Improving the Decision-Making Process of Self-Adaptive Systems by Accounting for Tactic Volatility

Jeffrey Palmerino, Qi Yu, Travis Desell and Daniel E. Krutz
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology, Rochester, NY, USA
Email: {jrp3143, qyuvks, tjdvse, dxkvse}@rit.edu

***Abstract*—When self-adaptive systems encounter changes within their surrounding environments, they enact *tactics* to perform necessary adaptations. For example, a self-adaptive cloud-based system may have a tactic that initiates additional computing resources when response time thresholds are surpassed, or there may be a tactic to activate a specific security measure when an intrusion is detected. In real-world environments, these tactics frequently experience *tactic volatility* which is variable behavior during the execution of the tactic.**

Unfortunately, current self-adaptive approaches do not account for tactic volatility in their decision-making processes, and merely assume that tactics do not experience volatility. This limitation creates uncertainty in the decision-making process and may adversely impact the system's ability to effectively and efficiently adapt. Additionally, many processes do not properly account for volatility that may effect the system's Service Level Agreement (SLA). This can limit the system's ability to act proactively, especially when utilizing tactics that contain latency.

To address the challenge of sufficiently accounting for tactic volatility, we propose a *Tactic Volatility Aware* (TVA) solution. Using Multiple Regression Analysis (MRA), TVA enables self-adaptive systems to accurately estimate the cost and time required to execute tactics. TVA also utilizes *Autoregressive Integrated Moving Average* (ARIMA) for time series forecasting, allowing the system to proactively maintain specifications.

***Index Terms*—Artificial Intelligence, Self-Adaptation, Machine Learning**

I. Introduction

The world is increasingly relying upon autonomous, self-adaptive systems that have the ability to function independently without human interaction. Examples of these self-adaptive systems include self-driving cars, medical devices, and many common Internet of Things (IoT) devices. Many self-adaptive processes utilize a closed-loop control mechanism that monitors the system's state and its surrounding environment. Furthermore, these mechanisms also determine if the system should be altered to perform any necessary adaptations [27], [5], [42], [11]. These self-adaptive approaches typically rely upon a set of *adaptation tactics* to make necessary changes [35], [43], [30], [20], [19]. Example tactics include the provisioning of an additional virtual machine in a web farm when the workload reaches a specific threshold, or reducing non-essential functionality on an autonomous Unmanned Aerial Vehicle (UAV) when battery levels are low.

Tactics frequently experience latency, which is the amount of time from when a tactic is invoked until its effect on the system is realized [35], [37], [39], [38]. Examples of tactic latency include a cloud-based system requiring more than a minute to update certain firmware nodes [33], [3], or a cyber-physical system requiring one minute to re-activate GPS signals [32], [36]. Tactics also frequently have a cost associated with them, which may be in the form of energy, monetary or other resource costs necessary for execution [26], [44], [25]. Examples of tactic cost include the required energy for moving a physical component in a UAV or the monetary cost of using a remote third-party resource for computational tasks.

Both tactic latency and cost are likely to be first-class concerns in the decision-making process, as they can directly impact if and when a tactic is executed [35], [39], [38]. Improperly accounting for tactic latency can lead to situations where tactics are begun too early or too late, or are not available when needed [35], [37]. Additionally, improperly accounting for tactic cost can result in the selection of a tactic that is more expensive than a tantamount, less costly alternative [25]. Therefore, it is imperative that self-adaptive systems properly account for both the latency and cost of tactics.

Real-world systems will frequently encounter *tactic volatility*, which is any rapid or unpredictable change that exists within the attributes of a tactic. For example, both tactic latency and cost may be highly volatile depending on the system's surrounding environment. A tactic of transmitting data could take longer than expected due to network congestion, or a tactic of moving a physical component in a UAV could be more expensive due to mechanical problems.

Unfortunately, state-of-the-art decision-making processes in self-adaptive systems do not account for tactic volatility. This limitation can be highly problematic, adversely impacting the decision-making process in several ways. For example, a system may execute a tactic too late to be effective if it assumes that the tactic will always take two seconds to conduct, when in reality it has been observed to consistently take longer.

A *Service Level Agreement* (SLA) helps to define system objectives such as keeping a value under a specific threshold, along with rewards and penalties for meeting defined objectives [35]. A challenge for purely reactive processes is that the system may not adhere to objectives defined in the SLA if tactics are expected to experience latency. For example, the SLA for a cloud-based self-adaptive system may define a reward for operating under a response time threshold. However, tactics such as adding servers to reduce response times will likely experience varying levels of latency. To perform optimally, the system should act proactively by beginning these tactics *before* the additional resources are required so that the resources are available when needed [38], [6]. Thus, a mechanism that enables the system to accurately forecast tactic latency and cost values is highly beneficial to the self-adaptive process.

To address the limitations of current self-adaptive processes in properly accounting for tactic volatility, we propose a *Tactic Volatility Aware* (TVA) solution. Through the use of a Multiple Regression Analysis (MRA) model, TVA enables the self-adaptive system to learn from previously experienced tactic volatility and make accurate estimates of how the tactic will behave in the future. TVA also includes an *Autoregressive Integrated Moving Average* (ARIMA) component to enable self-adaptive systems to proactively begin adaptations according to their SLA specifications.

To summarize, this work makes the following contributions:

1) **Problem Demonstration:** Using simulations, we demonstrate that accounting for tactic volatility is essential in self-adaptive systems, especially when the system is known to have unpredictable behavior.
2) **Concept:** To the best of our knowledge, our TVA approach is the first process that accounts for tactic volatility. Existing processes merely consider tactics to have static values, whereas our TVA approach uses run time predictions of latency and cost to handle tactic volatility.
3) **Experiments:** Our experiments demonstrate the positive impact that TVA has on the decision-making process. These experiments are conducted using real-world data, and thus provide additional confidence in our findings.
4) **Tool and Dataset:** Our VolAtiLity EmulaTor (VALET) tool includes data generated from a physical system. This enables the evaluation of our TVA process and provides other researchers with a foundational dataset that they may use to support their own research. This tool and dataset are available on the project website: https://tacticvolatility.github.io/

The rest of the paper is organized as follows. Section II motivates our research using examples of tactic volatility. Section III defines our TVA solution to effectively address tactic volatility. Section IV describes *VALET*, our tactic volatility dataset. Section V describes our systematic experimental evaluation of our proposed process. Section VI describes related works, while Section VII describes threats and future work. Section VIII concludes our work.

II. Problem Definition

Tactic latency volatility and *tactic cost volatility* are the primary motivators for addressing a system's SLA with proactive processes. If a system is purely reactive and does not properly anticipate future system changes in regards to the specifications defined in the SLA, it will be limited by its inability to provide support for proactive tactic implementation.

A. Tactic Cost Volatility

Examples of tactic cost vary widely and are largely domain specific, however tactic cost is frequently a primary concern in the decision-making process [25]. Possible examples of tactic cost include the energy necessary to move a mechanical component in a physical device, the monetary cost to utilize a resource, or the number of computations necessary to perform a tactic. Estimating tactic cost will likely be a first-class concern for the system, especially if there are defined cost thresholds, resource limitations, or if the system merely has the goal of attaining maximum utility at the lowest cost [35]. Unfortunately, despite the cost volatility that many real-world systems are likely to encounter, existing self-adaptive processes that account for cost consider it to be a static value that will encounter no volatility [37], [38], [36]. Accounting for tactic cost volatility is imperative for several reasons:

1) **Cost may be a primary consideration when selecting between multiple tactic options:** When the system has multiple tactic options that it may choose from, the cost of the tactic may be a determining factor when selecting between multiple, otherwise tantamount options.
2) **Determine if the cost exceeds reward:** In selecting which tactic(s) to execute, the system will frequently calculate the expected reward and cost for performing the tactic. If the cost exceeds the reward, then it may not be optimal for the system to execute the tactic.
3) **The estimated cost may impact the system's ability to execute concurrent or subsequent tactics:** The system may possess a finite amount of a resource, with one example of being battery power. Assume that a system has 10 units of battery power remaining. Processes that consider cost to be a static value may define the energy usage of tactic $a = 4$, and for tactic $b = 5$. Therefore the system will assume that it has the ability to execute both tactics concurrently or sequentially. However if the actual energy usage of each tactic is $a = 6$ and $b = 7$, in reality the system will not have the ability to fulfill the amount of energy necessary for the execution of both tactics either concurrently or sequentially.

B. Tactic Latency Volatility

The amount of time required to implement a tactic is known as *tactic latency* [7], which in real-world systems, can be highly volatile. For example, the precise amount of time a system needs to transmit a file across a network may fluctuate due to varying amounts of network traffic. Previous works have shown that latency aware self-adaptive processes offer several advantages over traditional, non-aware techniques [6], [35], [37]. However many state-of-the-art self-adaptive processes still consider tactic latency to be a predetermined and static value. Systems lack the capability to learn to adaptively accommodate for variability in tactic latency. Due to its large impact on the decisions that a system should make, accounting for tactic latency volatility is imperative for several reasons:

1) **Knowing when to begin the execution of a tactic:** A priority for many self-adaptive systems is to ensure that tactics are ready when needed. Therefore, if a tactic has expected latency, then the system will need to proactively begin its execution so that it is available when needed. However, implementing a tactic proactively will typically have additional costs involved, so proactive adaptation must be done with consideration to available resources.
2) **Determine when to augment a slow tactic with a faster tactic:** Accurate tactic latency knowledge is imperative for determining when to augment a slow tactic with a faster tactic. There may be instances when the system decides to utilize a faster, less effective tactic to augment a slower, but more effective tactic [36], [35]. For example, in a self-adaptive system some given tactic a may have higher latency while producing a higher benefit than that of a faster tactic option b. The system may decide to implement both tactic options, knowing that the system could potentially realize the benefits of tactic b while it is waiting for tactic a. When deciding to augment a slower tactic with a faster one, accounting for tactic latency volatility is essential to determine which tactic(s) should be used to augment a slower tactic.
3) **Ensuring the selection of the most appropriate tactic:** When selecting between multiple tactic options, ignoring tactic latency can be problematic. Consider a scenario where the system is deciding whether to use tactic a or tactic b. If tactic b is only slightly better than tactic a in terms of instantaneous utility improvement, then the decision-making process would favor tactic b. However, if tactic a is faster than tactic b, then tactic a would start to accrue utility faster. This means that tactic a may be the most optimal selection [36].

C. Motivating Example

As a motivating example we will use a cloud-based self-adaptive system, based on a similar scenario defined by Moreno *et al.* [35]. This example represents a multi-tier web application that is compromised a web server and database tier. The webserver(s) process a client's request and then access information stored in the database tier. To efficiently provide content while encountering variable workloads, the system can either add/remove servers from the pool or reduce optional content using the 'dimmer' feature. This system has has a goal of maximizing utility while minimizing cost.

The SLA defines the target response time (T) and how utility (U) is calculated. The system incurs penalties if the target response time is not met and accrues rewards for meeting the target average response time against the measurement interval. The average response rate is a, the average response time is r, the maximum request is k and the length of each interval is defined as τ. Provided content is reduced as necessary using a dimmer value (d). Optional content has reward of (R_O), and produces a higher reward than mandatory content (R_M). We will slightly modify the equation to incorporate cost (C), which could be the monetary or energy cost:

$$U = \begin{cases} (\tau a(dR_O + (1-d)R_M)/C & r \leq T \\ (\tau \min(0, a-k)R_O)/C & r > T \end{cases} \tag{1}$$

This system can account for increases in user traffic using two different tactics: (I) reducing the proportion of responses that include the optional content (dimmer), and (II) adding a new server. While reducing optional content will have negligible latency, adding a new server can take several minutes.

Tactic Latency Volatility If the system anticipates that the response threshold will be surpassed in the immediate future, then the system could proactively start the tactic of adding a server to keep the response time under the defined threshold. Overestimating latency could result in scenarios where the system unnecessarily incurs additional cost, as servers would be 'active' longer than necessary. Additionally, if the system determines that it is likely to surpass the defined response time threshold before a new server can be added, then the most appropriate system action may be to use the faster tactic that reduces the amount of optional content while it waits for a new server to be added. Improper tactic latency predictions can lead to situations where the system executes the tactic too soon or too late, or even selects the improper tactic for the encountered scenario. *Accounting for tactic latency volatility is a paramount concern, especially when utilizing a proactive adaptation approach, or when utilizing complementary tactics.*

Tactic Cost Volatility In the provided scenario, it is important to account for cost volatility, especially since not accounting for cost volatility could lead to inaccurate utility calculations. In the event that cost is defined to be higher than what the system is actually encountering in the real-world, then this may lead to scenarios where optional (O) content is shown too infrequently. Conversely, if the cost is defined lower than what is being encountered in the real-world, then could lead to scenarios where optional (O) content is shown too frequently. *A volatility aware solution that enables the system to more accurately predict cost would enable the system to make decisions that lead to more optimal outcomes.*

Reactive Specifications Monitoring If the motivating example was a purely responsive system and did not employ any proactive functionality, it will only determine if the defined response threshold (T) is being surpassed at the current moment. This can be problematic since the tactic of adding additional resources to reduce response time in our example has latency, so the system will need to begin adapting *before* the tactic is actually needed to be complete. Otherwise, the system will incur penalties or not realize rewards while it waits for the tactic to become available. *A process that enables the system to better anticipate how it was going to act in accordance with the SLA would help the system to perform more optimally in dynamic environments.*

III. Proposed TVA Process

Our TVA process consists of the following two phases: (I) Time-series forecasting with *Autoregressive Integrated Moving Average* (ARIMA) and (II) Run-time model generation using Multiple Regression Analysis (MRA); where ARIMA predicts if system specifications in the SLA may be broken in the future and MRA creates run-time models for predicting tactic latency and cost.

A. Autoregressive Integrated Moving Average

The first component of the TVA process is ARIMA, which is a commonly used approach for the prediction of time series data. ARIMA is a generalization of the ARMA (autoregressive moving average) model, which accounts for non-stationary data using differencing. A full ARIMA treatment requires the following notation, *ARIMA(p, d, q)*, where p represents the number of lag observations included in the model, d represents the degree of differencing, and q represents the size, or *order* of the moving average window.

This work uses a Box-Jenkins approach to find the best fit of a ARIMA model for predicting future time series values. This involves applying the following three steps:

1) ***Identification:*** The first step is to determine the order of the ARIMA model, by utilizing *differencing* to transform the potentially non-stationary data to stationary data. Mathematically, differences are shown as $y'_t = y_t - y_{t-1}$, where y_t represents the current observation and y_{t-1} represents the immediate-prior observation. Differencing allows the model to remove any changes in the levels of time series data, thus eliminating trend and seasonality. In some cases, differencing may need to be applied a second time to obtain stationary data. This second order differencing includes subtracting another term $y''_t = y_{t-1} - y_{t-2}$ to the ARIMA model. Our work used a differencing order of $d = 1$, as there was a small amount of non-stationary time series data. Following this, classic autocorrelation and partial autocorrelation plots were used to determine the order of the autoregressive and moving average terms, which resulted in $p = 1$ $q = 0$, respectively.
2) ***Estimation:*** In order to estimate parameters for the Box-Jenkins models, we must apply a solution that can numerically approximate nonlinear equations. As the goal was to minimize a loss or error term, we used the *maximum likelihood estimation* (MLE) method over a nonlinear least squares estimation to determine the model's optimal parameters.
3) ***Model Checking:*** The final step in applying a full treatment of the Box-Jenkins approach is to perform model checking. Since we have the ability to modify orders in the ARIMA model (p and q) it is important that we optimize these parameters. In general, optimization should examine: I) if the model is overfit and II) residual errors. The former is crucial because it effects how generalizable our model is to other time series data, while the latter deals with how well the model performed in terms of predictions. After examining multiple ARIMA models by fine-tuning the three main parameters p, d and q, the final ARIMA model for our work was $ARIMA(1, 1, 0)$. With this notation, our model is known as a *differenced first-order autoregressive model.*

B. Multiple Regression Analysis

The second component of our TVA approach deals with run time predictions of tactic latency and cost. As discussed previously, current self-adaptive processes consider these values to be static attributes. In real-world scenarios, this is an unlikely phenomena because certain events have different outcomes depending on the surrounding environment. For example, a UAV may need to determine the time it will take to transmit a critical file to its base station. This latency could drastically vary depending on the distance of the UAV to the base station, weather conditions and even component functionality. Therefore, the UAV must have a method of modeling its surrounding environment so it can accurately anticipate the length of time required to transmit the file.

In our work we applied this same ideology to predicting tactic latency and cost. There are many different types of regression models, and even machine-learning models that could be used to accomplish this. Initially, we examined a machine-learning approach called Bayesian Ridge Regression (BRR) which through a Bayesian process allows the model to train itself over time as more data becomes available. However, we found that the required feature space for BRR was too large and that the data we collected could not support it. Therefore, our TVA approach uses Multiple Regression Analysis (MRA) to support run time predictions of tactic latency and cost. Consider a classic regression model:

$$y(x, w) = w_0 + \sum_{j=1}^{M-1} w_j x^j = w' x \qquad (2)$$

where $x = (x^1, ..., x^M)'$ with $x^0 = 1$ and x^j being the j-th observation of the independent variable x. Given a set of N training variables $(x_1, ..., x_N)$ along with the observed responses $(t_1, ..., t_N)$, we can solve for the best possible weights of this model through minimizing the error function:

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2 \tag{3}$$

$E(w)$ is a quadratic function of w (the observation weight), which can be conveniently minimized, leading to:

$$w^* = (X'X)^{-1}X't \tag{4}$$

where X is the design matrix whose rows correspond to the observation vectors of the variables and $t = (t_1, ..., t_N)'$ denotes the observed response values of N variables. After minimizing Equation 3, we now have the estimated weights w^* in Equation 4, which can be used by our regression model to predict tactic latency and cost.

It is important to note that regression analysis estimates the conditional expectation of the dependent variable, that is, the average value of the dependent variable when the predictor variables are fixed. Other related methods such as Necessary Condition Analysis (NCA) could be used to estimate the maximum value of the dependent variable, however, we decided that this was out of scope for our approach since estimating maximum values would be considered the *worst-case scenario* for the tactic.

TVA Workflow

TVA combines an ARIMA time series model with an MRA model to improve the self-adaptive process. In doing this, the system is able to properly maintain its defined specifications while also accounting for tactic volatility, leading to better adaptations and better overall performance. As shown in the pseudo code in Algorithm 1, the workflow of TVA is reasonably straightforward. The top level definitions under *procedure* represent the specifications that are being monitored and any calculations that are made. For instance, *spec* represents the current specification that we are analyzing, while *resp* represents the response we obtain from performing the actual analysis. Furthermore, the *loop* of the workflow represents the main component of TVA.

We will next discuss a concrete example that examines the two primary steps of our TVA process as shown in the pseudo code. For this, we will continue to use the self-adaptive hosting service example described in Section II-C.

Step 1: Monitoring Specifications: The first step in our TVA approach is to gather and monitor the specifications defined in the SLA. Using the self-adaptive hosting service example, we will assume that it has been configured to record the response time every six seconds. We will also assume that the system's SLA specification defines that a penalty will be incurred if response times do not stay under a 0.7 second threshold. Unfortunately, the majority of self-adaptive processes act in a reactive manner, so the system would not be able to adapt until after the threshold was surpassed. Through the application of the time series model ARIMA discussed in this section, we will enable the system to act more proactively by allowing it to anticipate future specification values.

Algorithm 1 TVA Workflow

```
1: procedure WORKFLOW
2:     spec ← the specification to analyze
3:     resp ← quantified response from ARIMA analysis
4: loop:
5:     $resp = analyzeSpecification(spec)$
6:     if $resp$ is potentiallyBroken then
7:         $makeLatencyEstimate()$
8:         $makeCostEstimate()$
9:     goto loop
```

Given the specification and parameters required to make decisions, historical data is used to determine parameters to the ARIMA models (Algorithm 1, Line 5), which provide anticipated future values. For example, in the hosting service when the response time specification is analyzed with ARIMA, the "response" is the forecast value. If this forecast value determines that the specification may be broken in the future or is close to being broken, we continue with the rest of our TVA approach.

Step 2: Tactic Latency and Cost Prediction: When the system has determined that it needs to adapt, whether it be because a specification *will* or *may* be broken, our approach then uses regression analysis to predict tactic latency and tactic cost. For example, if our ARIMA analysis from step 1 determines that the hosting service needs to adapt, our TVA approach then predicts tactic latency and cost for all available adaptation tactics. This could entail predicting the latency and cost for a tactic that adds extra computing resources so that response time can be reduced. These predictions are made through a Multiple Regression Analysis (MRA) model. A benefit of focusing on predicting tactic latency and tactic cost is that this can be easily adopted into many existing decision-making processes. For example, one simple option is for the predicted values to merely replace the static, predefined values in the system's decision-making process.

IV. VALET: VolAtiLity EmulaTor Tool

An additional contribution of this work is the development of the VolAtiLity EmulaTor (*VALET*) tool to generate real-world tactic volatility data to enable the evaluation of our TVA process. While existing datasets such as 'The Internet Traffic Archive'[1] are commonly used when evaluating self-adaptive processes [18], [9], these are limited as they do not contain an adequate amount of tactic volatility necessary to evaluate our work. To create a sufficient dataset to demonstrate the benefits of TVA, VALET provides tactic volatility data in the form of latency and cost. The tool and generated dataset is available on the project website: https://tacticvolatility.github.io/

The first action performed in VALET utilizes two physical Raspberry Pi devices, with one acting as the operator and the other as a monitoring instrument. The operator gathers latency data by recording the time required to download an identical 75MB file at one minute intervals, from three different download mirrors around the world. To determine energy usage for this operation, the monitoring instrument collects and records its own energy usage as well as that of the operator. Then, the monitoring Pi calculates the difference between the operating and monitoring Pi's power before and after each download as the overall energy used for the operation. This action provides both real-world latency and cost information. Real-world volatility is also introduced by variations in network speeds that impact the time required to download the file.

The second action performed by VALET obtains an additional form of tactic latency and cost values by performing a 'grep' activity. After the file is downloaded, a simple *grep* command searches files contents for a specific string. The amount of time and energy required to perform this task is recorded.

These operations are performed at one minute intervals over the course of an entire day, creating over 1,400 records of tactic volatility data daily. Figure 1 represents a portion of the time series data gathered from our operations, and also visualizes the volatile latency values that we obtained. As shown, the latency times gathered from Germany were quite volatile, with some spikes in volatility in Massachusetts and Ontario download times.

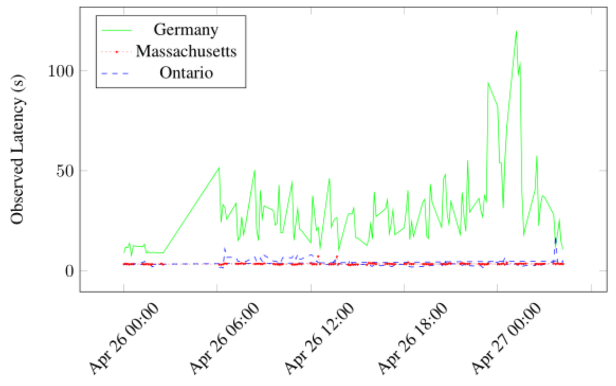

Figure 1. 9-Hour+ Snippet of Latency Data

VALET benefits the software engineering community by enabling developers and researchers to perform evaluations using real-world, time-series data containing tactic volatility. Although VALET represents a specific example of a simple self-adaptive system, the created data is generic enough that it can be used to conduct preliminary evaluations of other machine-learning processes and tactic volatility aware self-adaptive processes.

V. Evaluation

This evaluation addresses the following research questions:

RQ1. *How does not accounting for tactic volatility affect the decision-making process?* Using the statistical tool R, we demonstrate the negative impact of not accounting for tactic latency volatility and tactic cost volatility in the self-adaptive process.

RQ2. *How effective is ARIMA in allowing the system to monitor system specifications?* In our experiments using time series analysis, we demonstrate that time series forecasting with ARIMA helps the system to become more proactive in maintaining specifications defined in the SLA.

RQ3. *How effective is MRA in predicting tactic latency and cost at run time?* In evaluating our multiple regression model, we demonstrate that strong predictive power exists throughout our experiments when estimating tactic latency and cost, even when faced with varying amounts of volatility in the gathered data.

RQ4. *Does using TVA provide substantial improvement to the self-adaptive process over simply using static values for latency and cost?* In comparing our approach against existing self-adaptive approaches, such as those that consider tactics to have static latency and cost values, we demonstrate the substantial improvement to the decision-making process that our TVA process provides.

A. Experimental Data Analysis

To analyze the results found from our VALET experiments, we used the statistical metrics of Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) to evaluate the systematic benefits of our TVA approach. Where y_i is the predicted value and t_i is the actual value at time i, RMSE and MAE are defined as:

$$RMSE = [\sum_{i=1}^{N}(y_i - t_i)^2/N]^{1/2} \tag{5}$$

$$MAE = \frac{\sum_{i=1}^{n}|y_i - t_i|}{n} \tag{6}$$

These metrics allowed us to determine TVA's ability to reduce uncertainty and increase the effectiveness of the decision-making process by examining the prediction accuracy of our time series and machine learning models. RMSE provides a better search landscape for determining model parameters and was used to determine how well our models performed in terms of predictive ability, as larger prediction errors become more pronounced due to the squaring of such errors. On the other hand, MAE was used to examine the absolute value of these error differences. MAE was also used for looking at forecasting errors in time series analysis, which is one of its most common uses [21].

B. *Results*

RQ1: How does not accounting for tactic volatility affect the decision-making process? We first explored RQ1 by performing a proof of concept evaluation using the statistical tool R, where we emulated the negative effects of poor decision-making by a self-adaptive system. We began by defining two tactics (A & B) that had arbitrarily defined costs associated with their latency times, where if the tactic took X amount of time to execute, it had C units of cost.

Table I
CHARACTERISTICS OF SAMPLE TACTICS FOR PROOF OF CONCEPT SIMULATION

	Cost	Distribution	Average	SD
Tactic A	5	Positively Skewed	3	0.5
Tactic B	7	Normal	3	0.5

We next generated a normal distribution and positively skewed distribution with the same mean and standard deviation to represent latency times. We gave the tactic with the higher cost (B) the normal distribution, while the tactic with the lower cost (A) was given the positively skewed distribution. By using two divergent values, we are able to demonstrate the impact of not accounting for tactic volatility in a near-perfect environment (the normal distribution) and in a volatile environment (the positively skewed distribution). It is possible that both latency and cost do not follow these types of distributions, but because they are two extremes, it was sufficient for our proof of concept. Table I shows the characteristics of each tactic.

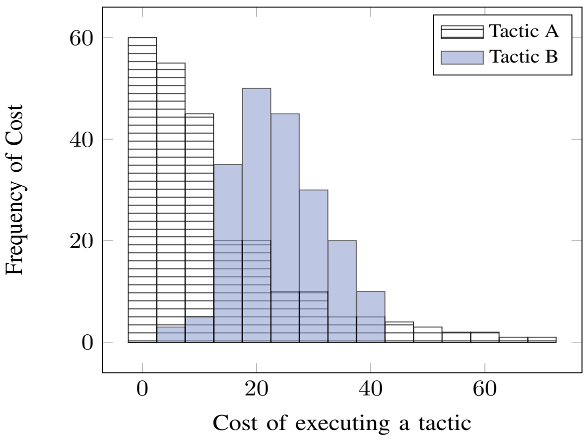

Figure 2. Overall costs of executing a tactic in the R simulations.

After generating the characteristics of each tactic, we then performed 100 simulations of random sampling from each tactic's latency and multiplied the sampled latency value by its associated cost. This enabled us to build a simulated distribution of "tactic executions", where the same tactic is executed every time, but with different latency values. This also allowed us to gather overall cost values that could be used to compare how varying data distributions can impact the predictability of cost for executing a tactic without accounting for any form of volatility.

In our analysis, we found that tactic B may not have had the lowest cost values in the simulations, but it was significantly more consistent (Figure 2). However, tactic B's data followed a normal distribution, which is an unlikely occurrence for many real-world systems. Furthermore, even though tactic A had a lower cost of execution with a value of five, it resulted in much more sporadic, and extremely high, overall costs to the system.

Outcome: *Our proof of concept simulations in R successfully demonstrate that accounting for tactic volatility is essential in self-adaptive systems, especially when the system is known to have unpredictable behavior.*

RQ2: How effective is ARIMA in allowing the system to monitor system specifications? To evaluate our ARIMA model for time-series forecasting, we used the energy data collected from our VALET tool. Although we had been using a self-adaptive hosting service as an example in this paper, specifically one that monitored response time, the energy usage data is of the same concept. For both, data is gathered over a period of time at equal intervals, thus qualifying it as time-series data. However, the only difference with this analysis is that we did not consider the energy usage when VALET is downloading the file. Therefore, the data used in this research question is strictly the energy usage fluctuations when the device was idle and *not* performing a tactic activity.

To specifically address this research question, we had to loop through the data multiple times to ensure that ARIMA could provide sustainable time-series forecasting predictions. We did this by creating 50 experiments using a randomly selected 90% portion of our data as training data, and the other 10% as test data. This type of process can also be seen as a form of *k-cross validation*, which ensures that each data point is included in the training set at least once. After performing this validation, we then calculated both RMSE and MAE values to determine the predictive ability of the ARIMA model.

Figure 3 shows the MAE results from our ARIMA model compared to a previously used Hidden Markov Model (HMM). Each simulation was independently performed, so no patterns should be inferred from the left to right sequence. Over the course of the experiments we saw fairly stable MAE values, represented by the small range between the lowest MAE value and the highest. Furthermore, the ARIMA model was not only successful by itself, but also better than that of the HMM model. We believe this occurred as we did not have enough features in our model to allow for a full treatment of a HMM.

As discussed previously, larger prediction errors will become more pronounced and smaller prediction errors will become less pronounced when using RMSE. Due to this, we expected

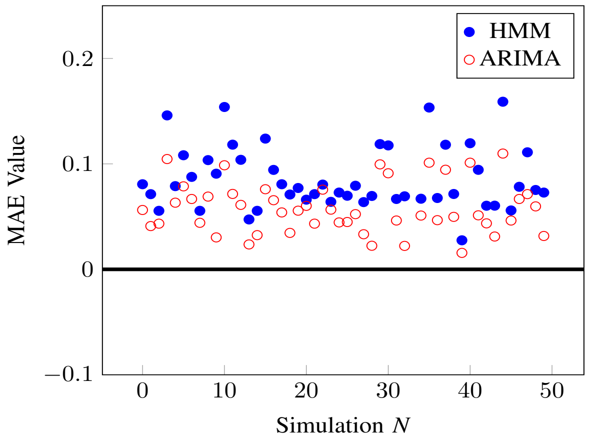

Figure 3. MAE Values Over 50 Experiments Using ARIMA vs HMM for Time Series Forecasting

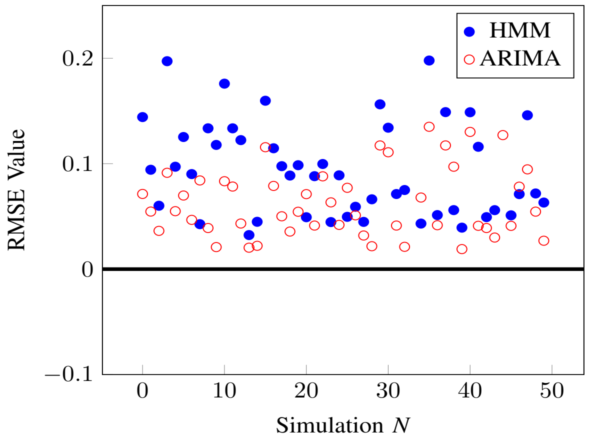

Figure 4. RMSE values over 50 experiments using ARIMA vs HMM for Time Series Forecasting

the RMSE graph to be slightly more dispersed compared to MAE. In examining the RMSE differences between HMM and ARIMA, we found exactly this. Looking at Figure 4 closely, we can see how the RMSE values have more variation than the MAE values in Figure 3. However, this does not mean there was less predictive power or that the model is less useful. We report on both to show the differences in possible inferences. For example, in applying our TVA technique the system's engineers may care more about larger prediction errors; deeming RMSE a more powerful statistic for determining predictive ability of their models. Conversely, if the system's engineers want more of a "man-in-the-middle" statistic, they may deem MAE more appropriate. For our experiments, using RMSE or MAE would lead to the same conclusion – both statistics clearly favor the ARIMA model over the HMM model for time series forecasting.

Outcome: *Through an evaluation process using time series data, TVA demonstrated it's ability to positively support proactive adaptations.*

RQ3: How effective is MRA in predicting tactic latency and cost at run time? Unpredictability is considered to be an undesirable trait of a self-adaptive system and is frequently associated with much of the uncertainty that surrounds self-adaptive systems [41], [45], [28], [14]. To determine how well TVA can improve the predictability of a self-adaptive system, we examined the prediction errors across our tactic latency data and our tactic cost data. For space considerations and the use of RMSE being more appropriate in this context, this research question does not report MAE values.

Unlike RQ #2 where we only examined the energy usage for when the Raspberry Pi was idle, RQ #3 only examined the energy usage when the device *was* performing the file download. If we had used the energy usage data from when the device was idle, our models would have been invalid. This is because the file download represented a tactic of gathering more information, thus any energy data gathered while the device was idle did not represent tactic latency.

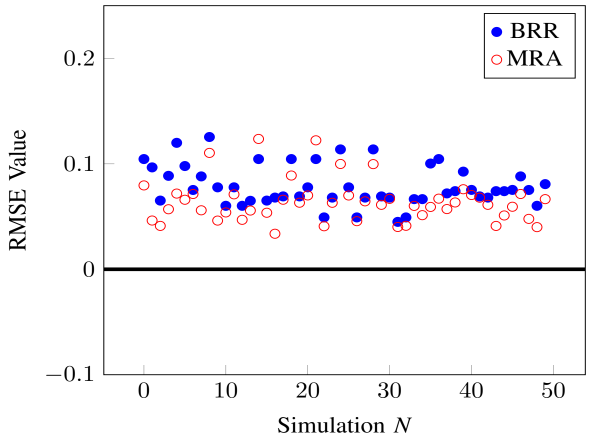

Figure 5. RMSE Values for Tactic Cost Predictions of MRA vs BRR

Figure 5 demonstrates why MRA is more appropriate for TVA. Over the course of the 50 simulations, MRA reported lower RMSE values than BRR in almost every case. In cases where BRR did outperform MRA, the differences were fairly negligible. However, the plots are closer together than what we would have initially expected. We believe this was caused by only having a few cases of extreme volatility in the cost data, therefore not allowing the two models to really differentiate themselves.

As shown in Figure 6, we observed very similar results between the RMSE values calculated for tactic latency predictions and those calculated for tactic cost. In most cases, prediction errors with MRA were much smaller than those of BRR, and in examining the figures closely, the differences in volatility that were experienced can be observed. As mentioned previously, there were not as many extreme cases of volatility in the tactic cost data. However, within the tactic latency data we saw many cases of volatility, with some cases being extreme. This can partially be seen in Figure 6 since more volatility will likely lead to larger prediction errors. Thus,

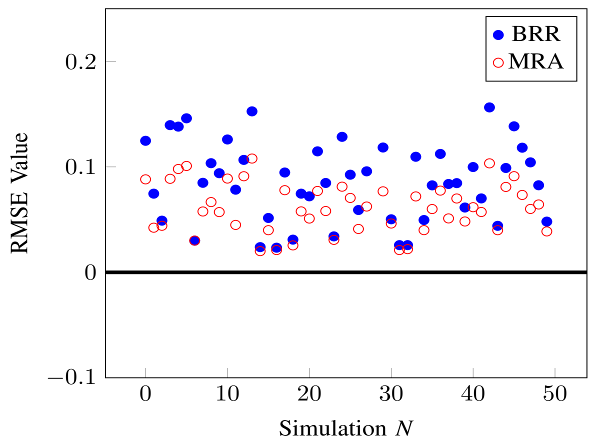

Figure 6. RMSE values for tactic latency predictions of MRA vs BRR

it came at no surprise that the RMSE values for tactic latency had a wider-spread than those associated with tactic cost.

In order to say the MRA model can be generalized to other data sets for predicting tactic latency and cost, observing consistent RMSE values regardless of the data it was modeling was imperative. Figure 5 and Figure 6 demonstrate that not only were RMSE values low for both data sets, but the values were stable. Therefore, MRA was able to handle the volatile latency data and cost data collected for these experiments when making its predictions.

Outcome: *The RMSE results gathered from addressing this research question demonstrate that MRA is able to handle tactic volatility when predicting tactic latency and cost. Throughout our experiments, MRA consistently provided stable predictive power, even in the presence of volatile data.*

RQ4: Does using TVA provide substantial improvement to the self-adaptive process over simply using static values for latency and cost? Thus far, we have demonstrated the benefits of using an ARIMA time series model and a MRA model in a self-adaptive process to predict tactic latency and cost, while also maintaining specifications defined in the SLA. However, to provide further confidence in our TVA approach, we also compared it to current self-adaptive processes. Since current processes consider tactic latency and cost to be static values, there is no one specific work that TVA can be compared to. Rather, for this research question, we compared the results from our approach to what we consider the “baseline” model defined below:

***Baseline Model**: A model that uses a simple average of previous tactic latency and cost values as its prediction process for estimating future tactic behavior.*

While utilizing the average value for latency and cost values may seem like a naive comparison, in many cases it can be a be a strong predictor especially when compared a static value (as in other current self-adaptive processes) on a dynamic time series. Similar to RQ #2, we will also report the MAE and RMSE values for model comparison.

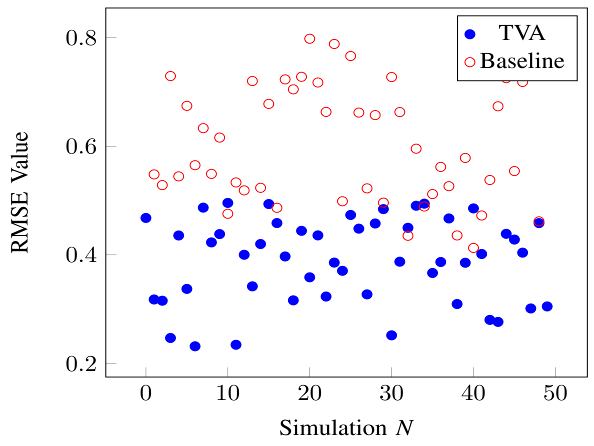

Figure 7. Baseline Value Approach vs TVA - RMSE Differences

As shown in Figure 7, TVA was able to obtain substantially lower RMSE values. In comparison, TVA had an average RMSE value of 0.0396 while the baseline approach had an average RMSE value of 0.0694. Also represented in this diagram is the ability of TVA to handle volatility. While the spread of RMSE values remained fairly consistent for TVA, the baseline approach saw a much wider spread; a direct result of only using a static value for latency and cost. Although there were cases where the baseline approach did outperform TVA (7 in total of 50 experiments), the differences in these values were fairly negligible.

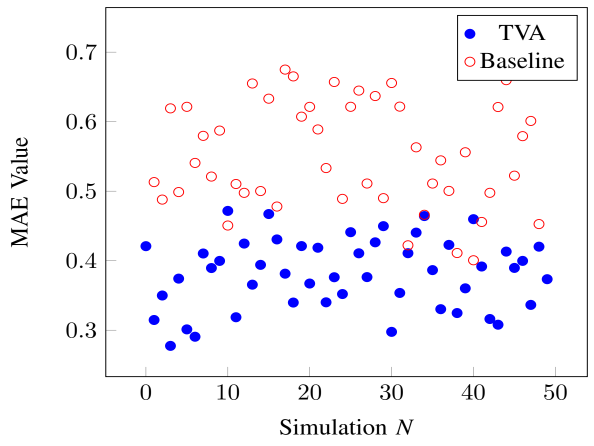

Figure 8. Baseline Value Approach vs TVA - MAE Differences

In examining the MAE values shown in Figure 8, we found that MAE did not behave as we expected. In RQ #2, it is fairly evident that RMSE made larger prediction errors larger, and smaller prediction errors smaller. However, in this research question, those kinds of results are not as evident. In examining the MAE values from TVA, we can see the plot is a bit more condensed than then RMSE plot, but the MAE values for the baseline model did not follow the same trend as strongly. We believe this occurred because the baseline model's prediction method was extremely poor, justified by both the RMSE and MAE plots. Regardless, TVA was still superior in predictive ability compared to the baseline model.

Outcome: *These findings demonstrate that TVA is beneficial for decision-making processes in self-adaptive systems, and offer a significant improvement over assuming tactic latency and tactic cost to be static values.*

VI. Related Works

Although our work is, to the best of our knowledge, the first known to make tactic volatility a first-class concern in the self-adaptive decision-making process, previous works have examined the impact of tactic latency in self-adaptive systems. Cámara *et al.* [6] was likely the first to consider tactic latency and examined how considering latency could be used to assist the proactive adaptation process. However, this work differs from TVA in that it does not consider any forms of tactic volatility that are likely to be encountered.

In SB-PLA, the latency of an adaptation is considered in the adaptation decision-making process [31]. A primary benefit of SB-PLA is that systems that cannot use tactic-based adaptations can still include latency awareness in their decision-making process [35]. In addition to supporting pro-activeness and concurrent tactic execution, PLA techniques also account for latency. PLA considers the amount of time necessary for a tactic to execute, in order to avoid situations that are not achievable when time dimensions are recognized. However, like with many latency aware approaches, latency is still considered to be a static attribute. In our work, we do not consider latency to be static and provide the system with the ability to predict tactic latency at run time.

Jamshidi *et al.* [24] presented `FQL4KE`, a self-learning fuzzy cloud controller. This enables systems to not rely upon design-time knowledge, but allows users to simply adjust weights that represent system priorities. This work found that their proposed process outperformed a previously devised technique that did not have a learning mechanism. Our work is similar in that we both utilize learning to enable the system to make better decisions. However, our work differs in that Jamshidi *et al.* focused on improving resource planning, and not in addressing tactic volatility as is accomplished by TVA.

While our work is the first to account for *cost volatility*, existing research has considered cost in the self-adaptive decision-making process. Several works have included cost in their utility equations, however they consider it to be a static value and do not account for real-world volatility [37], [38], [35], [36]. Jung *et al.* [26] demonstrated that ignoring cost can have a significant impact on the ability to satisfy response-time-based SLAs. This work also proposed a cost-sensitive self-adaption engine using middleware to create adaptation decisions. This work differs from ours in that it only considers cost in cloud-based controllers, while we focus on cost during the entire decision-making process.

Esfahani *et al.* [15] utilized learning to improve the self-adaptive process. A primary contribution of this work is a new process of reasoning and assessing adaptation decisions using online learning. A preliminary work by Elkhodary [23] proposed combining feature-orientation, learning and dynamic optimization techniques to create a new class of self-adaptive systems that would be able to modify their adaptation logic at run time. TVA differs from this work in that learning is utilized to predict tactic latency and cost at run time, while also providing away to estimate future values for requirements.

Kinneer *et al.* [29] developed a process for reusing prior planning knowledge to help the system to adapt to unexpected situations. This process considered that tactics may fail, and supports reasoning about tactic latency. While this work is helpful for assisting the overall planning process, it does not enable the system to actively learn and predict future values for tactic latency and cost like our TVA approach does.

Machine learning has been utilized to help determine the most efficient configurations for self-adaptive systems, while also performing adaptation planning. Quin *et al.* [40] enhanced the traditional MAPE-K feedback loop through the use of a learning model that selects subsets of adaptation options from a larger set of adaptation possibilities. This process enables the system to make more efficient analysis decisions. Jamshidi *et al.* [22] used machine learning to discover Pareto-optimal configurations to eliminate the need to explore every configuration. This work also restricted the search space necessary to make planning tractable. These works differ from ours in that while they use machine learning to create a more efficient self-adaptive process, they do not use machine learning to address the issue of tactic volatility.

The popular FIFA 98 dataset [4], [2] is a collection of requests made to the 1998 World Cup website over an approximately four month span. This dataset is widely used in self-adaptive research and can be used to simulate when new servers would have to be activated up to handle additional web traffic or when to disable resource heavy features on a website because the traffic load is impeding the system. However, this dataset does not contain latency. Each request is a log entry that consists of information such as timestamp, size, status, and URL hit. Because this dataset does not contain any latency information, such as a request received and request fulfillment time, it cannot be used in collaboration with researching and evaluating latency-aware strategies such as our proposed TVA process.

VII. Threats and Future Work

Our evaluations have demonstrated TVA's ability to enable systems to better account for tactic volatility. However, there are limitations to this work. In many systems, tactic cost may be an ambiguous and tough-to-define measurement. This inability to accurately measure cost could inhibit the adoption of our process by limiting the quality and quantity of observed input values into our prediction process. Furthermore, cost can also be a relative term, and in our TVA approach we consider it to be a quantifiable value. For example, one could argue that the 'cost' for performing an action could be the wear and tear on a physical component in the device. Such cost is difficult to quantify in most situations. Therefore, when using TVA the notion of cost must be restricted to a value that is reasonably easy to quantify.

Although TVA provides a method of monitoring specifications defined in the SLA, not all specifications are necessarily measurable with time-series analysis. For example, a system may have the specification that it must be available 99.99% of the time. This is not something that could be measured or predicted using time-series analysis, rather it would need to be accounted for using fault detection techniques in the system's architecture. In systems that do not directly utilize a SLA, the improved tactic estimations can still be used to benefit the decision-making process by providing more informed and accurate tactic attribute values. Future work should be conducted to examine precisely how our TVA process can be incorporated into these systems and determine the benefits that they will have.

To be completely proactive, future work must be done to update our ARIMA method to consider tactic latency. Currently, this process can alert the system of a specification that is about to be broken, however the time between monitoring intervals may not leave enough time to fully execute a tactic. Therefore, there may be occurrences where specifications are slightly broken for a period of time. This future work will likely examine other time series models as well that can be flexible to different systems and datasets. This in turn would also help us to start addressing the problem of not having enough data to build other kinds of models and would allow this work to develop into an entire decision-making framework.

Although we have demonstrated the benefits of our TVA approach with real-world experimental data, we have yet to implement TVA on any physical devices. Future work will consist of including our adaptation process into physical equipment such as IoT devices, small unmanned aerial vehicles (UAV), and self-adaptive web systems.

ARIMA demonstrated its ability to perform time series forecasting, however no mechanisms are in place to quantify uncertainty within these predictions. Future work could be done to include confidence intervals around predictions made by ARIMA. For example, instead of only predicting a single point value for a specification, we could predict a range that states something such as the following, "*we are 95% certain response time will be between 3.1 and 3.7 seconds*". This would allow the system to have a "buffer" zone around its predictions, therefore providing the decision-making process with more information.

Additionally, the ARIMA models utilized in this work were pre-trained on gathered historical data. It is also possible to have the ARMIA models be updated online as new data is gathered by a system to adapt themselves as more information becomes available. Other methods for time series prediction such as recurrent neural networks (RNNs), which have been successfully used for a variety of time series forecasting problems [10], [34], [17], [16], [8], [13], [12], can be examined as well. RNNs may also potentially be able to incorporate non-time series data into predictions.

Our evaluation data was created using two Raspberry Pi's, and does not simulate a complicated system such as a self-driving car or a UAV. However, this generated data was intended to help demonstrate the capabilities of TVA and the benefits of accounting for tactic volatility. In reality, the data generated by these tools could represent virtually any form of tactic volatility (*e.g.,* the time required for a UAV to communicate with a base station or the energy cost of a self-driving car performing a tactic).

There are a few potential limitations to the use of our VALET tool and dataset in the evaluation of our proposed TVA process. First, VALET generates its data by performing a small number of tasks. A real-world self-adaptive system would likely perform a large number of tasks in any given adaptation, which depending on the system, could impact one another. VALET is also limited in the forms of variability that it may encounter as opposed to a real-world system. For example, VALET is significantly less likely to be actively targeted by human hackers than many real-world self-adaptive systems, which could limit the encountered variability. However, it is somewhat unreasonable to expect that any created evaluation system would have the capabilities to address a majority of real-world events and possibilities. Despite these possible limitations, we are confident in the ability of VALET in creating satisfactory evaluation data for not only our study, but for future research conducted by others.

VIII. Conclusion

In this work, we propose a Tactic Volatility Aware (TVA) process that is able to account for tactic volatility in multiple ways. TVA first uses time-series forecasting with an autoregressive integrated moving average (ARIMA) model to monitor system specifications defined in the SLA, supporting the system in acting more proactively while maintaining them. Next, TVA uses multiple regression analysis (MRA) to predict tactic latency and cost helping to improve the decision-making process. Our contribution also includes a tool that utilizes physical devices to create real-world tactic volatility data.

Acknowledgements

Elements of this work are sponsored by the United States Air Force Research Laboratory (AFRL).

REFERENCES

[1] The internet traffic archive. http://ita.ee.lbl.gov/.

[2] Worldcup98. http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[3] I. D. P. Anaya, V. Simko, J. Bourcier, N. Plouzeau, and J.-M. Jézéquel. A prediction-driven adaptation approach for self-adaptive sensor networks. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 145–154. ACM, 2014.

[4] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. *IEEE network*, 14(3):30–37, 2000.

[5] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Software engineering for self-adaptive systems. chapter Engineering Self-Adaptive Systems Through Feedback Loops, pages 48–70. Springer-Verlag, Berlin, Heidelberg, 2009.

[6] J. Cámara, G. A. Moreno, and D. Garlan. Stochastic game analysis and latency awareness for proactive self-adaptation. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 155–164. ACM, 2014.

[7] J. Cámara, G. A. Moreno, D. Garlan, and B. Schmerl. Analyzing latency-aware self-adaptation using stochastic games and simulations. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(4):23, 2016.

[8] E. Choi, M. T. Bahadori, and J. Sun. Doctor ai: Predicting clinical events via recurrent neural networks. *arXiv preprint arXiv:1511.05942*, 2015.

[9] A. Dhanapal and P. Nithyanandam. An effective mechanism to regenerate http flooding ddos attack using real time data set. In *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, pages 570–575. IEEE, 2017.

[10] L. Di Persio and O. Honchar. Artificial neural networks approach to the forecast of stock market price movements. *International Journal of Economics and Management Systems*, 1, 2016.

[11] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, Dec. 2006.

[12] A. ElSaid, S. Benson, S. Patwardhan, D. Stadem, and T. Desell. Evolving recurrent neural networks for time series data prediction of coal plant parameters. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 488–503. Springer, 2019.

[13] A. ElSaid, F. El Jamiy, J. Higgins, B. Wild, and T. Desell. Optimizing long short-term memory recurrent neural networks using ant colony optimization to predict turbine engine vibration. *Applied Soft Computing*, 73:969–991, 2018.

[14] N. Esfahani. *Management of uncertainty in self-adaptive software*. PhD thesis, George Mason University, 2014.

[15] N. Esfahani, A. Elkhodary, and S. Malek. A learning-based framework for engineering feature-oriented self-adaptive software systems. *IEEE transactions on software engineering*, 39(11):1467–1493, 2013.

[16] M. Felder, A. Kaifel, and A. Graves. Wind power prediction using mixture density recurrent neural networks. In *Poster Presentation gehalten auf der European Wind Energy Conference*, 2010.

[17] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation, Vol. 12, No. 10 , Pages 2451-2471*, October 2000.

[18] 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)*, pages 57–64, 2014.

[19] D. Garlan, B. Schmerl, and S.-W. Cheng. *Software Architecture-Based Self-Adaptation*, pages 31–55. Springer US, Boston, MA, 2009.

[20] N. Huber, A. Hoorn, A. Koziolek, F. Brosig, and S. Kounev. Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments. *Serv. Oriented Comput. Appl.*, 8(1):73–89, Mar. 2014.

[21] R. Hyndman and K. A. Another look at measures of forecast accuracy. pages 314–333, 2005.

[22] P. Jamshidi, J. Cámara, B. Schmerl, C. Kästner, and D. Garlan. Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots. In *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '19, pages 39–50, Piscataway, NJ, USA, 2019. IEEE Press.

[23] P. Jamshidi, C. Pahl, and N. C. Mendonça. Managing uncertainty in autonomic cloud elasticity controllers. *IEEE Cloud Computing*, 3(3):50–60, May 2016.

[24] P. Jamshidi, A. M. Sharifloo, C. Pahl, A. Metzger, and G. Estrada. Self-learning cloud controllers: Fuzzy q-learning for knowledge evolution. In *Proceedings of the 2015 International Conference on Cloud and Autonomic Computing*, ICCAC '15, pages 208–211, Washington, DC, USA, 2015. IEEE Computer Society.

[25] M. Jeroen Van Der Donckt, D. Weyns, M. Iftikhar, and R. Singh. Cost-benefit analysis at runtime for self-adaptive systems applied to an internet of things application. pages 478–490, 01 2018.

[26] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. A cost-sensitive adaptation engine for server consolidation of multitier applications. In *Proceedings of the ACM/IFIP/USENIX 10th International Conference on Middleware*, Middleware'09, pages 163–183, Berlin, Heidelberg, 2009. Springer-Verlag.

[27] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[28] C. Kinneer, Z. Coker, J. Wang, D. Garlan, and C. L. Goues. Managing uncertainty in self-adaptive systems with plan reuse and stochastic search. 2018.

[29] C. Kinneer, D. Garlan, and C. Le Goues. Information reuse and stochastic search: Managing uncertainty in self-* systems. 2019.

[30] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *Future of Software Engineering (FOSE '07)*, pages 259–268, May 2007.

[31] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with prism: A hybrid approach. *International Journal on Software Tools for Technology Transfer*, 6(2):128–142, 2004.

[32] J. Liu, B. Priyantha, T. Hart, H. S. Ramos, A. A. Loureiro, and Q. Wang. Energy efficient gps sensing with cloud offloading. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 85–98. ACM, 2012.

[33] L. Määttä, J. Suhonen, T. Laukkarinen, T. D. Hämäläinen, and M. Hännikäinen. Program image dissemination protocol for low-energy multihop wireless sensor networks. In *2010 International Symposium on System on Chip*, pages 133–138. IEEE, 2010.

[34] N. Maknickienė and A. Maknickas. Application of neural network for forecasting of exchange rates and forex trading. In *The 7th international scientific conference" Business and Management*, pages 10–11, 2012.

[35] G. A. Moreno. *Adaptation Timing in Self-Adaptive Systems*. PhD thesis, Carnegie Mellon University, 2017.

[36] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 1–12. ACM, 2015.

[37] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl. Flexible and efficient decision-making for proactive latency-aware self-adaptation. *ACM Trans. Auton. Adapt. Syst.*, 13(1):3:1–3:36, Apr. 2018.

[38] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl. Efficient decision-making under uncertainty for proactive self-adaptation. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*, pages 147–156, July 2016.

[39] G. A. Moreno, A. V. Papadopoulos, K. Angelopoulos, J. Cámara, and B. Schmerl. Comparing model-based predictive approaches to self-adaptation: Cobra and pla. In *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '17, pages 42–53, Piscataway, NJ, USA, 2017. IEEE Press.

[40] F. Quin, D. Weyns, T. Bamelis, S. S. Buttar, and S. Michiels. Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning. In *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '19, pages 1–12, Piscataway, NJ, USA, 2019. IEEE Press.

[41] A. Rodrigues, R. D. Caldas, G. N. Rodrigues, T. Vogel, and P. Pelliccione. A learning approach to enhance assurances for real-time self-adaptive systems. *arXiv preprint arXiv:1804.00994*, 2018.

[42] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, 4(2):14, 2009.

[43] B. R. Schmerl, J. Cámara, J. Gennari, D. Garlan, P. Casanova, G. A. Moreno, T. J. Glazier, and J. M. Barnes. Architecture-based self-protection: composing and reasoning about denial-of-service mitigations. In *HotSoS*, 2014.

[44] N. Stakhanova, S. Basu, and J. Wong. A cost-sensitive model for preemptive intrusion response systems. In *21st International Conference on Advanced Information Networking and Applications (AINA '07)*, pages 428–435, May 2007.

[45] S. Tomforde, S. Rudolph, K. Bellman, and R. Würtz. An organic computing perspective on self-improving system interweaving at runtime. In *Autonomic Computing (ICAC), 2016 IEEE International Conference on*, pages 276–284. IEEE, 2016.