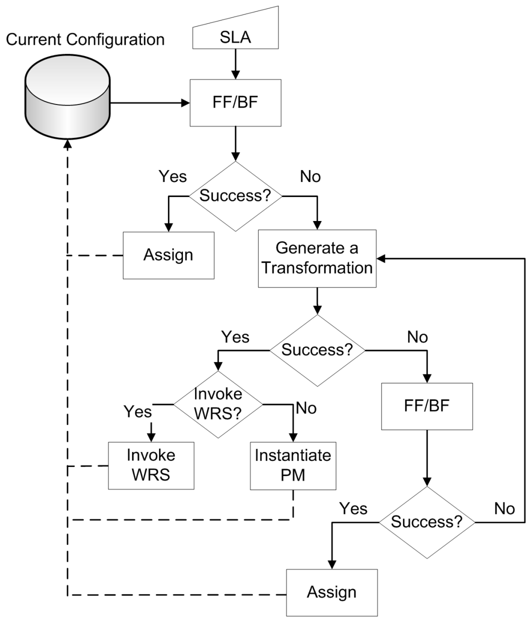

Fig. 1. The WAS Component of MorphoSys.

Our first strategy is based on an early-pruning approach: at each stage of our search, if the aggregate utilization of the workloads under consideration thus far is greater than the current capacity of the "best" solution found based on our adopted schedulability condition, then we prune that branch of the tree on the assumption that a feasible solution cannot exist down that path.[5]

Our second strategy is based on a smaller-degree-first-search approach: we build the search space by greedily starting with tasks that have the smallest number of transformations. This ensures that when pruning is applied, it will likely maximize the size of the pruned subspace. This optimization strategy was shown to be quite effective in reducing the solution search space for network embedding problems [16].[6]

Repacking Policies: Our WRS service could be instantiated based on one of three possible repacking policies: No Repacking (NR), Periodic Repacking (PR), and Forced Repacking (FR). NR is used to disable WRS, PR allows repacking to run at designated epochs/periods based on a system defined parameter. FR allows repacking to be applied in a "on line" fashion (triggered by the WAS).

Migration Policies: The effectiveness of the repacking policy depends on the ability to migrate workloads from one host to another. However, adding hosts increases the total number of workloads to be repacked which in turn results in an increase in the total service turnaround time. Thus we model three types of migration policies: No Migration (NM), Constrained Migration (CM), and Unconstrained Migration (UM). NM policy allows for repacking on the condition that workloads will not migrate from the host to which they are assigned.

[5]Initially, the "best" solution is set to the total number of hosts used prior to repacking.

[6]For practical purposes, we set an upper-bound on the execution time of the repacking heuristic, which we take to be 5 minutes for services operating on hourly "pay-as-you-go" reservations.

This approach will naturally consider *one* host at a time, and is suitable for running WAS in an "online" fashion. CM and UM policies allow for workloads to migrate from one host to another as long as it results in a more efficient repacking of these workloads. This is suitable when WAS is run in an "offline" fashion. The difference between CM and UM is in the host selection criteria: UM considers all system hosts, whereas CM considers hosts that satisfy a host selection condition.

Host Selection Condition: A host is a candidate for repacking if it satisfies a condition on its utilization. Let $0 < \phi < 1$ be the average host utilization, which we define as follows.

$$\phi = \frac{\sum_{i=1}^{n} u_i}{n}$$

where $u_i > 0$ is the utilization of host i. Furthermore, let $\omega = \sum_{i=1}^{k} C_i/T_i$ be the sum of the utilizations of the workloads on a specific host (based on the original workload SLA and not the transformed workload SLA). A host is a candidate for repacking if $\phi - \omega \geq \epsilon$, where $0 < \epsilon \leq 1$ is a tunable parameter (CM reduces to UM, when $\epsilon = 0$).

V. MORPHOSYS: EVALUATION

In this section we present results from extensive experimental evaluations of the MORPHOSYS framework. Our main purpose in doing so is to establish the feasibility of our proposed service by: (1) comparing the schedulability of QoS workloads, with and without applying our safe SLA transformations, (2) evaluating the effect from using different migration policies on the efficiency of colocation, (3) evaluating the effect of changes in the mix of fluid and non-fluid workloads, and (4) evaluating the effect of changing the flexibility of fluid workloads on the efficiency of colocation.

Simulation Setup: Our setting is that of a cloud storage service used to host a large number of streaming servers. This setting is general enough to represent different forms of applications, such as a cloud content provider streaming and other multimedia services. Typically for such applications, the disk I/O constitues the bottleneck of the overall system performance [17]. The maximum throughput delivered by a disk depends on the number of concurrent streams it can support without any SLA violation.

To drive our simulations, we utilize a collection of video traces from [18]. We assume that the underlying system of the provider is a disk I/O system that serves requests of different streaming servers using a fixed priority scheduling algorithm, which we take to be Rate Monotonic. The usage of fixed priority algorithms for disk scheduling was suggested by Daigle and Strosnider [19], and Molano et al [20].

The video traces [18] provide information about the frames of a large collection of video streams under a wide range of encoder configurations like H.264 and MPEG-4. We conducted our experiments with a subset of 30 streams, with HD quality, and a total duration of one hour each. We initially identify the period for serving a video stream request as the period of the `I` frames (*a.k.a.*, Group of Pictures, or GoPs). Overall, there were three unique periods in our collection of video traces.

We model the SLA associated with each stream as follows: The SLA specifies a periodic (disk I/O) demand C over a periodic allocation time T. For a given stream, the periodic demand C is set as follows:

$$C = \frac{max(\sum_{i=0}^{n-1} b_i)}{\theta * T}$$

where b_i is the volume in bytes of the stream in interval $[i * \theta * T, (i+1) * \theta * T]$.

The allocation period T is set to be equal to $\theta * T'$, where T' is one of the three unique periods in our video traces and θ ($\theta \geq 1$) models the tolerance of the client (the recipient of the stream) to burstiness in the allocation over time. In particular, for any given value of θ, it is assumed that the client is able to buffer (and hence absorb) up to $T = \theta * T'$ seconds of the stream (*i.e.*, θ GoPs). A large value for θ implies that the allocation is over a large number of GoPs, and hence a tolerance by the client for a bursty disbursement of periodic allocation. A small value for θ specifies a smoother disbursement over time. Each client request specifies a value for the parameter θ which is chosen at random between a lower bound β and an upper bound γ. In our experiments we set $\beta = 1$ and $\gamma = 10$. To model the level of fluidity (flexibility) of an SLA, we allow the period T to range from $(\theta - \sigma) * T'$ to $(\theta + \sigma) * T'$, where σ ($\theta \geq \sigma \geq 0$) determines the allowable deviation from the nominal allocation period. A non-fluid SLA is one where $\sigma = 0$.

We model churn in the system as follows. Client arrivals (requests for streams) are Poisson (independent) with a rate λ. Poisson arrival processes for VoD have been observed in a number of earlier studies (*e.g.*, [21]). A client's session time is set to be the length of the entire stream served to the client. The specific stream requested by the client is chosen uniformly at random from among all streams in the system. Experiments with skewed distributions (*e.g.*, Zipf) resulted in results that are similar to those obtained using exponential preference,[7] and thus are not reported.

In our experiments, our purpose is to evaluate the efficiency of computing a colocation configuration for our workloads, as opposed to the performance of system deployment. Thus, to measure the efficiency of a colocation strategy X, we report the *Colocation Efficiency* (CE), which is defined as follows:

$$CE = 1 - \frac{W(X)}{W(FF)}$$

where $W(X)$ is the amount of wasted (unallocated) resources when colocation strategy X is used, and $W(FF)$ is the measure of wasted resources when our baseline First-Fit (FF) strategy is used. Thus, CE can be seen as the degree to which a strategy is superior to FF (the reduction in wasted resources relative to FF, which according to theoretical bounds [10] , can be up to 30%). In our experiments, all CE values are calculated with 95% confidence.

Relative Performance of Various Strategies: Recall that the assignment of an incoming workload (request) is done

[7]This is expected given the relatively similar lengths of the streams in the trace.

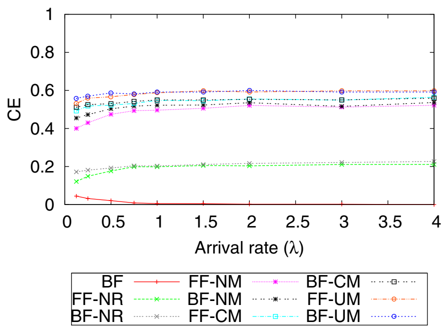

Fig. 2. Colocation Efficiency: Baseline Results

using WAS, which attempts various SLA transformations on an incoming request until the potentially transformed request is possible to assign to a host (disk) using either First-Fit or Best-Fit.

In a first set of experiments, we compared the performance of WAS with No Repacking under both FF and BF (namely FF-NR and BF-NR) to that of the plain FF and BF heuristics (*i.e.*, without attempting any SLA transformations). Figure 2 shows the results we obtained when varying the arrival rate (λ) for the different packing strategies: BF, FF-NR, and BF-NR.

In general, the performance of BF is only marginally better that FF, whereas both FF-NR and BF-NR show measurable (up to 20%) improvement over both FF and BF, with BF-NR performing slightly better than FF-NR. These results suggest that there is a measurable improvement in colocation efficiency even when minimal SLA transformations are allowed (namely the transformation of the SLA of the incoming request only).

To evaluate the benefit from repacking and migration, we ran a similar experiment with the repacking policy set to Forced Repacking (FR). Figure 2 shows the measured CE values for different arrival rates (λ) and different repacking strategies. Our "online" repacking strategies with no migration – namely FF-NM and BF-NM – improved colocation efficiency significantly. For lower arrival rates, CE was around 0.4, implying a 40% reduction in wasted (unallocated) resources compared to FF. For moderate and higher arrival rates, the reduction is more pronounced around 50%.

Figure 2 also shows results of experiments in which various migration policies are enabled – namely Constrained Migration (CM) and Unconstrained Migration (UM). Both approaches result in better performance compared to NM approaches, yielding CE values between 0.55 and 0.6.

To summarize, this initial set of experiments suggests that, even in the absence of any fluidity in the workload (SLA flexibility), a reduction of up to 60% in the wasted resources is to be expected through the use of SLA transformations and repacking.

Benefit from Fluid Transformations: To measure the effect of fluidity on the overall colocation efficiency, we performed experiments using the same setting as before, while allowing a certain percentage of the requests to be fluid (with $\sigma = 1$), and only allowing fluid transformations to be applied. In other words, non-fluid workloads were not subjected to any transformations.

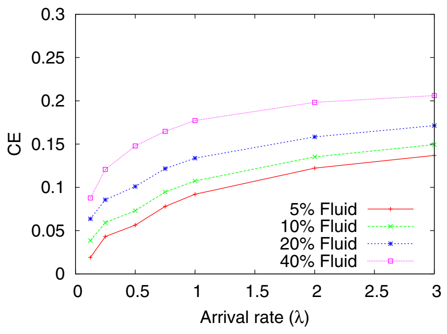

Fig. 3. Effect of fluid SLAs when only fluid transformations are allowed, $\sigma = 1$, FF-NR.

Figures 3 shows the result obtained using FF-NR,[8] for various mixes of fluid and non-fluid workloads. As the result suggests, having a mix with even a small percentage of fluid workloads results in improvements (up to 20%) that are comparable to what we obtained when transformation of non-fluid workloads was allowed with no repacking (cf. Figure 2).

In the previous experiment, we fixed the fluidity level ($\sigma = 1$) and studied the effect of changes in the mix of fluid versus non-fluid SLAs. Figure 4 shows result of additional experiments in which we changed the level of fluidity (the parameter σ) while keeping the value of $\lambda = 1$, for various mixes of fluid and non-fluid workloads. The results show that only small levels of flexibility ($\sigma < 2$) provided most of the achievable improvements when only fluid transformations are considered.

Combined Benefit from Fluid and non-Fluid Transformations: Fixing the level of fluidity to a small value ($\sigma = 1$), Figures 5 shows result from experiments in which all possible transformations are allowed in a No-Repacking setting (*i.e.*, FF-NR) for different workload mixes. The results show that the resulting performance is marginally better (by only a few percentage points) than applying *either* non-fluid transformations *or* fluid transformations.

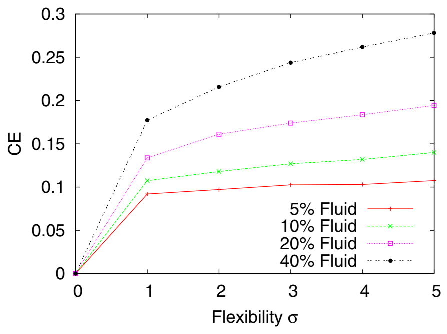

Fig. 4. Effect of fluidity level when only fluid transformations are allowed, $\lambda = 1$, FF-NR.

System Scalability: In our experiments, the WAS component of MORPHOSYS was able to handle large clusters of resources (disks) – up to $4,000$. If migration of workloads is not enabled, WRS is able to handle even larger clusters in an "online" fashion. Enabling migration introduces significant

[8]BF-NR results (found in [14]) exhibit similar patterns to that of FF-NR, and are not reported due to space constraints.

computational overheads when dealing with large clusters. This makes the use of migration in WAS more practical for "off-line" (batch) use.

In a typical IaaS setting, there might be even more than a few thousand resources under management, which would be more than what a single WAS instance can handle. We note that in such cases, a practical solution would be to group the resources under management into separate clusters, each of which is managed by a single WAS.

We note that our measurement of scalability deals only with the computational aspect of MORPHOSYS (namely, computing efficient colocation configuration). In actual deployments, scalability will also depend on additional considerations due to system overheads that are dependent on the specific setting. For IaaS settings like the one considered in the experiments we presented in this paper (colocation of streaming servers), one would not expect much reconfiguration overheads except that of migrating and merging user stream requests.[9] However, in other settings involving more significant overheads (*e.g.*, the handling of large memory VM images to allow VM migration across hosts), the scalability of MORPHOSYS will depend on the efficient management of such aspects.

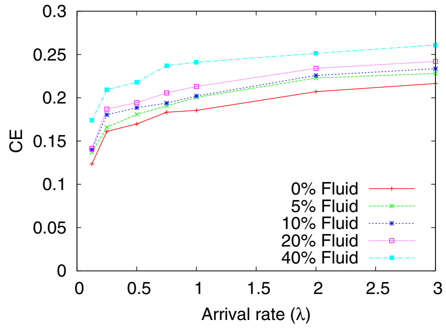

Fig. 5. Combined benefit from applying both fluid and non-fluid transformations, FF-NR.

VI. RELATED WORK

Service Level Agreements (SLAs): There has been a significant amount of research on various topics related to SLAs. The usage of resource management in grids have been considered in [24], [25]; issues related to specification of SLAs have been considered in [26]; and topics related to the economic aspects of SLAs usage for service provisioning through negotiation between consumers and providers are considered in [27], [28]. A common characteristic (and/or inherent assumption) in the above-referenced body of prior work is that the customer's SLAs are immutable. We break that assumption by recognizing the fact that there could be multiple, yet functionally equivalent ways of expressing and honoring SLAs. Our MORPHOSYS framework utilizes this degree of freedom to achieve significantly better colocation.

VM Colocation: VM consolidation and colocation are very active research topics (*e.g.*, the work by Jason *et al* [29]) that aim to minimize the operating cost of data centers in terms of hardware, energy, and cooling, as well as providing a potential benefit in terms of achieving higher performance at no additional cost. Much work has gone into studying the consolidation of workloads across various resources: CPU, memory, and network [3]–[5], [30], [31]. We note that in all these works, the specification of the resource requirements for a VM is static and based on some fixed average requested capacities. In our work, the specification of resource needs is much more expressive as it allows VMs to control their resource allocation time-scale, as well as expose any flexibilities VMs may have regarding such timescale.

Real-Time Scheduling: Different scheduling algorithms were suggested to deal with scheduling of periodic/aperiodic hard real-time and soft-real time tasks [32] (and the references within). In addition, variants of proportional-share scheduling algorithms – based on the concept of Generalized Processor Sharing (GPS) have been suggested [11] – which allow the integration of different classes of applications. These approaches however do not take into consideration reservation of resources and fairness in allocating resources. The work by Buttazzo et al [33] present an elastic task model based on a task defined using a tuple $(C, T, T_{min}, T_{max}, e)$, where T is the period that the task requires, T_{min} and T_{max} define the maximum and minimum periods that a task can accept. Our SLA model allows us to express classes of applications that are more general than the elastic task model. Moreover, the SLA transformations that we utilize allow us to serve workloads under completely different (C, T) server supplied resources.

Resource Allocation in Distributed Settings: Different approaches have been suggested to deal with resource allocation in distributed settings [25], [34]–[36] among many others. In these works, the main mechanisms used for providing QoS guarantees to users are through resource reservations. Such reservations can be immediate, undertaken in advance [34], or flexible [25]. To achieve efficient allocation and increased resource utilization, these approaches model workloads as having a start time and end time. Under such approaches the resources allocated to a workload would still be based on a percentage reservation, which results in performance variability specifically for periodic workload requests. Our work complements these models by allowing for an expressive SLA model that admits the specification of constraint flexibilities. We believe that providing this capability is crucial for the deployment of QoS-constrained workloads while at the same time ensuring efficient utilization of resources. This is the case, especially when such capabilities are coupled with the possibility of safely transforming the workload characteristics.

VII. CONCLUSION

In this paper, we proposed a new SLA model for managing QoS-constrained workloads in IaaS settings. Our SLA model supports an expressive specification of the requirements for various classes of applications, thus facilitating auditability and performance predictability using simple measurement techniques. We presented the architectural and al-

[9]The effect of such overheads can be reduced using a number of ways including manipulation of stream playout rates or by inserting secondary content in the stream [22] (e.g. as done in systems like Crackle [23]).

gorithmic blueprints of a framework for the deployment of dynamic colocation services.MorphoSys utilizes workload SLA transformations (exploiting any flexibility therein) for efficient management of QoS-constrained workloads in the cloud. We evaluated our framework by considering a cloud storage service scenario, and performed extensive evaluation using real video traces. The results reported in this paper – which suggest significant reduction in unallocated (wasted) resources of up to 60 percent – underscore the potential from deploying MorphoSys-based services.

Our on-going research work is pursued along three dimensions. Along the first, we are investigating extensions to our SLA model to allow for yet more expressive forms of SLAs – *e.g.*, allowing the specification of additional constraints such as geographic location and group temporal colocation. Our second line of work is concerned with the development of pricing models that provide incentives to customers of a MorphoSys-enabled cloud service to declare any flexibilities in their workload SLAs. Our third line of work is focused on developing a prototype of a MorphoSys-based service that will allow us to measure the performance of MorphoSys, not only in a dynamic setting that is subject to churn (as we have done in this paper), but also in a setting that is subject to overheads resulting from actual relocation of workloads. Elements of this prototype have been developed as part of our work on XCS – a VM cloud colocation service [30].

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and The Art of Virtualization," in *SOSP '03*, New York, NY, USA, 2003.

[2] A. Haeberlen, "A case for the accountable cloud," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 52–57, April 2010.

[3] R. Nathuji and K. Schwan, "VPM Tokens: Virtual Machine-Aware Power Budgeting in Datacenters," in *HPDC*, New York, NY, 2008.

[4] M. Cardosa, M. Korupolu, and A. Singh, "Shares and Utilities Based Power Consolidation in Virtualized Server Environments," in *Proc. of IFIP/IEEE Integrated Network Management*, 2009.

[5] V. P. Xiaoqiao Meng and L. Zhang, "Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement," *INFOCOM*, March 2010.

[6] S. Craciunas, A. Haas, C. Kirsch, H. Payer, H. Röck, A. Rottmann, A. Sokolova, R. Trummer, J. Love, and R. Sengupta, "Information-Acquisition-as-a-service for Cyber-physical Cloud Computing," in *Hot-Cloud'10*. USENIX Association, 2010, p. 14.

[7] J. Napper and P. Bientinesi, "Can Cloud Computing Reach The Top 500?" in *UCHPC-MAW '09: Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*. New York, NY, USA: ACM, 2009, pp. 17–20.

[8] D. Patnaik, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Performance Implications of Hosting Enterprise Telephony Applications on Virtualized Multi-core Platforms," in *IPTComm '09*, New York, NY, USA, 2009.

[9] "Single-chip Cloud Computer," http://techresearch.intel.com/articles/Tera-Scale/1826.htm, May, 2010.

[10] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, no. 1, 1973.

[11] K. J. Duda and D. R. Cheriton, "Borrowed-virtual-time (BVT) Scheduling: Supporting Latency-sensitive Threads in a General-purpose Scheduler," *SIGOPS Oper. Syst. Rev.*, vol. 33, no. 5, pp. 261–276, 1999.

[12] J. P. Lehoczky, L. Sha, and Y. Ding, "Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," in *Proc. of the 11th IEEE Real-time Systems Symposium*, Dec. 1989.

[13] V. Ishakian, A. Bestavros, and A. Kfoury, "A Type-Theoretic Framework for Efficient and Safe Colocation of Periodic Real-Time Systems," *IEEE RTCSA*, pp. 143–152, 2010.

[14] A. B. Vatche Ishakian, "MorphoSys: Efficient Colocation of Qos-Constrained Workloads in the Cloud," CS Department, Boston University, Tech. Rep. BUCS-TR-2010-002, January 2011.

[15] M. Garey and D. Johnson, *Computers and Intractability. A Guide to The Theory of NP-completeness. A Series of Books in the Mathematical Sciences*. San Francisco, 1979.

[16] J. Londoño and A. Bestavros, "NETEMBED: A network resource mapping service for distributed applications," in *IPDPS 2008.*, April 2008.

[17] L. Cherkasova and L. Staley, "Building a performance model of streaming media applications in utility data center environment," in *CCGrid '03*, 2003.

[18] G. Van der Auwera, P. David, and M. Reisslein, "Traffic and Quality Characterization of Single-Layer Video Streams Encoded with the H.264/MPEG-4 Advanced Video Coding Standard and Scalable Video Coding Extension," *IEEE Transactions on Broadcasting*, vol. 54, no. 3, pp. 698 –718, 2008.

[19] S. Daigle and J. Strosnider, "Disk Scheduling for Multimedia Data Streams," in *Proceedings of the IS&T/SPIE*, 1994.

[20] A. Molano, K. Juvva, and R. Rajkumar, "Real-time Filesystems. Guaranteeing Timing Constraints for Disk Accesses in RT-Mach," in *RTSS*. IEEE, 2002, pp. 155–165.

[21] E. Veloso, V. Almeida, W. Meira, A. Bestavros, and S. Jin, "A Hierarchical Characterization of a Live Streaming Media Workload," *IEEE/ACM Transactions on Networking*, vol. 14, no. 1, February 2006.

[22] P. Basu, A. Narayanan, W. Ke, and T. Little, "Optimal scheduling of secondary content for aggregation in video-on-demand systems," in *ICCCN*. IEEE, 1999, pp. 104–109.

[23] Crackle, http://www.crackle.com/.

[24] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, vol. 2537, 2002, pp. 153–183.

[25] M. A. Netto, K. Bubendorfer, and R. Buyya, "SLA-Based Advance Reservations with Flexible and Adaptive Time QoS Parameters," in *Proceedings of the 5th international conference on Service-Oriented Computing*. Berlin, Heidelberg: Springer-Verlag, 2007.

[26] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *J. Netw. Syst. Manage.*, vol. 11, pp. 57–81, March 2003.

[27] A. Barmouta and R. Buyya, in *GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration*, ser. IPDPS '03, Washington, DC, USA, 2003.

[28] L.-O. Burchard, M. Hovestadt, O. Kao, A. Keller, and B. Linnert, "The Virtual Resource Manager: An Architecture For SLA-aware Resource Management," in *CCGRID '04*, Washington, DC, 2004.

[29] J. Sonnek and A. Chandra, "Virtual Putty: Reshaping the Physical Footprint of Virtual Machines," in *HotCloud'09*. USENIX, 2009.

[30] V. Ishakian, R. Sweha, J. Londoño, and A. Bestavros, "Colocation as a Service. Strategic and Operational Services for Cloud Colocation," in *IEEE NCA*, Boston, USA, 2010.

[31] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner, "Memory Buddies: Exploiting Page Sharing for Smart Colocation in Virtualized Data Centers," in *VEE*, Washington, DC, 2009.

[32] R. Davis and A. Burns, "A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems," University of York, Department of Computer Science, techreport YCS-2009-443, 2009.

[33] G. C. Buttazzo, G. Lipari, and L. Abeni, "Elastic Task Model for Adaptive Rate Control," in *RTSS '98*, Washington, DC, USA, 1998.

[34] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy, "A Distributed Resource Management Architecture That Supports Advance Reservations and Co-allocation," in *IWQoS'99*, 2002.

[35] C. Castillo, G. N. Rouskas, and K. Harfoush, "Resource Co-allocation For Large-scale Distributed Environments," in *HPDC '09*, New York, NY, USA, 2009.

[36] B. Lin and P. Dinda, "Vsched: Mixing Batch and Interactive Virtual Machines Using Periodic Real-time Scheduling," *Proceedings of ACM/IEEE SC (Supercomputing)*, 2005.