Proactive Memory Scaling of Virtualized Applications

Simon Spinner, Nikolas Herbst and Samuel Kounev
University of Würzburg
Würzburg, Germany
Email: {firstname.lastname}@uni-wuerzburg.de

Xiaoyun Zhu, Lei Lu, Mustafa Uysal and Rean Griffith
VMware, Inc.
Palo Alto, USA
Email: {xzhu, llei, muysal, rean}@vmware.com

***Abstract*—Enterprise applications in virtualized environments are often subject to time-varying workloads with multiple seasonal patterns and trends. In order to ensure quality of service for such applications while avoiding over-provisioning, resources need to be dynamically adapted to accommodate the current workload demands. Many memory-intensive applications are not suitable for the traditional horizontal scaling approach often used for runtime performance management, as it relies on complex and expensive state replication. On the other hand, vertical scaling of memory often requires a restart of the application. In this paper, we propose a proactive approach to memory scaling for virtualized applications. It uses statistical forecasting to predict the future workload and reconfigure the memory size of the virtual machine of an application automatically. To this end, we propose an extended forecasting technique that leverages meta-knowledge, such as calendar information, to improve the forecast accuracy. In addition, we develop an application controller to adjust settings associated with application memory management during memory reconfiguration. Our evaluation using real-world traces shows that the forecast accuracy quantified with the MASE error metric can be improved by $11 - 59\%$. Furthermore, we demonstrate that the proactive approach can reduce the impact of reconfiguration on application availability by over 80% and significantly improve performance relative to a reactive controller.**

I. Introduction

Enterprise applications often have workloads that change over time, with overlapping seasonal patterns and trends. At the same time, virtualization offers great flexibility in dynamically changing the amount of resources allocated to running applications. In the past decade, this flexibility has proven to be very valuable: instead of allocating resources based on the peak demands of an application, we can dynamically match the amount of allocated resources with the actual resource demands of the application at runtime. This results in higher utilization of physical resources without sacrificing the application's quality of service.

There are a number of mechanisms one can use to change the resource allocation of an application using virtualization. At the most basic level, the scheduling parameters of a virtual machine (VM) running in the hypervisor can be changed to affect the VM's CPU and memory consumption, or the IO bandwidth. Another method is to dynamically migrate a VM from one physical host to another so that it can have access to potentially larger amounts of resources on the latter. A third approach is to spawn more VM instances for a bottlenecked application tier, an approach commonly referred to as *horizontal scaling* of an application. In most recent years, hypervisors added the capability to add (or remove) virtual resources, such as virtual CPUs (vCPUs), memory, or I/O devices to running VMs. This is referred to as hot-add (or hot-remove). This way, for example, one can reconfigure a VM from a 2-vCPU, 8 GB memory configuration to an 8-vCPU, 32 GB memory configuration without restarting it. This is referred to as *vertical scaling* of a VM.

Vertical scaling provides a running VM immediate access to more resources (bigger memory, more CPU instances). However, many applications are unable to immediately start consuming the newly available resources (e.g., memory); in some cases, the guest operating system (OS) or the application itself needs to be restarted. On the other hand, we cannot just allocate the maximum amount of resources to a VM that it may require at peak times, as this would create huge overheads in terms of scheduling overheads and larger page tables or it may lead to serious resource over-provisioning. As a result, it is important for the resource allocation systems to determine the right configuration for the VMs, and to reconfigure the VM resources prior to the incoming demand peaks.

Unfortunately, existing resource management systems do not have the capability to proactively reconfigure application resources to satisfy their projected resource demands in the future. Existing systems are *reactive* in nature, changing resources after a need for a change in allocation is already observed, typically causing performance degradation in the application. For vertical scaling, this approach could be detrimental, causing additional slowdowns during reconfiguration while the resource demands are rising.

In this paper, we introduce a new proactive approach for dynamically configuring virtual resources of an application, using a combination of demand forecasting and resource prediction techniques. Demand forecasting creates a model of application demands over time allowing to make the resource allocation decisions ahead of changes in the workload demand. Resource prediction enables the determination of the right resource configurations that would be able to satisfy the projected demand. Our demand forecasting method relies on multiple time series analysis techniques and incorporates additional meta-knowledge (e.g., calendar information) in the models to better capture the seasonality patterns in the workloads. In the evaluation, we focus on main memory as the primary resource, as memory reconfigurations are the most costly operations and it is extremely critical for most applications to be provisioned with sufficient memory resources.

We evaluate our forecasting method on three different request traces from real-world applications. The results show that incorporating calendar information in the forecast model significantly improves its accuracy compared to state-of-the-art statistical forecast methods. The MASE error metric is improved between 11% and 59% for all three traces. Using the forecast method as a foundation, we implemented a proactive resource controller to adapt the memory size of a VM according to the predicted workload demand. In a case study, we applied the proactive controller to the Zimbra Collaboration Server [1] subject to a trace-driven, time-varying workload. The proactive controller reduces the impact of the reconfiguration on the application availability by more than 80% and significantly improves application latency compared to a threshold-based, reactive controller. Furthermore, we show that using our extended forecast method the over- and under-provisioning of the VM's memory is below 11%.

II. Motivation and Challenges

Modern hypervisors (e.g., VMware ESX) support two different approaches to dynamically adapt the actual amount of memory available to a virtualized application: memory hot-add (and hot-remove), or memory ballooning. Memory ballooning techniques [2] allow to reclaim memory from the guest OS at run-time. This mechanism is used to reallocate the physical memory between VMs in over-commited scenarios. However, memory-ballooning only works within the bounds of the initially configured memory size. While it would be possible to set this size to the maximum physical memory size, it is not recommended as it usually results in memory overheads. Furthermore, it can also severely limit the possibilities to migrate such a VM between physical hosts in a heterogeneous data-center. In the following, we describe the challenges of memory scaling caused by deficiencies of many modern operating systems and applications.

Most current versions of the Linux and Windows OS can dynamically activate additional memory without a restart. However, memory reconfigurations can fail at the OS level. For instance, Linux requires a contiguous physical memory space for its internal memory management tables. If there is a high memory pressure by the application, we observed frequent failures when activating the additional memory as the OS is unable to find enough space to enlarge its memory management tables.

Many enterprise applications, including their underlying middleware and database systems, implement their custom memory management mechanisms. Examples include database systems, such as the MySQL server, which maintains a buffer cache to keep frequently used pages in memory, and process VMs, such as the Java VM, with their garbage collected heap space. These applications also need to be made aware of any additional memory added at runtime so that they can adapt their behavior accordingly. However, the parameters controlling the application's memory management mechanisms often cannot be changed without restarting the application (e.g., the MySQL buffer pool size, or the maximum heap size of a Java application).

In [3], the authors propose an extension to MySQL and OpenJDK to integrate memory ballooning techniques in the application memory management. However, this approach works only within certain bounds configured initially and it requires profound changes in the OS and application source code [3]. As long as the application does not allow to dynamically changes its memory management configuration, the only option is to restart the application although it is an expensive operation, both in terms of availability and performance. In particular, data-intensive applications need to reload their working set data back into memory after a restart. The Zimbra Collaboration Server [1], which is an example for such a data-intensive application, becomes unavailable for up to 10 minutes due to the restart and shows a severely degraded application performance (a slowdown factor of 10) for over half an hour while reloading its internal caches (see Section V-A).

In summary, the major challenges when reconfiguring memory of virtualized applications are the following: a) the settings of the memory management mechanisms of applications must be adjusted to reflect the additional capacity, b) many practical applications need to be restarted to update the memory management configuration, and c) the reconfiguration is unreliable when memory demand is high and may cause additional overheads.

III. Approach

Our approach is based on a control loop which proactively adds or removes memory resources to VMs match its future workload demand and to improve application availability and performance. We use a proactive approach, as it enables us to plan the reconfiguration in advance and schedule it to be executed during a phase of low application load (e.g., at night). This has the following benefits: a) reconfiguration failures at the OS level are avoided, and b) if an application restart is required, the impact on the performance and availability can be significantly reduced (see Section V).

We assume that the application is subject to a dynamic, time-varying workload with different seasonal patterns, such as daily and weekly patterns, as well as different long-term trends (increasing or decreasing). If a application restart is required, the memory reconfiguration may take place during pre-defined maintenance windows when a short application outage is acceptable (e.g., between 3:00 and 3:15 AM). This results in significantly longer control periods compared to many other runtime management systems. While the long control period limits the elasticity of the system, it helps to avoid counter-productive reconfigurations at peak workload.

The control loop consists of the components shown in Figure 1. For each virtualized application, our approach periodically adjusts the memory allocations to the VMs using a

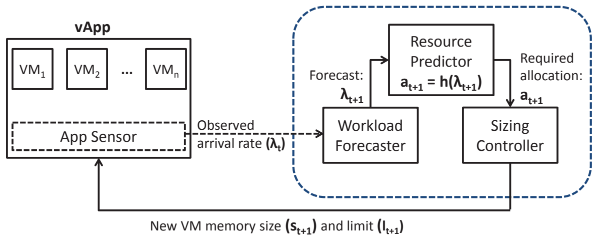

vApp
VM_1
VM_2
...
VM_n
App Sensor
Observed arrival rate (λ_t)
Forecast: λ_{t+1}
Resource Predictor
$a_{t+1} = h(\lambda_{t+1})$
Required allocation: a_{t+1}
Workload Forecaster
Sizing Controller
New VM memory size (s_{t+1}) and limit (l_{t+1})

Fig. 1. Approach overview.

closed-loop controller. As described in Section II, a memory reconfiguration may also require changing the memory settings of the application, including a possible restart of the application. Our approach therefore needs to determine a memory allocation a_{t+1} sufficient to serve the peak workload during the next control interval $t+1$ (e.g., next 24 hours) and then it reconfigures the system in advance during a maintenance window. The *App Sensor* continuously monitors the arriving workload and stores the aggregate observations in a time series $\boldsymbol{\lambda}_t = \{\lambda_t^1 ... \lambda_t^m\}$. The parameter m controls the smoothing of the input data and also the computational overhead of the approach. Assuming a control interval of one day, we usually use $m = 48$ or $m = 24$ corresponding to a sampling interval of 30 to 60 minutes.

In order to predict the workload for the next interval, the time series $\boldsymbol{\lambda}_t$ is fed as input to the *Workload Forecaster* component. This component builds a statistical model based on the historical data $\boldsymbol{\lambda}_1 ... \boldsymbol{\lambda}_t$. Our workload forecaster relies on multiple time series analysis techniques and exploits additional calendar information to distinguish between different types of seasonal patterns (e.g., work days vs. non-working days). We refer to this approach as *splitTs* in the rest of the paper. The result of the workload forecaster is the expected peak workload λ_{t+1} of the control interval $t+1$.

The *Resource Predictor* component determines the required memory allocation a_{t+1} to a VM to be sufficient for the peak workload λ_{t+1}. To this end, it requires a function $a_{t+1} = h(\lambda_{t+1})$ that maps a given workload λ_{t+1} to a memory requirement a_{t+1} for the control interval $t+1$. In the following, we assume a step function mapping between a discrete set of memory sizes (e.g., in 2 GB steps) and the corresponding sustainable application workload. However, more generic functions can be used depending on the application. The function may be either provided to the resource predictor as input knowledge or learned over time by analyzing historic application performance data.

Based on the required memory allocation a_{t+1}, the *Sizing Controller* component determines the new VM memory settings for the application. This includes the configured memory size (s_{t+1}) and the memory limit (l_{t+1}) of the VM. The limit setting is used to reduce the memory consumption of a VM using memory-ballooning techniques. This is because hot-removal of memory is currently not supported by guest operating systems without a restart of the VM. Therefore, the sizing controller uses the limit setting to scale down the memory of a VM if it does not require all configured memory. The freed memory can then be used by other co-located VMs enabling a higher consolidation ratio on the physical host.

1) Workload Forecaster: Based on the observed arrival rates $\boldsymbol{\lambda}_t$ the workload forecaster predicts the expected arriving workload during the next control interval. After observing several days (typically three complete days), it is possible to forecast the arriving workload for a complete day using time series analysis methods as described in [4]. To avoid under-/over-provisioning of memory resources, it is crucial to predict the peak workload of the next day accurately.

A shortcoming of all forecasting methods based on time series analysis is their limited capability to identify and cope with multiple overlapping seasonal patterns at the same time. The effects of days, weeks and months are examples of such overlapping patterns as commonly found in real-world workload traces. Many workload traces from public webservers (e.g., in [5], [6]), or enterprise systems (e.g.,in [7]) show regular differences in the workload intensities between different week days (e.g., working vs. non-working days).

In order to cope with these types of overlapping seasonal patterns, splitTs classifies the observed, historic data $\boldsymbol{\lambda}_1 ... \boldsymbol{\lambda}_t$ into subsets $S_d = \{\boldsymbol{\lambda_i} : f(\boldsymbol{\lambda_i}) = d, 1 \leq i \leq t\}$ with $d \in D$. The classificator f is based on calendar information provided as static meta-knowledge. When predicting the arrival rate $\boldsymbol{\lambda}_{t+1}$, first $d = f(\boldsymbol{\lambda}_{t+1})$ is determined, and then only subset S_d is used for the forecast. In our evaluation, we use a fixed $D = \{workingday, nonworkingday\}$. More complex classifications are possible, however, it comes at the cost of an increased number of days required to learn the forecast model. As shown in Section IV, this classification significantly improves the forecasting accuracy on the considered real-world traces. In future work, we plan to extend splitTs to automatically cluster the different days based on the historic data.

In order to obtain a forecast from a subset S_d, we use the WCF method described in [7]. WCF dynamically selects between different underlying statistical methods based on time series analysis depending on the forecasting objectives and incorporates direct feedback on the forecast accuracy.

2) Sizing Controller: The sizing controller expects the required memory allocation a_{t+1} for the next control interval as input. It first determines the new memory size (s_{t+1}) and memory limit (l_{t+1}) of the VM considering the current memory size and technical constraints from the hypervisor (e.g., VMware ESX expects the memory size to be a multiple of 128 MB). If a_{t+1} is larger than the VM's current memory size, the hot-add functionality of the hypervisor is used to add additional memory to the VM without restarting it. Furthermore, the memory limit is also set to the required memory size. If instead, a_{t+1} is less than its current memory size, only the memory limit is adjusted accordingly.

The individual steps to adjust the memory settings of a VM also include the necessary adjustments to the OS and application configurations. The following steps are executed in the given order:

1) The application is stopped in order to adjust static configuration settings (e.g., Java maximum heap space, or the

database buffer pool size).

2) The new memory settings of the VM are fed to the hypervisor using the supported reconfiguration API.
3) Any additional memory is activated in the OS to become available to the memory scheduler. Depending on the OS, and its version this step is either triggered automatically by the OS or needs to be executed manually.
4) Any application-specific, memory-related configuration settings are updated to reflect the new memory size. The required automation scripts need to be provided by a system administrator and included in the VM.
5) The application is started with the new settings and it resumes serving user requests.

Two observations are worth noting. First, if memory usage is high, step 3) may fail if the OS cannot find enough free, contiguous physical memory to extend its memory management tables. This is why step 1) is executed before step 3) to reduce the memory usage in the VM and prevent reconfiguration failures. Second, steps 1) and 5) are not needed if the application is able to dynamically adapt its memory management to the new memory size of the VM.

IV. Forecast Accuracy

In this Section, we evaluate the splitTs forecaster with regards to the improvement in accuracy through exploiting calender meta-knowledge.

1) Workload Traces: In order to evaluate the prediction accuracy, request arrival traces are needed that fulfill the following requirements: (a) they should contain request arrivals from a real-world application, (b) they should include daily patterns, as this is an assumption of our approach, (c) they should cover a period of several weeks (at least four weeks) in order to learn a forecast model that also captures weekly patterns. We found three request arrival traces fulfilling our requirements: *FIFA'98 World Cup*, *Wikipedia*, and *CICS transactions*. The FIFA'98 traces [5] were taken from the web servers of the official FIFA'98 world cup web site. We used the first five weeks of the traces for our evaluation. The wikipedia traces [6] contain every 10th request to the official wikipedia site. For our evaluation, we used four weeks of trace data from the English language site (September 19th to October 21st, 2007). The CICS transaction time series reports the number of started transactions at a real-world deployment of an IBM z10 mainframe server. The trace data was taken from the case study described in [7]. We used a total of four weeks of this trace (January 31st to February 27th, 2011).

2) Error Metrics: We use the Mean Absolute Scaled Error (MASE) to evaluate the accuracy of the forecast values. Given a time series $Y_1...Y_n$ of actual observations from the system and a time series of forecasts $F_1...F_n$, the forecast error e_t is generally defined as $e_t = F_t - Y_t$ for $t = 1...n$. Then the MASE is defined as:

$$MASE = mean\left(\frac{|e_t|}{\frac{1}{n-1}\sum_{i=2}^{n}|Y_i - Y_{i-1}|}\right) \quad (1)$$

MASE scales the error with the error from a one step naïve forecaster, which takes the last observation as the forecast.The MASE error is scale-independent and can be used for comparisons across multiple time series [8]. In contrast to the mean relative error, it is not influenced by skewed error distributions and thus ensures unbiased comparisons [8]. Assuming a forecast horizon of one interval, a MASE < 1 indicates that on average the considered forecast method yields smaller errors than the one step naïve approach. In our experiments, we forecast the workload for a complete day resulting in a forecast horizon of 24 or 48 depending on the sampling interval. Therefore, MASE is expected to be larger than one [8].

3) Results: Figure 2 shows the resulting forecasts by the splitTs method on the FIFA'98 traces. The splitTs is able to capture the daily and weekly patterns (i.e., the differences between weekdays and weekends) in its model and to reflect them in the forecasts. The forecasting method is executed at 3am each night when the workload is low. Due to space constraints, the complete forecast time series of the Wikipedia and CICS transaction traces are not included in this paper. More information on their shape can be found in [6] and [7].

In order to assess the improvement achieved in our splitTs approach, we compare its forecast accuracy to three state-of-the-art approaches, WCF [7], ARIMA [9] and tBATS [10] (using their implementations from the R package `forecast` [11]). The first days required to learn the seasonal patterns are excluded from the comparison (in total 6 complete days as the splitTs approach requires three workdays and three non-workdays to learn the seasonal patterns).

TABLE I
Forecast accuracy

		splitTs	WCF	ARIMA	tBATS
FIFA'98	MASE	1.42	2.18	2.65	2.33
	Infs	0	0	0	24
	C.I width	840	1075	852	954
Wikipedia	MASE	1.14	1.28	1.68	2.21
	Infs	0	0	0	0
	C.I width	39878	50820	58683	43323
CICS transactions	MASE	1.23	3.01	4.97	3.28
	Infs	0	0	0	0
	C.I width	9584	24774	15322	24536

Table I shows the summarized forecast errors of the splitTs compared with the WCF, ARIMA and tBATS methods. The splitTs approach can reduce the MASE errors by between 11% (Wikipedia traces) and 59% (CICS transaction traces). The differences can be explained by the different weekly patterns present in these two traces. The CICS transaction traces stem from an enterprise application and the differences between the weekend and weekday workloads are higher. In contrast, the Wikipedia traces have less distinctive weekend patterns and therefore, the improvements through the splitTs approach are smaller. Figure 3 shows the distribution of the absolute scaled errors.

In Table I, we also included the number of infinite values forecast by the methods and the mean width of the 80% confidence interval. On the FIFA'98 traces, the tBATS ap-

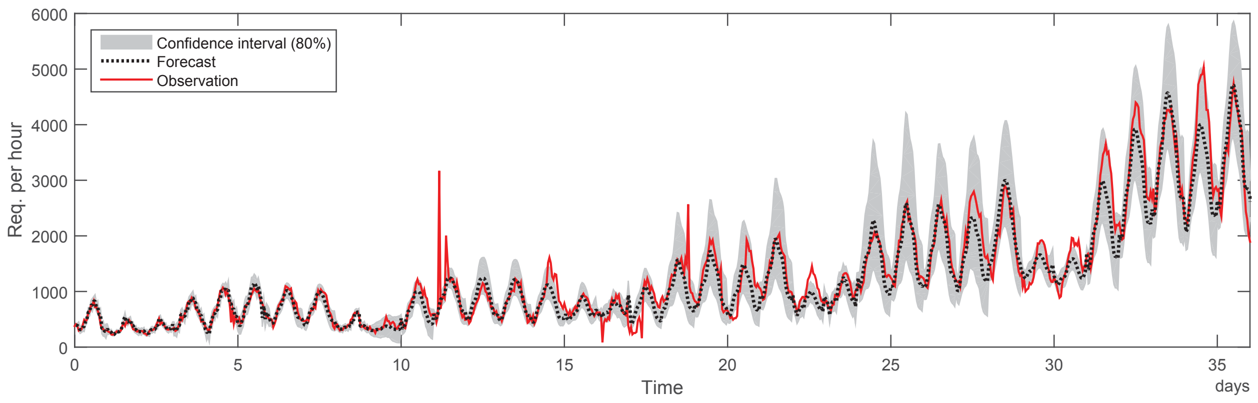


Fig. 2. Forecasts using splitTs on the FIFA'98 traces (first day is Friday).

proach fails to fit a forecast model in one interval resulting in 24 infinite values. The mean confidence interval length is significantly reduced by the splitTs on all three traces indicating a better fit of the forecast model to the observations.

In summary, the results show that by classifying different types of days and executing the time series analysis only on subsets with similar seasonal patterns, splitTs improves forecast accuracy significantly compared to existing state-of-the-art techniques based on time-series analysis. On the considered data sets, existing time-series analysis techniques were not able to forecast the overlapping weekly seasonal patterns (workingdays vs. nonworkingdays) correctly.

V. Vertical Memory Scaling

In this section, we evaluate in a case study how the proactive approach can help to reduce the impact of the reconfiguration on the application availability and performance by comparing it with a reactive approach. The case study is based on the Zimbra Collaboration Server [1] (or Zimbra), an open-source groupware and collaboration server.

A. Experiment Setup

Zimbra is a distributed enterprise application consisting of a *mailbox server* (Java application server and MySQL database), a *mail transfer agent (MTA)* and a *LDAP server*. The mailbox server manages a mailbox for each Zimbra user and provides multiple interfaces (e.g., SOAP, IMAP, POP3) to access a mailbox. A mailbox contains the user's mails, calendars, address books, etc. The MTA is responsible for checking incoming and outgoing mails for spam and virus content and delivering them to the recipients' mailboxes. The LDAP server manages the central configuration of Zimbra and provides user authentication services.

In our setup, we deployed Zimbra using two VMs, one for the mailbox and LDAP servers, and the other for the MTA. The Zimbra VMs and the load generator were deployed on three physical hosts (each equipped with 8 core CPUs, 16 GB RAM and 500 GB HDD) running VMware vSphere 5.5. The VMs were configured with 2 vCPUs and 4 GB RAM running a CentOS 7.0 64-bit OS. We used Zimbra 8.5 and adapted its database configuration to store and reload the MySQL buffer pool when restarting in order to reduce cache warm up times.

Each mailbox contains approximately 5000 messages with content and different types of attachments from a dump of a set of mailboxes from a production mail server. A load generator executes a session-based workload on Zimbra consisting of a sequence of login, several mail read, mail send, and mail delete operations. Each session randomly chooses a mailbox on the server from a uniform distribution. The number of concurrent sessions is dynamically varied over the duration of an experiment. Given that we could not obtain suitable arrival traces from a production Zimbra server, we extracted the workload intensity from the FIFA'98 trace and scaled it to match the capacity of our system.

B. Proactive vs. Reactive Controllers

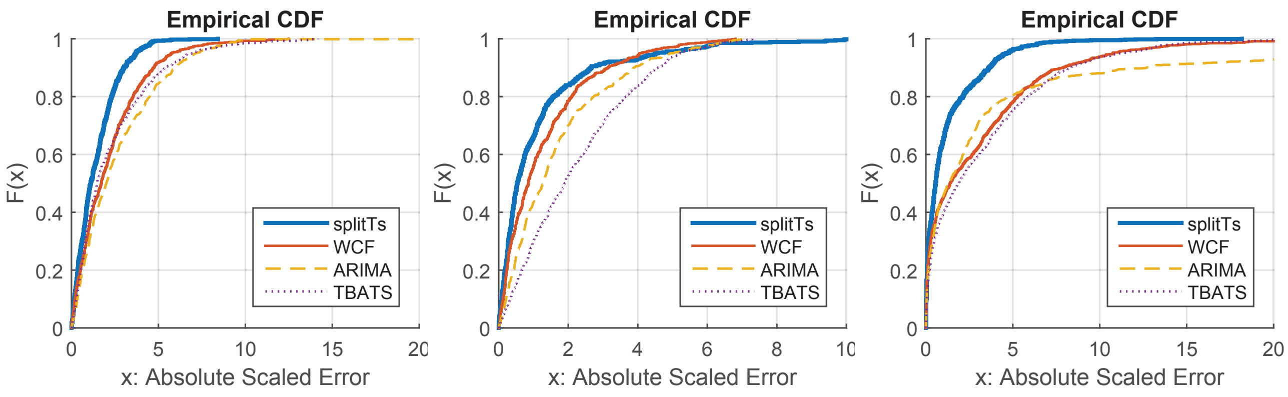
We compare our proactive controller to a threshold-based, reactive controller. In addition, we performed one baseline experiment without vertical memory scaling.

As the application does not directly support the observation of the incoming load, we use the monitored throughput as an approximation for the number of arriving requests, and learn the forecasting model based on the throughput. Given that we aggregate the collected statistics over a complete hour, we argue that this is a safe approximation. The proactive controller is configured to do a forecast every night at 3 AM when a minimum load on the system is expected. It predicts the number of requests for the next day resulting in 24 hourly arrival rates of which it takes the maximum. As thresholds, we use the maximum sustainable throughput of the system for a given memory size. We determined these thresholds in an offline profiling experiment. The controller reconfigures the memory in 4096 MB steps.

The reactive controller monitors the availability and response time of the mailbox server. It is triggered if the server is unavailable or if the observed response times are above one second for over a period of three minutes. The controller has a quiet time of one hour, i.e., after a memory reconfiguration the trigger will not fire again in this period.

C. Results

1) Impact of reconfiguration: In this experiment, we evaluate the impact of memory scaling on the availability and performance of the application. We used the setup described



(a) FIFA'98 (b) Wikipedia (c) CICS transactions

Fig. 3. Cumulative distribution function of absolute scaled errors.

TABLE II
COMPARISON OF CONTROLLERS.

	No control	Reactive	Proactive
Mean response time	7,567 ms	1,211 ms	52 ms
Maximum response time	349,830 ms	1,023,100 ms	1,077 ms
Timeouts	84	285	0
Errors	8493	1485	337
Time of reduced availability	176 min	33 min	4 min

in Section V-A for the experiment. The FIFA'98 trace covers a period of 92 days making it infeasible to run an experiment over the complete length. Therefore, we extracted a subset of four days from the trace, scaled its length to an experiment length of 16 hours, and used it as an input to our load generator. The subset covers the period from Saturday, June 30th 3:00 AM to Wednesday, July 4th, 3:00 AM.

In order to reduce the experiment duration, we used four weeks of historic data from the FIFA'98 trace to learn the forecast model. As the experiment begins, the proactive controller automatically switches to the live monitoring data from the Zimbra server. The live data is continuously appended to the historic data and the forecast model is updated according to the new observations.

In the following, we define the term "availability" as the number successful requests divided by the total number of requests during a time interval. If the availability is below 100%, we consider it *reduced availability*. Here the success of each request is measured from the client perspective, i.e., as observed by the load generator. If a request times out due to an overloaded application, we consider the request unsuccessful.

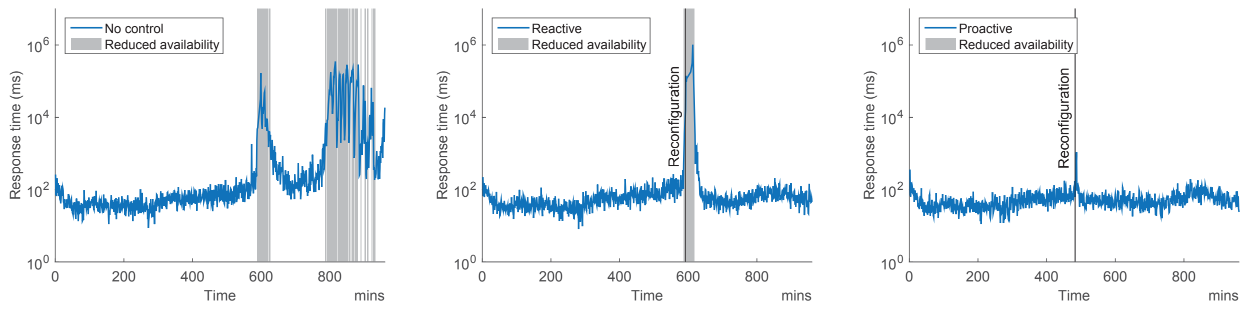
Figure 4 shows the observed average response time for three different experiments including the time of reduced availability during which not all requests can be successfully served. In the first experiment in Figure 4(a), no additional memory is added to the VM. On the third and forth day, the application is overloaded as the memory becomes a bottleneck resulting in high read load on the hard disk. The response times of the application therefore increase significantly (see also Table II). During peak periods the application is not able to serve all requests. In total, the availability is below 100% for a period of 176 minutes due to timeouts and connection errors (see Table II).

The reactive controller in Figure 4(b) is triggered by the unavailability of the application as the response times increase too abruptly for the controller to react.

As the application is overloaded, the steps described in Section III-2 take a long time to complete and afterwards the application also needs to reload its caches under a high workload causing additional overhead. In total, the application is only partially available over a period of 33 minutes due to the overload situation and the reconfiguration (see Table II). After that period, the reconfiguration effectively mitigated the memory bottleneck and the application is able to serve the workload with an acceptable performance again.

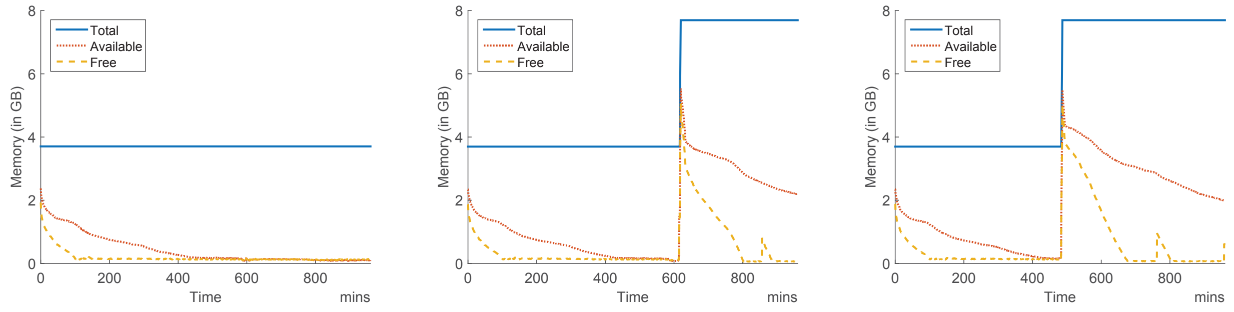
The results from the proactive controller are shown in Figure 4(c). The proactive controller correctly detects the future memory bottleneck in the night between the second and third day and proactively triggers the reconfiguration during a phase of low load. This results in a much lower impact of the reconfiguration on the availability and performance of the application. Given that the application needs to be restarted, it is unavailable for a period of 4 minutes (see Table II). After this period, it reloads its caches and serves user requests in parallel without overloading the VM. Compared to the reactive controller, our approach reduces both the time of lower availability and the number of errors during the reconfiguration by more than 80%. We conclude that using a proactive control approach to memory scaling can effectively reduce the impact of the required reconfigurations on the application.

2) Memory usage: Figure 5 shows the observed memory usage of the application as reported by the guest operating system in the VM. The free counter reports the unallocated memory and corresponds to the memory usage as seen by the hypervisor. The available counter does not include the memory reserved for the operating system buffer caches. As the Linux operating system greedily allocates memory for its buffer caches, the free counter does not reflect the actual memory demand of the VM. The available counter is a better indicator of the memory pressure within the VM. However, it does not show a clear correlation to the workload intensity. Therefore, it is necessary in our approach to benchmark the



(a) No control (b) Reactive control (c) Proactive control

Fig. 4. Observed response times of the mailbox server.



(a) No control (b) Reactive control (c) Proactive control

Fig. 5. Observed memory usage of the mailbox server VM.

TABLE III
ALLOCATION DECISIONS

	Step size	Overprovisioned		Underprovisioned	
		Days	Amount	Days	Amount
splitTs	1024 MB	3	1.08%	14	10.79%
	2048 MB	0	0%	9	10.07%
	4096 MB	0	0%	6	8.54%
WCF	1024 MB	6	3.6%	18	14.03%
	2048 MB	4	3.36%	14	13.42%
	4096 MB	2	2.44%	8	10.98%

application in advance to determine the maximum number of session that can be served with a given memory size.

3) Allocation decisions: Using the same thresholds as in the reconfiguration experiments, we also analyzed the allocation decisions of the proactive controller for the complete first 5 weeks of the FIFA'98 trace. Excluding the training phase of the forecaster, this results in 30 days for which the proactive controller can predict the required memory size. We compare the memory allocation resulting from the forecast arrival rate to the memory allocation that would be required for the actual arrival rate. As the memory allocation can only be a multiple of a certain step size in our approach, we calculated the over- and under-provisioning ratios using step sizes of 1024 MB, 2048 MB, and 4096 MB. The results are shown in Table III.

In summary, the proactive controller using the splitTs method results in a lower chance of over- and under-provisioning ($< 11\%$) compared to the WCF method. However, both methods tend to underestimate the resource allocation on the FIFA'98 trace. This is also visible in Figure 2. Although splitTs correctly captures the long-term increasing trend of the trace, more sophisticated methods to extract overlapping trends and seasonal patterns may improve the results. This will be part of our future work on combining splitTs with the load intensity modeling framework DLIM [12]. In order to obtain more conservative forecasts and reduce underprovisioning, we recommend to use the upper confidence level of the forecast. Compared to a constant factor as proposed in [13], the confidence interval has the advantage that its width also depends on the fitting quality of the forecast model.

VI. Related Work

Prior work on proactive resource management describe different approaches to use the demand forecast in allocation decisions. Surveys on the state-of-the-art in resource management of virtualized environments [14], [15], [16] list different approaches to address vertical scaling via VM resizing in a proactive manner. All methods in this category focus on the dynamic provisioning of virtual CPUs as the computational resource. Our work focuses on proactive provisioning of memory resources for memory intensive applications.

In [17], the authors describe how proactive resource management might be achieved at the granularity of an entire cluster of virtual machines. vManage [18] uses short range forecasts (15 minutes into the future) to optimize VM placement on physical hosts and avoid ping-pong of VM migrations. In [19], the authors use a combination of reactive and proactive techniques to horizontally scale an application to meet workload demands. AppRM [20] dynamically adapts the CPU and memory reservation and limit settings in a reactive manner. The scale up is, however, limited to the initially configured resource capacity. CloudScale [13] uses multiple techniques including *Fast Fourier Transforms (FFTs)* –to identify repeating resource usage patterns–, discrete time *Markov Chains* – to predict the demand in the near future– , online adaptive padding and incremental (adaptive) over-provisioning –to remedy and detect under-provisioning–. This information is used for short-term optimization of CPU and memory limits.

Approaches to dynamically adapt the memory management of applications are described in [3], [21], [22].The authors of [3] integrate memory balooning techniques in Java and MySQL applications to improve elasticity. In [21], a control mechanism is proposed to dynamically manage the heap size of Java applications running in the same VM depending on the current demand. Gingko [22] is an over-commitment framework for memory that is aware of application memory management. These approaches are orthogonal to ours and may be used to improve the elasticity further.

Our work, differs from these related works in the following aspects: (a) it leverages memory hot-add mechanisms of VMware ESX, (b) it enables application reconfigurations necessary to exploit the additional memory resources, and (c) it uses mid-term workload forecasts (e.g., one day horizon) to automatically schedule reconfigurations during a pre-defined maintenance window.

VII. Conclusion

In this paper, we present a proactive approach for vertical scaling of resources with a focus on memory intensive applications that often do not support state replication across nodes. We explicitly address the technical challenges of dynamic memory provisioning. Furthermore, we present our splitTs method to demand forecasting, which is based on multiple time series analysis methods and also incorporate meta-knowledge about the expected workloads. In our evaluation based on real-world traces, we demonstrate that splitTs significantly improves the forecasting accuracy. In the context of a case study with the Zimbra Collaboration Server, we show that our proactive approach can reduce the impact of reconfigurations on the application availability and performance of the application by more than 80% compared to a reactive controller. Using splitTs, we were also able to reduce the over- and under-provisioning of memory compared to WCF to under 11%.

As part of our future work, we plan to extend the descriptive modeling capabilities for capturing relevant meta-information to further improve the forecasting accuracy. Combining our splitTs method with the load intensity modeling framework DLIM described in [12] appears promising. We also plan to extend the set of experiments as soon as a standardized benchmark methodology for evaluating resource management mechanisms, as presented in [23], is available. Finally, we plan to further extend our approach to include methods for estimating memory demands based on monitoring data from the hypervisor in order to automatically determine the required memory size for a given workload at system runtime.

References

[1] “Zimbra.” [Online]. Available: http://www.zimbra.com

[2] C. A. Waldspurger, “Memory resource management in VMware ESX server,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 181–194, 2002.

[3] T.-I. Salomie, G. Alonso, T. Roscoe, and K. Elphinstone, “Application level ballooning for efficient server consolidation,” in *Proc. of the 8th ACM European Conf. on Computer Systems*, 2013, pp. 337–350.

[4] R. J. Hyndman and Y. Khandakar, “Automatic Time Series Forecasting: The Forecast Package for R,” pp. 1–22, 7 2008.

[5] M. Arlitt and T. Jin, “A workload characterization study of the 1998 World Cup Web site,” *IEEE Network*, vol. 14, no. 3, pp. 30–37, May 2000.

[6] G. Urdaneta, G. Pierre, and M. van Steen, “Wikipedia workload analysis for decentralized hosting,” *Elsevier Computer Networks*, vol. 53, no. 11, pp. 1830–1845, July 2009.

[7] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn, “Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning,” *Concurrency and Computation - Practice and Experience, John Wiley and Sons, Ltd.*, vol. 26, no. 12, pp. 2053–2078, 2014.

[8] R. J. Hyndman and A. B. Koehler, “Another look at measures of forecast accuracy,” *Intl. J. of Forecasting*, vol. 22, no. 4, pp. 679 – 688, 2006.

[9] G. Box, G. Jenkins, and G. Reinsel, *Time Series Analysis : Forecasting and Control*, 4th ed. Wiley, 2008.

[10] A. M. De Livera, R. J. Hyndman, and R. D. Snyder, “Forecasting time series with complex seasonal patterns using exponential smoothing,” *Journal of the American Statistical Association*, vol. 106, no. 496, pp. 1513–1527, 2011.

[11] “forecast: Forecasting functions for time series and linear models.” [Online]. Available: http://cran.r-project.org/web/packages/forecast

[12] J. G. von Kistowski, N. R. Herbst, D. Zoller, S. Kounev, and A. Hotho, “Modeling and extracting load intensity profiles,” in *Proc. of the 10th Intl. Symp. on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*, May 2015, accepted.

[13] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, “Cloudscale: Elastic resource scaling for multi-tenant cloud systems,” in *Proc. of the 2nd ACM Symp. on Cloud Computing*, 2011, pp. 5:1–5:14.

[14] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, “Auto-scaling techniques for elastic applications in cloud environments,” Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09, Tech. Rep., 2012.

[15] B. Jennings and R. Stadler, “Resource management in clouds: Survey and research challenges,” *Journal of Network and Systems Management*, pp. 1–53, 2014.

[16] G. Galante and L. C. E. d. Bona, “A Survey on cloud computing elasticity,” in *Proc. of the 2012 IEEE/ACM Fifth Intl. Conf. on Utility and Cloud Computing*, 2012, pp. 263–270.

[17] G. Shanmuganathan, A. Gulati, A. Holler, S. Kalyanaraman, P. Padala, X. Zhu, and R. Griffith, “Towards proactive resource management in virtualized datacenters,” *Runtime Environments, Systems, Layering and Virtualized Environments (RESoLVE)*, 2013.

[18] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan, “vManage: Loosely coupled platform and virtualization management in data centers,” in *Proc. of the 6th Intl. Conf. on Autonomic Computing (ICAC 2009)*, 2009, pp. 127–136.

[19] B. Urgaonkar, P. J. Shenoy, A. Chandra, and P. Goyal, “Dynamic provisioning of multi-tier internet applications.” in *Proc. 2nd Intl. Conf. on Autonomic Computing (ICAC)*, 2005, pp. 217–228.

[20] L. Lu, X. Zhu, R. Griffith, P. Padala, A. Parikh, P. Shah, and E. Smirni, “Application-Driven dynamic vertical scaling of virtual machines in resource pools,” in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2014)*, May 2014, pp. 1–9.

[21] N. Bobroff, P. Westerink, and L. Fong, “Active control of memory for java virtual machines and applications,” in *11th Intl. Conf. on Autonomic Computing (ICAC)*, Jun. 2014, pp. 97–103.

[22] M. Hines, A. Gordon, M. Silva, D. da Silva, K. D. Ryu, and M. Ben-Yehuda, “Applications know best: Performance-driven memory overcommit with ginkgo,” in *2011 IEEE Third Intl. Conf. on Cloud Computing Technology and Science (CloudCom)*, Nov 2011, pp. 130–137.

[23] A. Weber, N. R. Herbst, H. Groenda, and S. Kounev, “Towards a resource elasticity benchmark for cloud environments,” in *Proc. of the 2nd Intl. Workshop on Hot Topics in Cloud Service Scalability (HotTopiCS)*, March 2014.