Theta-Scan: Leveraging Behavior-Driven Forecasting for Vertical Auto-Scaling in Container Cloud

Josep Lluis Berral[†], David Buchaca[†], Claudia Herron[‡]
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya[†], Universitat Pompeu Fabra[‡]
{josep.berral, david.buchaca, claudia.herron}@bsc.es

Chen Wang, Alaa Youssef
IBM Research
Yorktown Heights, NY
Chen.Wang1@ibm.com, asyousse@us.ibm.com

Abstract—**Detection of behavior patterns on resource usage in containerized Cloud applications is necessary for proper resource provisioning. Applications can use CPU/Memory with repetitive patterns, following a trend over time independently. By identifying such patterns, resource forecasting models can be fit better, reducing over/under-provisioning via fewer resizing operations. Here we present ThetaScan, a time-series analysis method for vertical auto-scaling of containers in the Cloud, based on the detection of stationarity/trending and periodicity on resource consumption. Our method leverages the Theta Forecaster algorithm with deseasonalization that, in our provisioning scenario, only requires the estimated periodicity for resource consumption as principal hyper-parameter. Commonly used behavior detection methods require manual hyper-parameter tuning, making them infeasible for automation. Besides, it can be used at multi-scales (minute/hour/day), detecting hourly and daily patterns to improve resource usage prediction. Experiments show that we can detect behaviors in resource consumption that common methods miss, without requiring extensive manual tuning. We can reduce the resizing triggers compared to fixed-size scheduling around $\sim 10\% - 15\%$, reduce over-provisioning of CPU and Memory through periodic-based provisioning. Also a $\sim 60\%$ on multi-scale resource forecasting for traces showing periodicity at different levels in respect to single-scale.**

Index Terms—**Cloud Native; Machine Learning; Container; Auto-scaling; Stationarity Detection; Periodicity Detection; Time series Forecasting; Theta Forecaster**

1. Introduction

Understanding resource usage behaviors for better resource usage prediction is critical for proactive resource provisioning. Resource requirement from services and user-depending applications might change along time, and such changes might be gradual or sudden, making easier or harder to predict such consumption behaviors. Optimizing container resource allocation (e.g., CPU, Memory) requires accurate usage forecasting methods, which may vary for workloads with different usage behaviors, to prevent CPU throttling and container Out-Of-Memory evictions while avoiding over-allocating resources.

Numerous studies have focused on resource prediction using statistical modeling and learning, but automating the detection of stationary, trending or periodic patterns is still a challenge. Even more when detection must be combined with resource usage forecasting methods to auto-scale containers in large-scale Cloud environments, as current methods require hyper-parameter tuning. Such reliance on manual tuning impedes its applicability to auto-scaling, requiring tuning for applications with different resource usage behaviors. Also, usage patterns can occur at different time-scales. Fast-changing patterns are hard to catch at a large scale, or gradual long-term ones at a small scale, requiring detection at multiple scales.

In this paper, we present Theta-Scan, a method to detect and forecast behavior patterns on CPU/Memory demand in containerized applications at different time scales for container vertical autoscaling. Traditional forecasting methods usually show unsatisfactory accuracy on containers with periodic patterns, as they require an appropriate time window for forecasting, which is hard to know before detecting their cycle. Our method is based on the deseasonalized Theta Forecaster algorithm, a time-series method that only requires the observation window and the expected periodicity as parameters for a prediction. Theta-Scan is in charge of finding the cycle from the observed window, indicating stationarity/trending/periodic patterns, and obtaining a forecaster model without additional computational cost.

The contributions of this work are the following:

- We propose Theta-Scan, an online detection method to discover stationarity, trending, and periodic behaviors on resource consumption in containerized applications.
- We implement a resource provisioning method based on behavior-driven resource usage forecasting, particularly effective for periodic patterns, with a multi-time-scale approach (minute/hour/day).
- We employ a dynamic time window for auto-scaling to reduce container resizing operations, using the detected periodicity as reference.
- We compare our method to detect patterns against existing ones, revealing the obstacles of each one towards the required automation for auto-scaling.

The method is evaluated on the traces of containerized applications from the IBM Cloud Services for CPU and Memory consumption. Our method is agnostic of the type of resource, thus applicable to other resource types such as network or I/O utilization. Experimentation results show that our methodology identifies stationary and periodic traces successfully as classic methods do, which requires additional manual tuning not to miss specific periodic patterns. Also, periodicity detection helps to reduce the number of forecasting and resizing triggers by around 10%-15% on containers where frequent resizing is highly needed ($\leq 1hour$). Finally, through multi-scale forecasting, we can reduce the under-provisioning by $\sim 60\%$ compared to provisioning using single time-scale forecast models.

2. Related Work

Time series pattern detection has been an active area of research during the last decade. St-Onge et al. [1] propose an offline workload periodicity detection algorithm, based on prefix transposition, capable of identifying periodic cycles in the CPU and throughput workload when the lengths and amplitude of the patterns are dynamic. Elfeky et al. [2] show a convolution-based algorithm focusing on *segment periodicity* and *symbol periodicity*, while considering the whole time-series. Furthermore, Rasheed et al. [3] presented a suffix tree-based noise resilient algorithm that detects all the periodicity types and also multiple symbol patterns. These methods focused on identifying periodic patterns while our work, apart from detecting the potential period, uses it to forecast future resource usage. In addition, they consider discrete-value time-series while we assume continue-valued time-series, dealing with a large number of potential inputs. Also, Zhang et al. [4] developed a pattern-sensitive resource scheme that chooses the most suitable prediction method based on the detected pattern by using the Fast-Fourier Transform (FFT). However, FFT does not always succeed in detecting the optimal period.

About resource autoscaling in the Cloud using ML, Google's AutoPilot [5] is a state-of-art autoscaling method that estimates resource demands from statistics in the previous time window, showing good accuracy on stationary resource usage. However, it still fails to proactively adjust the provision for sudden changes of CPU/Memory demands. Other works propose load predictors based on the request arrivals, including regression and ARMA/ARIMA methods, i.e., Roy et al. [6], Yang et al. [7] and Calheiros [8]. These methods are well known for time-series, including trending and seasonality. However, such models do not fit well with irregular behaviors and sudden pattern changes. Finally, works like Berral, Buchaca et al. [9] [10] propose models to learn and predict higher-abstraction patterns like "phases" on resource consumption. Our work focuses on identifying stationary, trending, and recurring behaviors directly from resource usage in an online fashion without fully characterizing applications.

Our presented method leverages a time-series model, the Theta Forecaster introduced in [11] and [12], for automatic stationarity, trending, and periodicity detection, and resource demand forecasting. The Theta Forecaster implementation here used is a Simplification of the Exponential Smoothing [13] (aka Holt-Winters or SES) with detrending and deseasonalization. Having as input the time-series and a potential period, the time-series is 1) deseasonalized by removing the average "period" pattern along the series, 2) detrended by performing a period-by-period linear-regression, and removing the trend at each period, and 3) fitted a SES to learn the drift of the series. The model stores the SES, the average period pattern and the series trend. The SES is used then to forecast the drift for next steps, then the trend and period pattern are introduced again, returning in result the forecast for that series.

3. Methodology

This paper presents Theta-Scan, a forecasting technique focused on predicting resource usage behavior on containerized applications, where resource usage exhibits periodic patterns to provision such resources proactively. Here, high accuracy on forecasting is not as crucial as upper/lower-bound prediction on resources, usually depending on hard-to-capture cyclical trends. Our method goes through three steps: detecting time-series behavior, forecasting based on detected behaviors and refining the prediction with behavior-driven multi-scale forecasting.

3.1. Behavior Detection

When managing a resource (CPU/Mem/...), we are interested in its predictability towards the future. We consider a time series "periodic" when it has a recurring pattern repeated at least twice. We consider a time-series "stationary" when its statistics do not change much over time, and we consider a time series "trending" when its first-order difference is constant or the time series increases with a defined slope (also with some noisy variation). Such differences in behaviors will determine how we forecast container consumption towards auto-scaling.

Periodicity Detection. We want to detect the recurring resource usage patterns on containers, then forecast such usage for the next period rather than a predefined time window to finally adjust their provisioning. To identify such behavior, we look for periodicity in resource consumption. Many methods for periodicity detection require either human supervision, interpretation, or none-fit-all hyper-parameter tuning, preventing them from full automation.

Here we introduce the Theta-Scan Algorithm (THS), whose objective is to determine the $period$ of an observed time window, based on the Theta Forecaster (deseasonalized SES) that only requires the expected $period$ as input. Considering an observed $window$ with the resource consumption $[X_{T-N} \dots X_T]$ where $N = length(window)$, such window is split into training/test sets: the training set is the first $N - n_{test}$ values and the test set is the last n_{test} values. Given the training segment and a potential period, the Theta Forecaster will attempt to forecast the test segment. To find

Algorithm 1: The Theta-Scan algorithm

```
best_model, best_period, errors_list = $null$;
best_error = $\infty$;
for $i$ in $2 \ldots \lfloor n_{train}/2 \rfloor$ do
    model = fit (data = training_set, period = i);
    results = predict (model = model, steps = $n_{test}$);
    error = evaluate (results, test_set);
    if $error < best\_error$ then
        best_model, best_period = model, i;
        best_error = error;
    end
    errors_list = errors_list $\cup$ error;
end
return (best_model, best_period, errors_list);
```

the period, we scan a range of potential periods, i.e., from $2 \ldots \lfloor n_{train}/2 \rfloor$, and that one achieving the lowest Mean Absolute Error (MAE) is considered the expected period. Algorithm 1 shows the algorithm for Theta-Scan, considering that $fit()$ fits a Theta Forecaster, $predict()$ forecasts the indicated n steps from the last training point in the model, and $evaluate()$ returns the MAE. As long as the training set is bigger than two times of (and the testing set equals to) the actual periodic cycle, the algorithm should be able to detect the period. Once we detect the periodicity, we can forecast resource usage using that period as the forecasting window. Also, we can use the fitted model in Theta-Scan "for free", while other preferred time-series forecasters could be used fitting them with the detected period as forecasting window.

Stationary Detection. Containers showing periodic patterns with maximum and minimum values of resources not far from the mean can be treated as stationary because no container resizing actions are needed. A commonly used test for stationarity is the Augmented Dickey-Fuller test (ADF). ADF test detects series with periodic patterns as non-stationary and returns the p–$value$ indicating the likelihood of being stationary. However, when testing this method on CPU/Memory traces, it shows a high False Negatives as it gets fooled by fluctuations or noises. Nevertheless, the ADF test can be used as a reference to evaluate our method.

When using Theta-Scan, we expect a stationary trace will not have the best period to be found, as the variance of errors for different scanned periods should be very low. The ThetaScan method returns the Best Model to be used for forecasting, the corresponding Best Period for the dynamic window, and a vector of obtained errors for each candidate period. When the error variance is below a certain threshold (e.g., experimentally $< 10^{-5}$), we consider stationarity. Notice that non-stationarity does not imply periodicity. By selecting a history window to fit a model, THS can select the best fitting window rather than a randomly chosen one.

Trending Detection. For trending detection in time series, as the Theta Forecaster model already fits the slope for the next window, it provides information of whether the trace will grow on the next step. There is no need to apply additional trending detection methods.

Algorithm 2: Behavior-driven forecast algorithm

```
series: resource consumption time-series
default_step: minimum resize time-span
n: observation window size
t = n;
while $t <$ len(series) do
    w = series $[t - n \ldots t]$;
    $\langle$b_model, b_period, errors$\rangle$ = ThetaScan (w);
    stationary = var(errors) $>$ threshold;
    if not stationary then
        step = b_period;
        model = b_model;
    else
        step = default_step;
        model = fit (data = window, period = step);
    end
    forecast = predict(model = model, steps = step);
    t = t + step;
end
```

3.2. Behavior-Driven Forecasting

CPU and memory resource usage is monitored for a containerized application, , producing a time series of the current application behavior. We attempt to detect periodic behavior before forecasting to prevent frequent model refittings and unnecessary triggers of container resizing when a pattern is recurring. Moreover, the periodicity detection helps us better fit the Theta forecaster. Algorithm 2 describes the procedure to fit a theta forecaster based on scanning over different period values. Notice that we do not recompute the moving window $w = series[t - n \ldots t]$ at every time step. We want to avoid the high cost of frequent model fitting when there is periodicity detected in the time series, and the selected best Theta model can accurately predict elements in the next time window w. Figure 1 shows the workflow of the ThetaScan given a resource consumption trace.

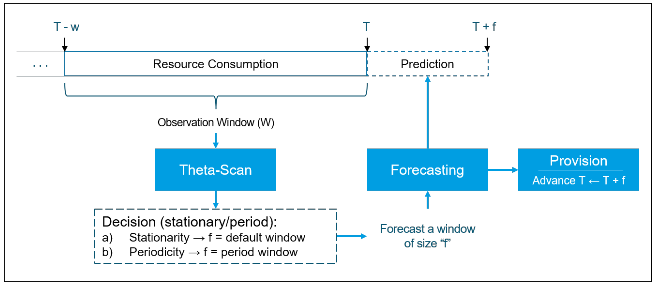
T - w
T
T + f
. . .
Resource Consumption
Prediction
Observation Window (W)
Theta-Scan
Forecasting
Provision
Advance T ← T + f
Decision (stationary/period):
a) Stationarity → f = default window
b) Periodicity → f = period window
Forecast a window of size "f"

Figure 1: Diagram of the ThetaScan Algorithm

Note that the previous algorithm has two hyper-parameters, the default step referring to the minimum time interval for one resizing operation and the observation window size. In our environment, application containers run for minutes, hours, days, and weeks.

3.3. Multi-scale Forecasting

Applications might contain periodic behaviors at different time scales, and processing the observed metrics at different scales allows us to perform the periodicity and stationarity detection at varied time scales. We aggregate the elements in w by their maximum value. This decision favors over-provisioning against under-provisioning to avoid Out-Of-Memory evictions and CPU Throttling. Note that aggregating with the maximum implies that higher time-scales will tend to be more generous on resource providing, and lower ones will under-provision more often. The main challenge here is to choose the best time scale for prediction.

We adapt our predictions at different time scales with a simple rule-based mechanism. We use different forecasters at each time scale, f_m for minutes, f_{30m} for half hours, f_h for hours and f_d for days. Each forecaster makes predictions (one per time-scale), and the best one predicting the last values in W is selected. When predicting series at different time scales, lower-scale forecasters are more accurate than higher-scale forecasters when fitting the same model. However, making container resizing decisions based on predictions at a small time scale might introduce too many resizing operations, which is costly and unnecessary.

4. Experiments

The following experiments evaluate our method, first by comparing it with other used methods for stationarity and periodicity detection, and second by evaluating its capacity to forecast and allocate resources against default methods. The used workload are real traces from the IBM Cloud Services, from ~300 executions of hours to weeks of length, with a metric sampling rate of 15 seconds. For some experiments, we have also manually generated some traces to produce extreme scenarios (e.g., overlapping of periodic patterns at different time scales) towards testing the method with non-trivial behaviors.

4.1. Evaluation Metric

Evaluating a forecasting method using traditional error metrics (MAE, SME, etc.) works for scenarios favoring the average usage. But here we want to assesses the over/under-provisioning trade-off through two metrics: the percentage of time the resource manager has provisioned a container above/under its demand and the accumulated resources over/under-provisioned over time, shown in Equation 1.

Container provisioning requires allocating at least the required resources needed while minimizing over-provisioning. Under-provisioning a container produces CPU throttling (degrading application progress) and Out-Of-Memory errors (OOM, killing the container). Over-provisioning resources waste unused resources that other containers could use. Note that applications with sudden bursts might become unpredictable unless there is periodicity detected, and resource provisioning methods will always fail to provision without prohibitive high margins. We consider these cases as outliers in our scenario.

$$\begin{aligned} over\_prov &= \sum_t max(Res_t^{prov} - Res_t^{req}, 0) \\ under\_prov &= \sum_t max(Res_t^{req} - Res_t^{prov}, 0) \end{aligned} \tag{1}$$

4.2. Evaluation of Behavior Detection Methods

Stationarity Detection. Here we compare the classic Augmented Dickey–Fuller (ADF) method with our stationarity detection method. We use both methods over CPU/MEM traces and compute the false negatives/positives observed. Figure 2 shows the agreement between both methods, where both methods agree in half of the series. Since we do not have the ground truth of the "periodic" behaviors, we manually inspected traces where both methods disagree on the stationarity behavior. We observed that ADF-Stat/THS-NonStat traces, ADF considers the average trend stationary and can not detect long periodic patterns. From those traces, in more than 75% the CPU is constant or with Memory variation lower than $100MB$, low enough to be ignored for autoscaling. Also, those traces considered ADF-NonStat/THS-Stat have a slight slope or are step-wise traces, displaying long stationary steps.

	THS (CPU)	
ADF↓	Stat	NoStat
Stat	102	68
NoStat	31	61

	THS (MEM)	
ADF↓	Stat	NoStat
Stat	83	11
NoStat	134	34

Figure 2: Confusion matrices for stationary detection of CPU and Memory, in number of analyzed containers

Periodicity Detection. We have evaluated our periodicity detection method against the classical Auto-Correlation (AC) [14]. Such a method shows the similarity between the original time series and a delayed time series. Figure 3 compares the periods detected (with a maximum of 500 steps) using AC against our method, and we observed that most of the periods found by AC are longer than the given maximum. To be specific, in many examples, AC considers as period the entire length of the trace. For THS, detected periods are distributed across the different ranges. We have also experimented with Fast Fourier Transform (FFT) [15] to find periodicity, using high FFT frequency values as candidates of periodicity, and we found similar behaviors as with AC. Again, the best candidates given by FFT are usually the length of the whole time series.

4.3. Evaluation of Prediction for Resource Demand

Next, we evaluate different forecasting methods for container resource autoscaling by measuring the over/under-provisioning. We choose an observation window always at a larger time scale than the forecasting window, e.g., if we forecast a minute, the observation window is an hour, and for hourly forecasting, the observation window is a day.

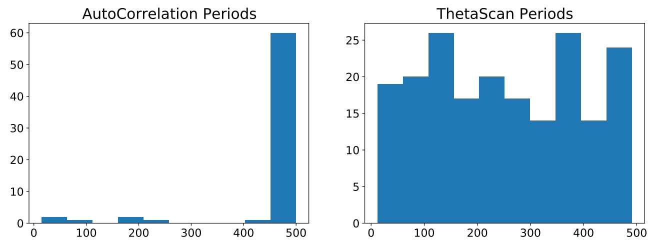

Figure 3: Histogram comparison of periods found in the data, for 300 analyzed containers

As default-step for forecasting, we choose $1/6^{th}$ of the observation window W. THS can find periods at a maximum $1/3^{rd}$ of W, and we select $1/2$ of that length to minimize the risks of missing periodicity.

Dynamic vs. fixed forecasting window. We compare the period-based dynamic size forecasting window against always using a default-sized window, fixing the default size as the minimum window. Figure 4 shows how the dynamic window triggers fewer resizing operations for short time scales than the fixed-window policy. Also, over/under-provisioning remains similar for both methods, considering results "good enough". For 30-minutes and 1-hour periods, we reduce the number of CPU resizing between 10% and 15%; while we increase an 18% in 3 to 6-hour periods, that in such time-scale it means only 1 or 2 additional resizing per application. While for Memory, we reduce them between 10% and 13% in 30-minute and 1-hour periods and increase them a 3% for 3 to 6-hour periods.

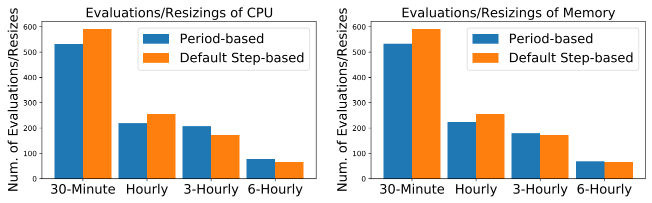

Figure 4: Number of times we evaluate and resize container, guided by period-detection vs. the default time-step

Comparison of forecasting methods. Here we compare the THS to popular forecasting methods to avoid fitting complex and costly models repeatedly. These methods include time-series K-NN (KN), Linear Regression (LR), Naïve prediction (NV), and a simple heuristic (denoted as Max) that returns the maximum usage in the previous window as the forecasting value. Under and over-provisioning are used as quality metrics, and it is more important to avoid under-provisioning than over-provisioning. Other methods like SARIMA are compared. Although they perform similar to THS, they show high complexity and difficulty for automation and execution times $\times 30$ compared to THS.

Figure 5 shows the comparison of the different methods. Results are similar for the TH, KN, and LR, and accumulated under-provision is in acceptable limits compared to over-provisioning amounts, even more in hourly resizing. Please note the 1:1000 scale difference on the figure, where

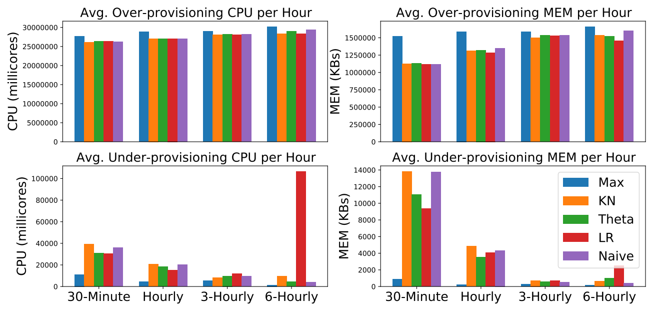

Figure 5: Accumulated Over/Under-provision obtained from using different forecasting methods (on regular workloads)

the over-provisioning reduction in memory is $\times 100$ the under-provisioning gain in the worst scenario (30-minute). For some specific workloads, i.e., step-wise ones, we observe some differences between methods. THS and LR tend to over-provision on higher time scales, unlike Max, KN, and NV. From these results, it is preferred to use the previous window maximum in forecasting-based provisioning.

4.4. Evaluations of Multi-Scale Behavior Methods

We evaluate the method at multiple time scales to detect different periodic patterns, e.g., if periodicity occurs both hourly and daily. For such test, we took traces with a known ∼11-minute period and added a four-hour periodic pattern on top, then performed periodicity detection with two forecasters f_m and f_{30m} fitted with a minute and half-hour scales Detection at minute-scale finds a mean period $\mu = 12.9$ and standard deviation $\sigma = 8.9$ for an actual 11-minute period, observing that some detections correspond to period $\times 2$ (22-minute periods are detected). Detection at 30-minute-scale finds a mean period $\mu = 7.8$ (3.9 hours) and a standard deviation $\sigma = 0.7$ (20 minutes) from the 4-hour pattern. Figure 6 shows that in both forecasting scales, the sequences fit by guiding the Theta Forecaster with the detected periods. The third sub-figure shows how higher-scale aggregation using "max" produces a conservative forecast preventing CPU-Throttling and OOM errors (green line) by losing the precision offered by more fine-grained forecasting (orange line). In real scenarios, a per-minute resizing is infeasible, and a larger scale becomes preferable.

4.5. Combined Multi-Scale for Provisioning

Finally, we evaluate the error and over/under-provisioning with different time-scale patterns, predicting using the two different scales vs. combining them. Table 1 shows how smaller time-scales (hourly) adjust more to resources than larger ones (daily), but when combined, the error is slightly reduced compared to single time-scale forecasting. Also, Table 2 shows how, when we combine time-scale forecasts, we reduce under-provisioning between 58% and 64% (for CPU and Memory) compared to the hourly time-scale forecasting while maintaining the

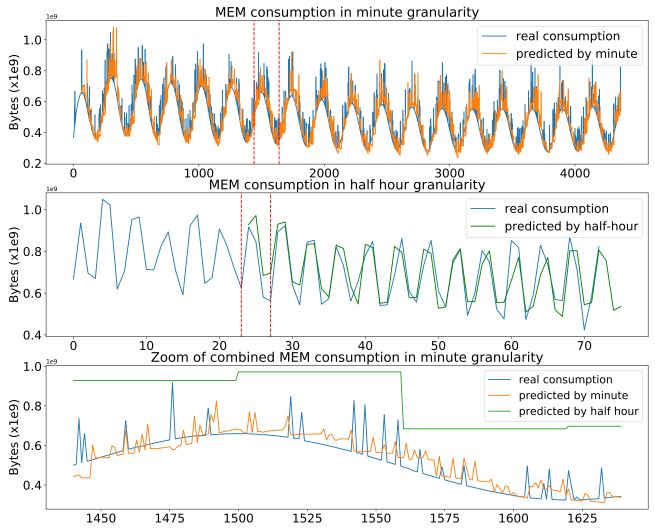

Figure 6: Provisioning at different time-scales (minutes and half-hours), and zoom on the red-marked region.

	Error_Hour	Error_Day	Error_Combined
CPU	0.004354	0.009777	0.003662
Memory	0.006078	0.010314	0.005266

TABLE 1: Relative MAE per Hour on demand vs. prediction, for Hourly and Daily forecasting (average ratio)

	Ov_CPU	Un_CPU	Ov_MEM	Un_MEM
Hourly	0.002910	0.001444	0.004271	0.001807
Daily	0.009562	0.000215	0.010302	0.000012
Combined	0.002892	0.000770	0.004406	0.000860

TABLE 2: Average ratio of Over/Under-provisioning (missed millicores and KBs) at Hourly scale

over-provisioning. Notice that the daily forecasting could reduce even more under-provisioning but by doubling over-provisioning as expected. E.g., Figure 7 shows a zoom of execution with hourly and daily periodicity, provisioning according to our multi-scale policy.

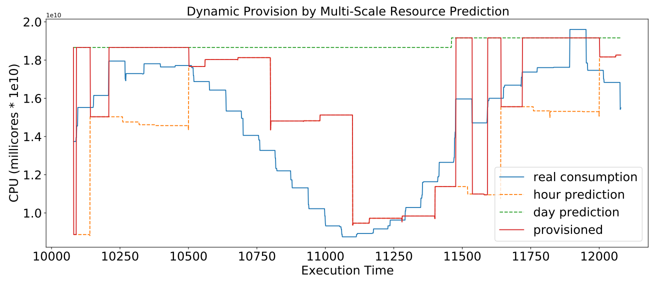

Figure 7: Example of hourly and daily CPU forecasting, selecting the $t-1$ best forecaster for provisioning

5. Conclusion

This paper presented ThetaScan, a methodology for autoscaling of containerized applications. We can auto-scale applications using a dynamic time window instead of a fixed time window for triggering container resizing by detecting periodicity or stationarity/trending. Such an approach reduces the number of resizing without skipping potential patterns in resource consumption. In addition, we detect periodicity and stationarity to forecast resource usage at different time scales, taking seasonality at different scales into account and reduce over and under-provisioning. We evaluated our method against classic behavior detection methods, compared different machine learning methods for forecasting after behavior detection, and evaluated our multi-scale forecasting process showing a reduction of under-provisioning when forecasting at multiple time scales instead of a single scale. The presented multi-scale methodology is applicable to handle the challenging problems of resizing Containers in Cloud scenarios.

Acknowledgments. This work has been partially supported by the Spanish Government (contract PID2019-107255GB) and by Generalitat de Catalunya (contract 2014-SGR-1051).

References

[1] C. St-Onge, N. Kara, O. A. Wahab, C. Edstrom, and Y. Lemieux, "Detection of time series patterns and periodicity of cloud computing workloads," *Future Generation Computer Systems*, vol. 109, 2020.
[2] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, "Periodicity detection in time series databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 7, pp. 875–887, 2005.
[3] F. Rasheed, M. Alshalalfa, R. Alhajj, and A. Member, "Efficient Periodicity Mining in Time Series Databases Using Suffix Trees," vol. 23, no. 1, pp. 79–94, 2011.
[4] F. Zhang, J. Wu, and Z. H. Lu, "PSRPS: A workload pattern sensitive resource provisioning scheme for cloud systems," in *Proceedings - IEEE 10th International Conference on Services Computing, SCC 2013*, pp. 344–351, 2013.
[5] K. Rzadca and et al, "Autopilot: Workload autoscaling at google," in *EuroSys conference*, EuroSys '20, ACM, 2020.
[6] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *IEEE Intl. Conf. on Cloud Computing*, pp. 500–507, 2011.
[7] J. Yang, C. Liu, Y. Shang, Z. Mao, and J. Chen, "Workload predicting-based automatic scaling in service clouds," in *2013 IEEE Sixth International Conference on Cloud Computing*, pp. 810–815, 2013.
[8] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using arima model and its impact on cloud applications' qos," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, 2015.
[9] J. L. Berral, C. Wang, and A. Youssef, "AI4DL: Mining behaviors of deep learning workloads for resource management," in *12th USENIX Workshop HotCloud*, 2020.
[10] D. Buchaca, J. L. Berral, C. Wang, and A. Youssef, "Proactive container auto-scaling for cloud native machine learning services," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pp. 475–479, 2020.
[11] V. Assimakopoulos and K. Nikolopoulos, "The theta model: a decomposition approach to forecasting," *International Journal of Forecasting*, vol. 16, no. 4, pp. 521–530, 2000. The M3- Competition.
[12] R. J. Hyndman and B. Billah, "Unmasking the theta method," *International Journal of Forecasting*, vol. 19, no. 2, pp. 287–290, 2003.
[13] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *International Journal of Forecasting*, vol. 20, no. 1, pp. 5–10, 2004.
[14] M. Vlachos, P. Yu, and V. Castelli, "On periodicity detection and structural periodic similarity," *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005*, 2005.
[15] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, no. 90, p. 297, 1965.