HyPA: Hybrid Horizontal Pod Autoscaling with Automated Model Updates

Kaan Aykurt*, Răzvan-Mihai Ursu*, Johannes Zerwas*, Patrick Krämer*,
Navidreza Asadi*, Leon Wong† and Wolfgang Kellerer*
*Technical University of Munich, †Rakuten Mobile Inc.

***Abstract*—Due to changing demand patterns driven by technological advancements and the rise of new applications and services, the provisioning of heterogeneous workloads is a crucial component of the resource allocation problem. Traditional resource allocation strategies such as reactive autoscaling or prediction-based proactive solutions, fail to meet the desired performance goals when the underlying demand arrival pattern changes.**

In this paper, we present HyPA, which combines reactive and proactive components to autoscale pods in a Kubernetes environment. In contrast to previous approaches of hybrid autoscaling, HyPA automatically reacts to drifts in the request arrival pattern. Specifically, it updates the model of its proactive component when the prediction performance decreases. The evaluation in a simulation on a variety of real-world traces, spanning multiple days, demonstrates that HyPA improves upon existing purely reactive and purely proactive horizontal pod autoscalers.

***Index Terms*—Kubernetes, horizontal pod autoscaling, demand forecasting**

I. Introduction

In today's rapidly evolving world of digitalization, the provisioning of containerized workloads is a crucial aspect of modern network management strategies. Examples of such workloads are web applications and mobile networks which become more modularized as white-box concepts like Open-RAN are adopted [1]. Kubernetes (k8s) is one of the most popular frameworks for managing and scaling containerized applications and it is being adopted by many Telco Providers where network functions are containerized. One of the key features of k8s is that it offers to tackle the challenges of scalability with its Horizontal Pod Autoscaler (HPA).

Traditional resource provisioning methods require expertise and human intervention in the decision-making process to allocate the required resources. This leads to a tedious and error-prone process, where a wrong estimation in the capacity requirements can lead to under or over-provisioning of resources [2]. In addition to application performance issues caused by under-provisioning, or waste of resources created by over-provisioning, rapid fluctuations in demand patterns render manual resource allocation methods infeasible and impractical. To this end, the k8s HPA aims to autonomously scale the number of running pods, i.e. autoscale, in response to critical performance metrics such as CPU utilization, memory utilization, or the Request Completion Time (RCT).

State-of-the-art autoscaling methods from the research community consider reactive, proactive, and hybrid autoscaling [3]. Reactive autoscalers take action when the performance metrics exceed a pre-defined threshold. Hence, their reaction is delayed or even too late to keep the performance metrics within the desired range. Avoiding this situation usually results in over-provisioning resources to have some slack resources (and lead time) to deploy additional resources when needed. Moreover, frequently collecting critical performance metrics from large-scale distributed systems introduces challenges to system design.

With the advances in data-driven modeling and forecasting methods, a proactive approach to HPA emerged [4]–[6]. The HPA predicts the incoming demand for a certain time horizon and scales out the resources accordingly. While this allows the system to prepare for future load variations, the performance of such an approach strictly depends on the quality of the forecast [6], [7]. In addition, user and application behavior is subject to (sudden) changes that may only be for a short duration or more long term, e.g., when an update of the application is deployed. In general, short-term effects appear in the form of bursts. For instance, for an online food delivery website, the incoming request numbers increase during lunch or dinner times. On the other hand, long-term effects reflect distributional shifts in the data where an increase or a decrease in the mean occurs due to seasonal effects. An example of such effects can be the increased number of requests for football-related web pages during periods of the World Cup. These changes reflect in the request arrival patterns as bursts or so-called distributional shifts for more persistent changes, and in turn may lead to wrong forecasts and ultimately, reduced performance.

In order to respond to changing user behaviors and the resulting deficiencies of proactive HPAs, *hybrid* HPAs were introduced [3]. Existing designs of such systems feature a combination of proactive and reactive approaches. Specifically, they rely on scaling decisions from the proactive component as long as the forecast quality is high and fall back to the reactive component if needed. A specific example is Chameleon [7]. Chameleon features two proactive components that use different prediction models and comes with a sophisticated mechanism to handle conflicting scheduling decisions. It periodically evaluates the forecast quality and falls back to a reactive approach when the forecast quality is below a given threshold. While this approach works well for short-period bursts, it lacks an integrated approach to update the forecasting models in case of long-term distributional shifts such as observed during the COVID19 pandemic [8].

In this paper, we propose HYPA, a hybrid HPA that automatically reacts to distributional shifts in the demand patterns. Similar to other systems, HYPA combines a proactive and a reactive HPA component. In normal operation mode, it monitors and checks the proactive component's performance. In case of performance degradation, HYPA enters into burst mode and applies reactive autoscaling in addition to the proactive scaling. If it continuously detects bursts for a longer time period, HYPA considers the model of the proactive component to be outdated and starts data collection for the update of the model. We evaluate HYPA in simulations on traces from production systems in a variety of settings, comparing it to purely proactive and purely reactive HPAs. HYPA outperforms the baselines in all these situations or performs at least as well while putting more focus on our primary metric, the RCT.

II. Related Work

The performance of an orchestration tool depends on its ability to use the resources efficiently with respect to changes in application characteristics, i.e., it should scale the resources provisioned to their applications in a smart manner. Hence, a considerable part of the literature focuses on designing autoscalers. A survey by Qu [3] elaborates on various methodologies to scale the resources of web applications. Mainly, the design of autoscalers is divided into three categories: reactive, proactive, and hybrid autoscalers.

Reactive autoscalers update the autoscaling decision when a pre-defined reaction threshold is reached. The default k8s HPA implements this as a control loop that queries the resource utilization and by comparing it with a pre-defined target utilization value, it reactively scales pods. Although this approach manages to scale after a certain threshold is reached, it has no ability to anticipate changes in workload patterns in the future. Hence, it can scale only after the control loop reacts to an increase in resource utilization.

Proactive autoscalers predict the required resource consumption in the future in order to scale by avoiding the need for reactive autoscaling. Among this category, the authors of [5] propose an LSTM based-model to scale based on the predicted workload. The authors of [4] present AutoScale– as a holistic approach that incorporates Load Balancing of requests and reactive autoscaling and it scales servers according to the current request arrival rate. Gandhi et al. [9] present an Extended Kalman Filter based method to estimate unobservable parameters in a three-tier web application, and use those estimates for autoscaling decisions. Luong et al. [10] proposes to combine a long-term and a short-term prediction to scale the number of application instances. The authors of [11] propose an algorithm to detect bursts and act proactively upon the detection of bursts. However, these methods do not include a reactive component. Therefore, in case of poor prediction performance, the autoscaler's performance degrades significantly. To overcome this problem, state-of-the-art in literature focuses on hybrid autoscalers.

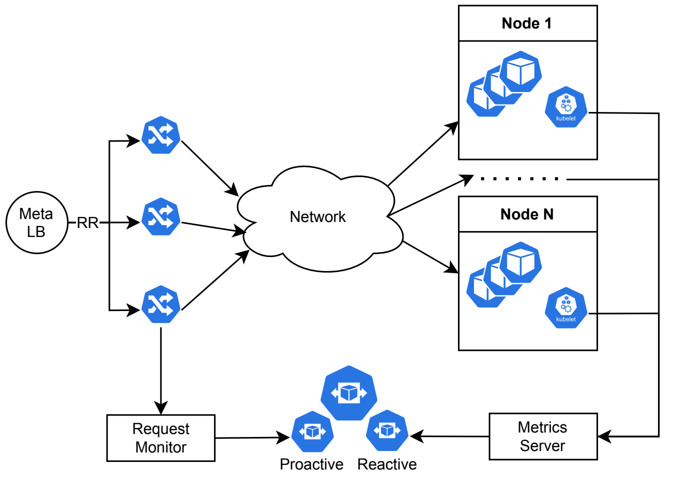


Fig. 1: Cluster setup for the Change Point Detection use-case.

Hybrid autoscalers combine reactive and proactive components for autoscaling decisions. The authors of [12] propose a hybrid controller, where the reactive component is responsible for scale-out decisions and the proactive component accounts for scale-in decisions. Overall, they show that their autoscaler outperforms purely reactive-based autoscalers. Most recently, Chameleon [7] and Chamulteon [13] frameworks combine proactive scaling mechanisms with reactive fallback. In their paper, the proactive components use two time series models to predict the future request arrival rate. The reactive component makes scaling decisions based on the current request arrival rate and the estimated application resource consumption profile. Further, the controller allows the integration of queuing models into the decision-making process, enabling the controller to use structural application knowledge. However, they do not consider updating the model parameters during runtime, i.e. they rely on a pre-trained model without considering distributional shifts in request patterns.

To the best of our knowledge, the literature lacks a comprehensive and lightweight hybrid autoscaler that automatically updates its proactive component in case of distributional shifts, and falls back to reactive autoscaling when the prediction performance of the proactive component decreases. To close this gap, we present HYPA.

III. Scenario Description

Figure 1 shows the envisioned scenario. We consider a deployment with 10s of pods and multiple stages of load balancers as considered in prior work [14], [15]. A meta load balancer distributes incoming requests across the second stage of load balancers using round robin. In the second stage, load balancers distribute the requests across the pods of the service(s). They might use more advanced load balancing techniques, e.g., based on the load of the pods. Two sets of monitoring data are collected from the system. The first set consists of platform metrics such as CPU utilization $u(t)$, memory utilization, disk I/O, and network utilization. The `Metrics Server` component available in k8s collects these

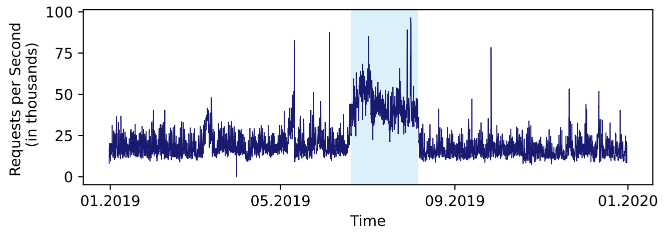


Fig. 2: Distributional Shift: The mean of the dataset faces a shift in the indicated time period. Hence, a performant proactive component needs to account for structural changes in the request patterns.

metrics from all nodes in the cluster. The reactive autoscaler component uses platform metrics for its scaling decisions. The second set of monitoring data is request-level data such as the number of arrived requests in a time interval $n_{req}(t)$. A dedicated request monitor collects this data from (a subset of) the load balancers. The proactive component uses the request-level data to evaluate the quality of its forecasting model and to perform burst detection. We consider *horizontal* autoscaling, *vertical* scaling is out-of-scope for this work.

A. *Workload Characterisation*

The existence of distributional shifts of demand patterns has been observed in prior work, e.g., [8]. To provide another example, we analyze the request arrivals of the BibSonomy system [16]. We investigate the request arrivals to the cluster in fixed time intervals t_{iar}. Specifically, requests are binned and counted with $t_{iar} = 1\,\mathrm{min}$. Our analysis shows that the number of requests that arrive every t_{iar} in the BibSonomy traces form a seasonal time series. Hence, the time series expresses regular and predictable changes that occur at specific times, i.e., after removing seasonal effects, the time series is stationary.

Moreover, our analysis also considers the time series of request arrivals may express random bursts in demand. Given the fact that the time series is stationary given the seasonality, we define a burst as a finite period of time during which the number of arrived requests is larger (or smaller) than an upper (or lower) bound on the predicted request arrivals.

Figure 2 shows the number of requests per second in the year 2019 for the BibSonomy dataset. The plot shows that in the marked period, the mean shifts for a period of 1.5 months, and then returns to the old mean. This is an example of possible abrupt changes in the request patterns. Motivated by this, we propose a hybrid autoscaling scheme consisting of a proactive and a reactive component, where the proactive component continuously predicts the required number of pods in the future and updates the model parameters in case of a reduction in prediction quality and falls back to reactive autoscaling in such cases.

IV. Hybrid Autoscaling with HyPA

HyPA combines the decisions of two autoscaling components: proactive and reactive. The two components both estimate the number of desired replicas in the cluster, $r_p(t)$ for the proactive, and $r_r(t)$ for the reactive component. The proactive component uses a model of the demand pattern to forecast the request arrival rate in the next forecast period, whereas the reactive component reacts to the utilization of the pods. Both scale the number of replicas to reach a target utilization u_t on average. The currently configured number of replicas is $r_c(t)$. The control loop estimates the number of replicas in the next iteration as $r_c(t) = \max(r_r(t), r_p(t))$. This policy prevents pre-mature scale-in decisions.

Each component of the autoscaler operates in its own synchronization period t_r (reactive) and t_p (proactive). During normal operation, we assume that $t_r > t_p$, i.e., the proactive component operates more often than the reactive component. The reason for this lies in the cost to obtain necessary monitoring data from the cluster nodes. Obtaining all platform metrics from a large number of nodes is much more costly than collecting request data from a small number of load balancers. One goal is the reduction of this overhead by relying on the predictive model in the proactive component. Moreover, a dominant proactive component allows one to follow the demand pattern more closely and increase resource efficiency. The difference $t_r - t_p$ is such, that the reactive component has enough time to react to demand peaks, i.e., bursts. The difference must be chosen with respect to the provisioning time of new replicas, the monitoring frequency of request arrivals, and an expectation of the speed with which bursts occur, i.e., how fast demand increases during bursts.

A. *HPA Components*

The following describes the scaling logic of the two HPA components of HyPA more in detail.

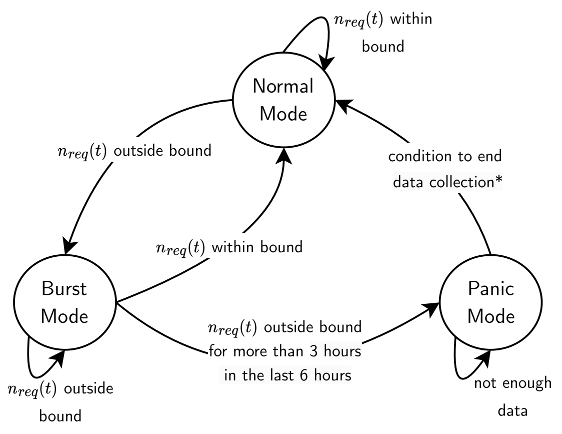
1) Reactive: The reactive component builds on the default HPA available in k8s. It uses the CPU utilization of the application pods to determine the number of needed replicas. Specifically, it calculates the average utilization over all pods and compares it against the given target value u_t:

$$r_r(t) = \left\lceil r_c(t-1) \cdot \frac{u(t)}{u_t} \right\rceil . \tag{1}$$

2) Proactive: The proactive component uses data-driven models to forecast the expected number of requests in each category for the next time slot. Based on the estimated number of requests and a model of the work that each request causes, the so-called application profile [17], the proactive component estimates the number of replicas in the system:

$$r_p(t) = \left\lceil r_c(t-1) \cdot \frac{\mathrm{E}[u(t+1)])}{u_t} \right\rceil . \tag{2}$$

Here, $\mathrm{E}[u(t+1)]$ uses the forecast request arrivals and the application profile to estimate the expected utilization of the cluster in the next time slot.



$n_{req}(t)$ within bound
Normal Mode
$n_{req}(t)$ outside bound
condition to end data collection*
$n_{req}(t)$ within bound
Burst Mode
Panic Mode
$n_{req}(t)$ outside bound for more than 3 hours in the last 6 hours
$n_{req}(t)$ outside bound
not enough data

Fig. 3: Hybrid HPA State Machine. HYPA consists of 3 states. Depending on different conditions specified in the figure, the states, and hence the operation mode of the HPA changes.

The proactive component works on a minute granularity (to account for the reaction times of the k8s cluster, e.g., for pod scheduling and application initialization). Specifically, the proactive component predicts the request arrival rate per second averaged over one-minute time windows. Designing the forecasting model for the resulting time series is an engineering task on its own. We found that for the used trace from the BibSonomy production system, a rather simple model consisting of a fixed offset and a seasonal component suffices. Moreover, we identified a strong diurnal and weekly pattern. Therefore, the seasonal component $s(x)$ is given by a look-up table with a single value for every minute of a week (10 080 values in total). As a result, we obtain

$$a(t+1) = b + s(mow(t+1)), \tag{3}$$

where $mow(.)$ returns the minute of the week for the given timestamp t (in minutes). We obtain these values by averaging each time slot over multiple weeks. Besides the mean value, the model also provides the standard deviation $std(.)$ as a measure of prediction confidence.

B. Operation Modes

Figure 3 overviews the state machine of HYPA and the different operation modes. When it detects deviations of the forecast from the observed values, HYPA first enters the so-called "Burst mode". If it is frequently in the Burst mode, HYPA transitions to the Panic mode. Here, it collects new data and updates the model. The modes and the transition conditions are described in more detail in the following.

C. Burst Detection & Burst Mode

The autoscaler detects a burst at a time slot based on the predicted number of requests $\tilde{n}_{req}(t)$ $(a(t) \cdot 60\,\mathrm{s})$, and the actual observed number of requests in that time slot $n_{req}(t)$. Currently, the burst detection focuses on positive bursts, i.e., time intervals in which the actual demand *exceeds* the predicted demand, since such bursts have an impact on the RCT. Specifically, HYPA detects a burst if $\frac{n_{req}(t)}{60s} \notin [a(t) - std(t), a(t) + std(t)]$.

During a burst, $r_r(t) > r_p(t)$ and the estimate of the reactive component is chosen. Once the burst is over, $r_r(t) \leq r_p(t)$, and the excessive number of replicas is scaled-in. Moreover, when the autoscaler detects a burst, the cluster changes the configuration of the reactive component. HYPA enters the so-called "Burst mode". Specifically, the synchronization period t_r is reduced to burst synchronization period t'_r. Finally, the collection of the platform metrics and reactive components is triggered. In the case of a burst, the reactive component alone is responsible for choosing the adequate number of replicas, i.e., $r_c(t+1) = r_r(t)$ due to the definition of bursts. Thus, the cluster actively changes the configuration of the reactive component. Since obtaining fresh measurements and the provisioning of new instances takes time, a scale-out threshold must be set such that the reactive component reacts when the cluster is not yet over-utilized.

D. Change points detection & Panic Mode

Next to expected unpredictable bursts, the cluster further monitors the accuracy of the forecasting model, specifically, the cluster performs a change point analysis. At a change point, the predictions of the forecasting model deviate significantly from the actually measured values. The difference to a burst is that for a burst, the predictions are off for a finite (short) duration. After some time, the burst is over and the model is accurate again. In case of a change point, the predictions of the models are inaccurate for at least i_{cp} prediction intervals within an observed time window w_{cp}:

$$\sum_{j=t-w_{cp}}^{t-1} \mathbf{1}\left(\frac{n_{req}(j)}{60s} \notin [a(j) - std(j), a(j) + std(j)]\right) \geq i_{cp}. \tag{4}$$

$\mathbf{1}(.)$ is an indicator function that $= 1$ if the condition evaluates to true.

For example, the total demand level could decrease and the forecast would always be too high, resulting in increased costs due to over-provisioning. Similarly, the estimates could be too low, resulting in Service-Level Agreement (SLA) violations since the reactive component does not have enough time to scale-out.

If the cluster detects a change point, the system enters "Panic mode" and starts data collection in order to update the model of the demand. This approach reduces the overhead of constant data collection. Moreover, for the whole duration of the data collection, the HPA operation of the burst mode is enforced. For data collection, HYPA records the observed arrival rates and fits the updated model when a sufficient number of samples was collected.

In principle, the duration of the panic mode is a trade-off between model accuracy and the time until the new model is available. It depends on the details of the used model and the observed data. It can either be fixed or determined

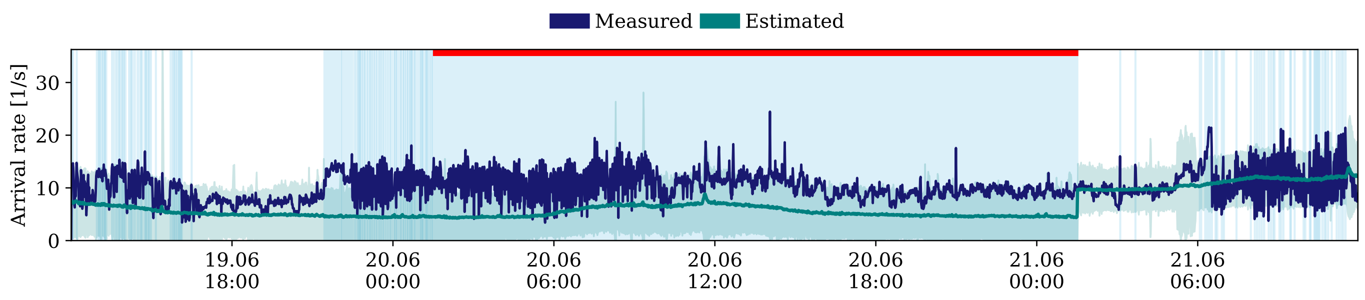


Fig. 4: Measured and estimated arrival rate over time. Traces are taken from simulation data with 24-hour data collection period and 70% utilization threshold. Background shades indicate burst modes. After spending 3 hours in the burst mode, the period marked with red indicates the panic mode. After data collection ends, the estimates are updated.

dynamically, e.g., based on the performance or convergence criteria of the model.

In this paper, we explore the fixed approaches where data collection (and Panic mode) stop after a given time duration.

When the new model is available, HyPA leaves the Panic mode and resumes normal operation. The parameters of the reactive component are reverted to their normal values and the proactive component dominates the scaling of the replicas.

V. Evaluation

In this section, we evaluate HyPA and compare it against a range of baselines. All presented results are obtained from the discrete event-based simulator presented in [17]. It combines data-driven models as well as white-box re-implementations of k8s' components.

A. Settings

1) Simulated Cluster: The cluster contains a single node with 64 CPU cores. Each pod occupies one CPU core and the simulator assumes resource limits per pod with isolated CPUs so that there is no interference between the pods on the CPU. Thus, a maximum of 64 pods can be allocated.[1] Requests arrive at a load balancer that distributes them across the available pods according to a round-robin policy. Request processing on the pods happens in a first-in-first-out manner for a given duration. The simulator resembles the metric collection pipeline from k8s and the collected data is available to all considered HPAs.

2) Metrics: HyPA trades off different objectives which serve as metrics for the comparison:

- **Request completion time (RCT):** RCT is defined as the time difference between the completion and the arrival time of the request in the system. Operators are usually interested in the 99th (or higher) percentiles of the RCT.
- **Total pod seconds:** This metric directly relates to the cost of the deployment and is often used for billing in managed k8s environments[2]. We consider the integral of the number of CPUs allocated for pods over time. Since in our case, each pod requests exactly one CPU, we refer to it as the "total pod seconds".

3) Algorithms: We compare three variants of HPAs. The candidates are described in the following:

- **HyPA (H)** combines proactive and reactive component. The proactive component runs every $t_p = 60\text{s}$. The reactive one runs every $t_r = 180\text{s}$ and every $t'_r = 15\text{s}$ in Burst mode. The target utilization is the same in both modes. We select these values to ensure the best RCT in a variety of scenarios, and we use a fixed **(F)** data collection duration of 24 hours.
- **Proactive (P)** uses only the proactive component as described in Section IV-A2. It synchronizes every $t_p = 60\text{s}$.
- **Reactive (R)** uses only the reactive component following Equation 1. We vary the synchronization period t_r among 60s, 180s.

4) Application and request pattern: We compare the algorithms on request arrivals from four continuous time periods from the Bibsonomy trace. The request arrivals follow the trace, i.e., use the provided timestamps. The inputs are shown in Figure 5. The duration of the periods varies between 48 and 120h and each of the selected periods has distinct characteristics such as high bursts or distributional shifts/drifts.

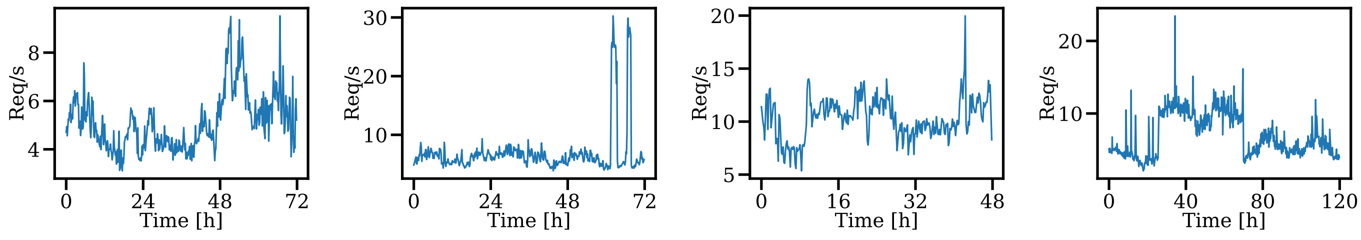
The evaluation considers a single request type with a fixed processing time. As the average arrival rate of the BibSonomy trace is low ($\approx 4-8$ requests per second), we use a processing time of 500ms to induce a substantial load on the system.

B. Temporal analysis of HyPA

Figure 4 shows the measured (dark blue line) and estimated (dark green line) arrival rate over a time over a period of 2 days for illustrating the behavior of HyPA with respect to changes in the demand patterns. The estimator's certainty bounds are shown with the green shaded area. Burst periods are also indicated with the shades in the background, and the period highlighted by the red bar at the top illustrates the Panic mode. HyPA continuously checks the performance of the proactive component by checking if the estimated $n_{req}(t)$ is within the pre-defined bounds (green shaded area). At the

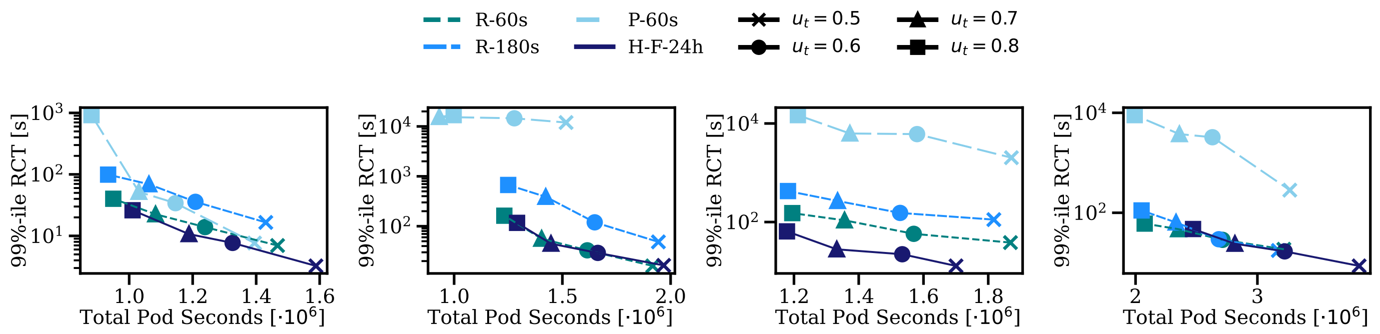
[1]Note that due to the strict isolation between pods, no significant change in the observed results is expected when evaluating in a multi-node cluster or increased number of CPU cores per pod.

[2]For instance, https://cloud.google.com/kubernetes-engine/pricing.



(a) Period 1 (b) Period 2 (c) Period 3 (d) Period 4

Fig. 5: Request arrival rate over time for the four considered input traces. All traces contain some deviation from a regular periodic (diurnal) pattern, e.g., a burst or a (temporary) shift of the mean value.



(a) Period 1 (b) Period 2 (c) Period 3 (d) Period 4

Fig. 6: Pareto-plot of the two performance metrics. Colors and line styles indicate the algorithm configurations. **R** stands for the *reactive*, **P** stands for *proactive*, **H** stands for *hybrid* HPA, and **F-24h** indicates a fixed data collection duration of 24 hours, whereas the time periods indicate the time intervals between consecutive synchronization periods in the legend. The markers show the desired average pod utilization. For two of the shown traces, HYPA outperforms the purely reactive and proactive solutions. For the other two traces, HYPA performs similarly as a purely reactive solution but trades off the two metrics differently (the line is shifted).

beginning of the plot, it can be seen that HYPA enters Burst mode. However, as the bursts are not persistent, it falls back to the normal operation mode.

When HYPA observes a frequent occurrence of bursts within a 6-hour window, HYPA recognizes that the bursts are persistent, enters into Panic mode, and starts data collection. This is also outlined by the area designated with red color. For this example, data collection continues for 24 hours, and at the end of the period, the model is updated. After the model update, the estimations capture the distributional shift in the demand patterns. Overall, this illustrates how the HYPA behaves with respect to changes in request patterns.

C. Comparison of HPAs

We start by analyzing the two metrics for the different time periods on an aggregated level. Figure 6 shows Pareto plots of the two metrics and several values of u_t (different markers) and all the algorithms. By varying the utilization threshold u_t, we aim to find the best configuration per algorithm. For both metrics, smaller values are preferred, i.e., markers and curves closer to the lower left corner represent better performance.

For Period 1 (Figure 6a), there are clear performance differences between classes of the algorithms. P-60s performs worst, followed by R-60s and all variants of HYPA, which overlap in this figure. Reducing u_t results in a lower 99%-ile of the RCT but increases the total pod seconds. This is intuitive since scale-out happens earlier. In principle, R-60s can achieve similar RCTs like HYPA by using a lower u_t (e.g., the markers for R-60s($u_t = 0.7$) and H-F-24h($u_t = 0.8$) are at $\approx$ 30s). However, this comes at the cost of increased total pod seconds ($\approx 10\%$). This behavior continues similarly for lower u_t.

The observations for Period 2 (Figure 6b) are similar but the performance gap between R-60s and HYPA is smaller, i.e. the two variants behave almost the same. HYPA has a slightly smaller RCT but higher total pod seconds. Only for $u_t = 0.5$, R-60s is slightly better than HYPA. Considering the input pattern (Figure 5b), the arrival rate is almost constant except for two bursts towards the end of the trace. Here, HYPA enters the burst mode (but not panic mode) and essentially behaves like the reactive HPA.

More differences are visible for Periods 3 and 4 (Figure 6c and 6d). For Period 3, we again observe that HYPA outperforms the pure variants. Here, there are the largest gains for HYPA. Lastly, for Period 4, the results are closer again. However, closer inspection reveals, that HYPA explores a different trade-off of RCT and total pod seconds compared to R-60s. The minimal achieved 99%-ile RCT is lower than for R-60s, however at significantly higher total pod seconds.

In conclusion, HyPA performs similarly or better than pure reactive or proactive solutions on a variety of input patterns. In particular, in the presence of mean shifts in the arrival pattern, we observe that HyPA can adjust the trade-off between RCT and deployment cost to outperform the vanilla HPA approaches.

VI. Conclusion

Efficient autoscaling is an important aspect of cluster operation. Proactive approaches that forecast the demand have shown benefits over purely reactive approaches that operate on system utilization. However, the former fall short in case of changes or shifts in the underlying demand patterns or distributions. Hybrid solutions have emerged to improve upon this. In this paper, we presented HyPA, a hybrid HPA that falls back to a reactive approach when the error between the predicted and the measured request arrivals increases. Moreover, if the mismatch persists for a longer time period, HyPA automatically updates its model to utilize again the proactive approach. Our evaluation demonstrates that HyPA outperforms previous approaches that are purely reactive or proactive.

This paper presents only an initial assessment of HyPA. Improving demand forecasting models and evaluations of HyPA in a physical testbed environment are possible avenues for future work.

References

[1] O-RAN Alliance, "O-RAN: Towards an Open and Smart RAN," O-RAN Alliance, Alfter, Germany, Tech. Rep., October 2018. [Online]. Available: https://www.o-ran.org/resources

[2] S. Singh and I. Chana, "Cloud resource provisioning: survey, status and future research directions," *Knowledge and Information Systems*, vol. 49, no. 3, pp. 1005–1069, Feb. 2016. [Online]. Available: https://doi.org/10.1007/s10115-016-0922-3

[3] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey," *ACM Comput. Surv.*, vol. 51, no. 4, jul 2018. [Online]. Available: https://doi.org/10.1145/3148149

[4] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch, "Autoscale: Dynamic, robust capacity management for multi-tier data centers," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, no. 4, pp. 1–26, 2012.

[5] M. Imdoukh, I. Ahmad, and M. G. Alfailakawi, "Machine learning-based auto-scaling for containerized applications," *Neural Computing and Applications*, vol. 32, no. 13, pp. 9745–9760, Jul. 2020. [Online]. Available: https://doi.org/10.1007/s00521-019-04507-z

[6] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, "Machine learning-based scaling management for kubernetes edge clusters," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 958–972, 2021.

[7] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev, "Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 800–813, 2018.

[8] A. Feldmann, O. Gasser, F. Lichtblau, E. Pujol, I. Poese, C. Dietzel, D. Wagner, M. Wichtlhuber, J. Tapiador, N. Vallina-Rodriguez, O. Hohlfeld, and G. Smaragdakis, "The lockdown effect: Implications of the covid-19 pandemic on internet traffic," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–18. [Online]. Available: https://doi.org/10.1145/3419394.3423658

[9] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Adaptive, Model-driven Autoscaling for Cloud Applications," in *11th International Conference on Autonomic Computing (ICAC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 57–64. [Online]. Available: https://www.usenix.org/conference/icac14/technical-sessions/presentation/gandhi

[10] D.-H. LUONG, H.-T. THIEU, A. OUTTAGARTS, and Y. GHAMRI-DOUDANE, "Predictive autoscaling orchestration for cloud-native telecom microservices," in *2018 IEEE 5G World Forum (5GWF)*, 2018, pp. 153–158.

[11] M. Abdullah, W. Iqbal, J. L. Berral, J. Polo, and D. Carrera, "Burst-Aware Predictive Autoscaling for Containerized Microservices," *IEEE Trans. Serv. Comput.*, vol. 15, no. 3, pp. 1448–1460, 2022. [Online]. Available: https://doi.org/10.1109/TSC.2020.2995937

[12] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *2012 IEEE Network Operations and Management Symposium, NOMS 2012, Maui, HI, USA, April 16-20, 2012*, F. D. Turck, L. P. Gaspary, and D. Medhi, Eds. IEEE, 2012, pp. 204–212. [Online]. Available: https://doi.org/10.1109/NOMS.2012.6211900

[13] A. Bauer, V. Lesch, L. Versluis, A. Ilyushkin, N. Herbst, and S. Kounev, "Chamulteon: Coordinated auto-scaling of micro-services," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 2015–2025.

[14] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A fast and reliable software network load balancer," in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, ser. NSDI'16. USA: USENIX Association, 2016, p. 523–535.

[15] T. Barbette, C. Tang, H. Yao, D. Kostić, G. Q. M. Jr., P. Papadimitratos, and M. Chiesa, "A High-Speed Load-Balancer design with guaranteed Per-Connection-Consistency," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 667–683. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/barbette

[16] D. Benz, A. Hotho, R. Jäschke, B. Krause, F. Mitzlaff, C. Schmitz, and G. Stumme, "The social bookmark and publication management system BibSonomy," *The VLDB Journal*, vol. 19, no. 6, pp. 849–875, Dec. 2010. [Online]. Available: http://www.kde.cs.uni-kassel.de/pub/pdf/benz2010social.pdf

[17] J. Zerwas, P. Krämer, R.-M. Ursu, N. Asadi, P. Rodgers, L. Wong, and W. Kellerer, "Kapetánios: Automated kubernetes adaptation through a digital twin," in *2022 13th International Conference on Network of the Future (NoF)*, 2022, pp. 1–3.