4-2019

TVA: A Requirements Driven, Machine-Learning Approach for Addressing Tactic Volatility in Self-Adaptive Systems

Jeffrey Palmerino
jrp3143@rit.edu

TVA: A Requirements Driven, Machine-Learning Approach for Addressing Tactic Volatility in Self-Adaptive Systems

by

Jeffrey Palmerino

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Software Engineering

Supervised by

Dr. Daniel Krutz

Department of Software Engineering

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

April 2019

The thesis “TVA: A Requirements Driven, Machine-Learning Approach for Addressing Tactic Volatility in Self-Adaptive Systems” by Jeffrey Palmerino has been examined and approved by the following Examination Committee:

Dr. Daniel Krutz
Assistant Professor, Department of Software Engineering
Thesis Committee Chair

Dr. Qi Yu
Associate Professor, Department of Information Sciences and Technologies

Dr. Travis Desell
Associate Professor, Department of Software Engineering

Acknowledgments

I am grateful for all of those who took time out of their schedules to guide and aid me in developing this thesis. I would also like to give special thanks to the United States Air Force Office of Scientific Research (AFOSR) who sponsored elements of this work.

Abstract

TVA: A Requirements Driven, Machine-Learning Approach for Addressing Tactic Volatility in Self-Adaptive Systems

Jeffrey Palmerino

Supervising Professor: Dr. Daniel Krutz

From self-driving cars to self-adaptive websites, the world is increasingly becoming more reliant on autonomous systems. Similar to many other domains, the system's behavior is often determined by its requirements. For example, a self-adaptive web service is likely to have some maximum value that response time should not surpass. To maintain this requirement, the system uses*tactics*, which may include activating additional computing resources. In real-world environments, tactics will frequently experience volatility, known as *tactic volatility*. This can include unstable time required to execute the tactic or frequent fluctuations in the cost to execute the tactic. Unfortunately, current self-adaptive approaches do not account for tactic volatility in their decision-making processes, and merely assume that tactics have static attributes.

To address the limitations in current processes, we propose a *Tactic Volatility Aware* (TVA) solution. Our approach focuses on providing a volatility aware solution that enables the system to properly maintain requirements. Specifically, TVA utilizes a Autoregressive Integrated Moving Average Model (ARIMA) to estimate potential future values for requirements, while also using a Multiple Regression Analysis (MRA) model to make predictions of tactic latency and tactic cost at runtime. This enables the system to both better

estimate the true behavior of its tactics and it allows the system to properly maintain its requirements. Using data containing real-world volatility, we demonstrate the effectiveness of using TVA with both statistical analysis methods and self-adaptive experiments. In this work, we demonstrate (I) The negative impact of not accounting for tactic volatility (II) The benefits of a ARIMA-modeling approach in monitoring system requirements (III) The effectiveness of MRA in predicting tactic volatility (IV) The overall benefits of TVA to the self-adaptive process. This work also presents the first known publicly available dataset of real-world tactic volatility in terms of both cost and latency.

Chapter 1

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 UAVs, as well as both IoT and medical devices. With widespread adoption into many domains, it is imperative that these systems remain effective, efficient and resilient, even when operating in stochastic environments. To accomplish this, one path we can take is make requirements the driving force behind adaptations, as they can play a key role in determining if, when, and how the system should adapt to change. For current self-adaptive systems, these requirements are defined in the Service Level Agreement (SLA). Within the SLA are definitions for how utility is calculated for the system, along with defined rewards for meeting specific requirements, and defined penalties for not meeting these requirements [20]. While self-adaptive systems spread across a wide variety of domains, SLA's can vary considerably from system to system.

Regardless if requirements varying from system to system, *tactics* are how these systems handle changes in their surrounding environments. Examples of these tactics include the provisioning of an additional virtual machine in a web farm when workload reaches a specific threshold, or reducing non-essential functionality on an autonomous plane when battery levels are low. In current self-adaptive systems, many approaches utilize *adaptation tactics* to make necessary changes [28, 35, 25, 14, 12] or closed-loop control systems to monitor current states and to determine the appropriate adaptation tactics to execute [22, 2, 34, 7]. However, tactics frequently have two characteristics in common: *tactic latency* and *tactic cost*. Tactic latency is the amount of necessary time from when a tactic

is invoked until its effect on the system is realized [28, 31, 32, 27]. Tactic cost refers to the relative expense that the system experiences in performing the tactic. Cost may be in the form of resource consumption (battery power, workload on servers, *etc.*), or in actual monetary value [20, 36, 18]. Both tactic latency and cost are likely to be first-class concerns in the decision-making process of a self-adaptive system, as they may directly impact the selection of the most appropriate tactic(s) and when potential future tactics may begin to be executed [28, 32, 27]. Due to their significant impact on the decision-making process, improperly accounting for tactic latency or tactic cost can lead to situations where tactics are executed too early, too late, or they consume too many additional resources; hindering the rest of the adaptation process. This will frequently result in higher levels of uncertainty within the decision-making process, as the system's behavior will become much more erratic and unpredictable.

Real-world systems will frequently encounter *tactic volatility*, which is variability in the attributes of the tactic. Tactic volatility can be caused by innumerable internal and external factors, and should be accounted for in a real-world, robust self-adaptive system. As an example, one tactic in a cloud-based system could be to add an additional server resource when needed to handle an increased workload [28]. The time necessary to add the extra server would be a primary concern for two reasons including: (I) When to begin the process of adding the resource since adding it too early before it is needed could incur unnecessary costs, while adding it too late will mean that it is not ready when needed; (II) The anticipated latency could impact the decision of if the tactic of adding a resource is the most appropriate tactic over alternative options. Due to multiple factors such as software configurations, varying network congestion or even hardware failure, the necessary amount of time to add the additional resource could widely vary. Therefore, using a static latency value this tactic is not sufficient for ensuring that the system operates at optimal levels.

Unfortunately, state-of-the-art decision-making processes in self-adaptive systems do not sufficiently account for tactic volatility, which can adversely impacting the decision-making process [28, 27]. For example, a system may assume that it always takes two

seconds to execute a tactic when in reality it has been observed to fluctuate rapidly between one and five seconds. Therefore, without being able to learn from these observations, the system will continue to make incorrect assumptions about how tactics behave – frequently leading to less than optimal decisions.

To address the limitation of current self-adaptive processes not accounting for tactic volatility, we propose a *Tactic Volatility Aware* (TVA) solution. Our approach focuses on handling tactic volatility, while also maintaining system requirements defined in the SLA. TVA utilizes an Autoregressive Integreated Moving Average (ARIMA) to estimate potential future values for requirements, while also using a Multiple Regression Analysis (MRA) model to make predictions of tactic latency and tactic cost at time of adaption. This not only allows the system to accurately predict how tactics are going to behave, but it also supports the system to better maintain its requirements. In this work, we use data containing real-world volatility to demonstrate the positive impact that TVA has in handling tactic volatility to maintain system requirements.

To summarize, this work makes the following contributions:

1. **Requirements Monitoring:** Our TVA approach takes into account requirements that have been defined by the systems engineers in the SLA. TVA uses time series forecasting with an ARIMA model to enable the system to maintain important functional requirements by knowing when and/or if a requirement will be broken in the near future. Furthermore, by examining requirements in the SLA, TVA ensures that requirements with higher priorities are addressed first.
2. **Concept:** Our TVA approach, to the best our knowledge, is the first process that accounts for tactic volatility. Existing processes merely consider tactics to have static values, whereas our TVA approach enables the system to better adapt to real-world tactic volatility through the use of runtime predictions of latency and cost.
3. **Experiments:** We conducted several experiments using real-world scenarios and systems that emulate a self-adaptive system to demonstrate the positive impact TVA

has on the decision making process. These experiments are conducted using real-world data, thus providing additional confidence in our findings. Furthermore, our proof-of-concept simulation demonstrates the negative effects of not accounting for tactic volatility.

4. **VALET Tool:** Our custom VolAtiLety EmulaTor (VALET) tool includes forms of tactic volatility data generated from a physical system. The data generated from the physical system enables the evaluation of our TVA process and provides other researchers with a foundational dataset that they may use to support their own research in self-adaptive systems and autonomy.

The rest of the paper is organized as follows: Section 3 defines the keywords in our work using a self-adaptive web service example. Section 4 discusses our proposed technique using ARIMA and MRA, while also examining the general workflow of our approach. Section 6 examines the research questions our work addresses, and discusses how experiments were set up and any analysis tools used. Section 7 and Section 8 discuss the results from evaluating TVA and its implications on self-adaptive systems. This work is concluded with details of the proposed VALET tool and limitations/future work ideas that came out of our work.

Chapter 2

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.*[4] 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 did not consider any forms of tactic volatility that tactics are likely to encounter.

In SB-PLA, the latency of an adaptation is considered in the adaptation decision-making process [26]. A primary benefit of SB-PLA is that systems that cannot used tactic-based adaptations can still include latency awareness into their decision-making process [28]. In addition to supporting pro-activeness and concurrent tactic execution, PLA techniques also recognizes latency awareness. 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, it 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 runtime.

Jamshidi *et al.*[17] 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. The experimental results found that this 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 accounted for cost in the self-adaptive decision-making process. While several works have considered cost in their utility equations, it is still considered to be a static value that does not account for real-world volatility [31, 27, 28, 30]. Jung *et al.*[21] demonstrated that ignoring costs 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 TVA in that cost is addressed in only cloud-based controllers, while we focus on cost in the entire decision-making process.

Esfahani *et al.*[9] 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 [16] 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 runtime, while also providing away to estimate future values for requirements.

Kinneer *et al.*[24] developed a process for reusing prior planning knowledge to help the system to adapt to unexpected situations. This process considered that tactics mail fail, and supports reasoning about tactic latency. While this work is helpful for assisting in 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.

Chapter 3

Problem Definition

To define the key concepts and problems addressed in our work, we will use a self-adaptive web service scenario as our motivating example.

3.1 Monitoring SLA Requirements

To begin our motivating example, we will define what an SLA might look like for a self-adaptive web service. We will assume that one of the main requirements of this web service is that "*response time is not allowed to, or should not, surpass specific thresholds*". Meaning, when response times surpasses some threshold, the web service is going to experience some kind of penalty as defined in the SLA [20]. This may come in the form of overloaded servers or interruptions in loading content causing negative impacts to the user experience. For this scenario specifically, a basic SLA with requirement (REQ) possessing our arbitrarily defined penalties (P) and rewards (RWD) may be defined as:

$$SLA = \begin{cases} REQ = \text{response time} \leq 3 \text{ seconds} \\ P = 3 \\ RWD = 10 \end{cases}$$

In real-world systems, however, the system is going to have more than one requirement defined for it. Therefore, it is imperative that the system has the ability to know

what requirements to do address and *when* to address them. Fortunately, this can be determined by another component of the SLA. By examining the defined reward (RWD) for requirements, the system is able to know the exact order in which requirements need to be addressed since (RWD) represents how much "reward" the system gains from maintaining the requirement. For example, if the web service also had the following requirement where $capacity$ represents the maximum load for a server:

$$SLA = \begin{cases} REQ = \text{server loads} \leq .75 * capacity \\ P = 2 \\ RWD = 7 \end{cases}$$

and determined that both needed to be addressed, it would first address the response time issue, and then handle the server load issue since the reward for the former is greater than that of the latter.

Yet, knowing all of this still does not allow the web-service to properly maintain its requirements. In order for the web service to do this, we must allow the system to become "proactive" rather than "reactive". In many of today's self-adaptive processes, systems adapt to surrounding changes in a "reactive" manner. For example, if the web-service was acting in a reactive manner it would not adapt to broken requirements until they were actually broken. This inevitably would lead to the system experiencing the penalty defined for the requirement since it is now broken, possibly leading to further issues within the system. However, if we provide the web service the ability to know when a requirement may be broken in the future, the system could adapt and handle this change before it becomes an issue – thus acting in a "proactive" manner.

3.2 Tactic Latency

The amount of time necessary to complete an implementation of a tactic is known as *tactic latency* [5]. Self-adaptive systems must account for tactic latency as accurately as possible

since it can have a large impact on the decisions the system makes. For example, the anticipated latency of a tactic can impact the selection of the most appropriate tactic(s) for an encountered scenario, and when the tactics are invoked to ensure that they are available when needed [28]. Previous works have shown that latency aware algorithms offer several advantages compared to those that do not consider latency [4, 28]. However, these same processes still consider tactic latency to be a predetermined, static value. In reality, these tactic latency times can be highly volatile due the environments the system is operating in. Thus, it is imperative that the system has a reliable, accurate way to estimate latency.

Since the success of a web service heavily depends on how network traffic is managed, whether it be to regulate resource consumption or provide a fluent user experience, dealing with tactic latency plays a crucial role in maintaining viable operations. For example, when a web service experiences spikes or elongated times of increased network traffic, it can account for this in two ways: (I) Reduce the amount of content they are showing to the user at a time, or (II) adding additional servers to support the increased network traffic. For the web service, these two options would be considered the available tactics. The former has negligible latency, but the latency of the latter can be on the order of minutes. Therefore, if the service determines that more servers are needed to handle the increase in network traffic, it is imperative that it anticipates this time accurately. For example, the web service may define that adding a server only takes one minute to become available (*i.e.*, the static approach to defining tactic latency). Due to this, it may decide that it is not worth reducing content shown to the user in the mean time. However, if the additional server suddenly takes four minutes to become available, the system has now found itself in a no-win scenario. Not only was the static assumption of latency extremely off, but the system could have already been dealing with the network traffic slightly by reducing content shown to the user. *Therefore, accounting for tactic latency volatility is a paramount concern.*

3.3 Tactic Cost

To demonstrate the need to account for tactic cost volatility, we will continue to use our example of the self-adaptive web service. However, in this scenario, we will assume that the system's main source of power is through a battery and not direct power through outlets or other resources.

An extremely common process in web services is the backing up of data by sending it to servers or logging it internally through their networks. Transmitting this data regularly to backup servers is crucial for some systems because it may contain user activity, account info, or person informational that would be detrimental if lost. Since this web service operates using mostly battery power, the energy cost to perform this action will likely be a primary concern in deciding when to perform this data transfer process. Furthermore, it is likely that this will also dictate the frequency of this operation as well. Unfortunately, the cost for performing this action could be volatile due to numerous reasons. These could include include erratic hardware behavior (battery or another component not operating at optimal levels), or the file transfer taking longer than expected (packet loss, adversarial actions). Since the system's battery has a finite amount of energy, the relative cost of this transfer process increases as the battery level decreases.

To demonstrate the deficiency of assuming tactic cost to be a static value, we will assume that the web service initially defined the backup process to have a cost (C) of 5 units. However, after observing the backup process for a period of time, it was found that cost behaved quite erratically; returning cost values of $C = \{10, 2, 7, 12, 15\}$. This means that when using a static value, the system could have potentially performed the backup process while consuming the remaining amount of battery power, thus leading to an overall crash of the system. *Therefore, when cost is a consideration in the decision-making process, volatility should be a first class-concern.*

Chapter 4

Proposed TVA Process

Our TVA process differs from current techniques in that we do not assume tactic characteristics to have static values [31, 19, 28]. Specifically, our approach combines a requirements-monitoring process that uses time-series analysis and a machine-learning model to predict tactic latency and cost at runtime – allowing the system to have a better understanding of how its tactics behave regardless of the environments they operate in. In this section, we describe our TVA approach in the following phases: I) An examination of Autoregressive Integrated Moving Average (ARIMA) models II) How Multiple Regression Analysis (MRA) is used to predict tactic latency and cost III) Discussing how ARIMA and MRA are specifically used in the TVA workflow.

4.1 Autoregressive Integrated Moving Average Model

The first component of our TVA work revolves around the ability to monitor requirements. To do this, TVA conducts time series analysis, which is *the examination of data that has been indexed over a specific period time, at equal intervals, in order to predict potential future values*. For example, recording the outside temperature at the same time everyday so you could predict the next day's temperature at that time would qualify as valid time series analysis. In our TVA approach, we specifically use *time series forecasting*, which is the analysis of time series data to predict future values based on old values. Initially, our work examined the use of a Hidden Markov Model (HMM) to perform time series forecasting. However, through multiple evaluations of HMM against other types of models, we decided

it was not appropriate (more details in Section 6). The time series model that we found was most appropriate for our work and the data we collected, was an Autoregressive Integrated Moving Average (ARIMA) model – described in this section.

ARIMA is one of the most commonly used approaches for time series modeling [13]. It is considered to be a generalization of an ARMA (autoregressive moving average) model, in which both models can be used to fit time series data in order to predict future points in the series. Specifically, ARIMA is defined by the following three parts (included in its name):

1. **AR** - Standing for autoregression, this portion of ARIMA uses a dependent relationship between an observation and some number of lagged observations
2. **I** - Standing for integrated, this portion of the model represents the use of "differencing" raw observations to make the time series stationary
3. **MA** - Standing for moving average, this portion of ARIMA represents the use of dependencies between an observation and its residual errors from a moving average model, applied to lagged observations

with the main difference between the two being the inclusion of I in ARIMA. Furthermore, a full ARIMA treatment also requires the following notation:

$$ARIMA(p, d, q) \tag{4.1}$$

where p represents the number of lag observations included in the model, d represents the degree of differencing (discussed further in this section), and q represents the size of the moving average window, also known as the order of moving average. Another ARIMA-like model that was potentially appropriate for our work was seasonal-ARIMA (SARIMA). SARIMA applies all the same techniques that ARIMA does, but is more robust when dealing with seasonal time series data. However, upon examination of the time series data collected for this work, we did not find any seasonality within the data.

There are multiple ways in which an ARIMA model can be applied. In this work, we used the Box-Jenkins approach to find the best fit of a time series model for predicting

future values. To properly apply this approach, we must apply the following three components/steps of Box-Jenkins:

Identification One of the main reasons ARIMA is more appropriate for our work than ARMA is that ARMA has a strict assumption that the time series data will always be stationary – therefore the model is incapable of "identifying" any non-stationary data. When non-stationary data exists, we must transform it to be stationary. This is done through a method called *differencing*, which examines the difference between consecutive observations. Mathematically, this is shown as:

$$y'_t = y_t - y_{t-1} \tag{4.2}$$

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. For some cases, it is possible that differencing needs to be applied a second time in order to obtain stationary data. If we find this to be true, we refer to this as *second order differencing*.

Once we have transformed non-stationary data to stationary data, the next step is to identify the order of our autoregressive and moving average terms (i.e. p and q). There are many different approaches to how p and q can be identified, but for this work we used the classic autocorrelation and partial autocorrelation plots. Both plots allow us to summarize the correlation of an observation with lagged values, however, the latter does so with lagged values that are not accounted for by prior lagged observations.

Estimation In order to estimate parameters for Box-Jenkins models, we must apply a solution that can numerically approximate nonlinear equations. Like the correlation plots, there are many ways to do this whether it be a nonlinear least squares estimation or a maximum likelihood estimation. However, the goal here is to minimize a loss or error term, therefore allowing any predictions that are made to be as accurate as possible. With this reasoning, our approach uses the preferred *maximum likelihood estimate* (MLE) method for estimating

the models parameters.

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 tweak orders in the ARIMA model (p and q) it is important that we perform model checking in order to optimize these parameters. In general, there are two things we want to examine when diagnosing our model, 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. Therefore, if the model is found to be overfit, it most likely means the model is more complex than it needs to be and steps should be taken to re-fit it. The latter deals with residual errors, *i.e.*, the differences between experienced values and estimated values. In an ideal model, residuals would resemble what is known as "white noise" or more commonly, a Gaussian Distribution. This means that the average of all the residuals is zero and that the distribution shape of the variances is symmetrical. Diagnosing residual errors also happens to go hand-in-hand with any overfitting issues described previously. This is because without the ability to generalize to other datasets, your model will have large prediction errors. Therefore, if you re-fit the model to be more generalizable, issues with residual errors will start to resolve themselves.

4.2 Multiple Regression Analysis

The second component of our TVA approach deals with runtime 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 a highly unlikely phenomena because certain events have different outcomes depending on the surrounding environment. For example, a self-driving car may be testing how many feet it takes to stop from 60 mph when the road is wet and when the road is dry. Intuitively, this will lead to different results nearly every time. Therefore, the self-driving car must have a way to model it's surrounding environment (the weather) so it can accurately plan for how long it will take to stop.

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 do 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, for many of the same reasons we did not move forward with the HMM approach for time series forecasting, 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 runtime predictions of tactic latency and cost. To explain, let's consider a classic regression model:

$$y(x, w) = w_0 + \sum_{j=1}^{M-1} w_j x^j = w'x \tag{4.3}$$

where $x = (x^1, ..., x^M)'$ with $x^0 = 1$ and x^j being the j-th feature of the data point. Given a set of N training data points $(x_1, ..., x_N)$ 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:

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, w) - t_n\} \tag{4.4}$$

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

$$w^* = (X'X)^{-1}X't \tag{4.5}$$

where X is the design matrix whose rows correspond to the feature vectors of the data points and $t = (t_1, ..., t_N)'$ denotes the observed response values of N data points. After minimizing Equation 4.4, we now have the estimated weights w^* in Equation 4.5, 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 like Necessary Condition Analysis (NCA)

could be used that would estimate the maximum value of the dependent value based on fixed independent variables. However, we chose to not use this approach since it would estimate the maximum latency time based on the predictor variables. We decided that this was out of scope for our approach since estimating the maximum latency time would be considered as the *worst-case scenario* for the tactic, and that this could be more appropriately addressed in future work.

4.3 TVA Workflow

TVA combines an ARIMA time series model with a Multiple Regression Analysis model to improve the self-adaptive process. In doing this, the system is able to properly maintain its defined requirements while also handling 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 requirements that are being monitored and any calculations that are made. For instance, req represents the current requirement that we are analyzing, while $resp$ represents the response we obtain from performing the actual analysis. Furthermore, the $loop$ of the workflow represents the body, or core of TVA, where we actually perform our process on the components defined in $procedure$.

Algorithm 1 TVA Workflow

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

Now that we have examined the specifics of how our TVA implementation works with

both ARIMA and MRA, let's look at a concrete example that examines the two main steps of our TVA process as shown in the pseudo code. For this, we will continue to use the self-adaptive web service example mentioned previously in Section 3.1.

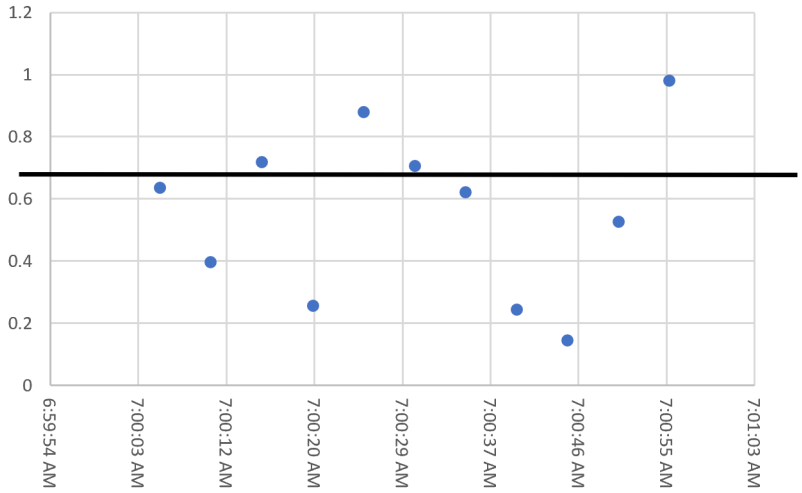

Figure 4.1: Self-Adaptive Web Service Recording Response Time Every 6 Seconds

Step 1: Monitoring Requirements

The first step in our TVA approach is to gather and monitor the requirements defined in the SLA. For this, let's consider a self-adaptive web service that has been configured to record response time every X seconds, or six seconds as in our case shown in Figure 4.1. If the system has an SLA requirement that states, "*response time should remain below a 0.7 second threshold*" (denoted by the black horizontal line), we must have a way to know when the threshold will be surpassed. Unfortunately, if we were using the majority of current self-adaptive processes that act in a "reactive manner", the system would not adapt to a surpassed threshold until it is actually broken. This is not an appropriate way to handle requirements, as the system is essentially configured to ignore problems until they become problems. Therefore, we want to transition the system from a "reactive" state to a "proactive" state by applying the time series ARIMA model discussed in this section.

Once we have analyzed a requirement with the ARIMA model approach, (Algorithm 1,

Line 5), the anticipated future value for that requirement is returned. For example, in the self-adaptive web service when the response time requirement is analyzed with ARIMA, the "response" is the forecasted value for response time. Since we are dealing with predictions and not explicitly known values, it is also possible that if the predicted value is within a certain range of the requirement threshold, the system could still choose to adapt. Regardless, by allowing the system to act in a proactive manner it is clear how requirements can be more appropriately monitored and maintained. Therefore, if the analysis determines that the requirement may be broken in the future or is close enough to being broken, we continue with the rest of our TVA approach.

Step 2: Tactic Latency and Cost Prediction

When the system has found itself in a scenario where it needs to adapt, whether it be because a requirement *will* or *may* be broken, our approach then uses regression analysis to predict tactic latency and tactic cost to aid the decision-making process. For example, if our ARIMA analysis from step 1 determines that the self-adaptive web service needs to adapt, our TVA approach then predicts tactic latency and cost for all available adaptation tactics. This could mean predicting how long and how much it would cost to activate some other kind of additional computing resources so response time could be lowered (additional servers etc.). We are able to make these predictions by implementing a Multiple Regression Analysis (MRA) model. The benefit of strictly looking at predicting tactic latency and tactic cost is that this can be easily adopted into many decision-making process that already exist. For example, self-adaptive systems have many different ways of deciding what they should do when it comes time to adapt. Some may simply select the tactic with the shortest latency, or others might build out a decision tree based on multiple tactics. However, the two driving forces of these decision-making processes are typically tactic latency and tactic cost. Therefore, our approach allows for more informed input into the decision-making process, thus leading to better decisions by the system.

Chapter 5

VALET Tool

Further contributions of this work include 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 [11, 6], 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, our VALET tool provides tactic volatility data in the form of latency and cost.

To gather data for our current and future work, VALET is comprised of two components: the Raspberry Pi 3 experiments and what we call the "grepping" experiments. In both these components, we were able to obtain both tactic latency and cost data that experienced volatility. Specifically, tactic latency within VALET was recorded as the time it took from when the initiating command was executed, until the operation was fully completed. To determine the cost (energy usage) of the experiments, we recorded the energy units used in order to perform the necessary operation. We will first discuss in detail how the Raspberry Pi experiments are conducted in VALET.

The first experimental component of VALET deals with two physical Raspberry Pi devices, with one Pi acting as the operator and the other as the measurement instrument. Within the operating device is how we gathered tactic latency data. For this, the Raspberry Pi downloaded an identical 75 MB file at one minute intervals from three different download mirrors across the world and recorded how long it took each time. This is what we refer to as the "operation" in these experiments. Although it recorded a substantial amount

of tactic latency data for our evaluation process, it too provided tactic cost data; which was it's biggest contribution. Since the Raspberry Pi also required energy for normal system operations in addition to the specific tactic operations being measured, we had to use a multi-device configuration for our evaluations. Mainly, this was done because it enabled us to limit any impact that concurrent processes might have on the defined processes, thus supporting collected metrics to more accurately reflect those caused by the tactics. Furthermore, this configuration enabled us to easily record the energy usage required for each operation. To determine actual energy usage from performing the operation, we took the average usage for the Raspberry Pi when it was performing the operation compared to when the Raspberry Pi was idle. This was all done by the "monitoring" device, as it collected and monitored the operating device while it performed the operation.

The second kind of experiments we performed in VALET were the grepping experiments, which obtained tactic latency data through a different kind of file download process compared to the Raspberry Pi. With these experiments, we downloaded an identical zip file from three different locations across the world. Those are: United States (Boston, MA area), Canada (Toronto, ON area), and Germany (area unknown). We then performed a search on the contents for specific keywords to simulate the idea of not only having to gather more information, but to also search for the information that you need. Overall, we performed this operation at one minute intervals over the course of an entire day. This created over 1,500 records of tactic latency data. The specific details of this process are detailed below:

1. **Download/Unzipping File:** The initial step is to repeatedly download and unzip an identical file from several servers located around the world. Performing this action on several locations introduces real-world volatility into our dataset, a necessity for our evaluation of TVA. This volatility may be caused by numerous factors including server availability, network congestion, and internet speeds.

2. **File Contents Search:** After the file is unzipped, a simple *grep* command searched the extracted contents for a specific string. This task was performed since it emulates

not only a system having to gather data, but also searching for the specific data it needs.

3. **Recording Latency:** We considered latency to be the time it took to download the file, unzip it, and search its contents.

Figure 5.1 represents a portion of the time series data gathered from our experiments, 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. This was essential for our evaluation because it enabled us to stress test our process and further evaluate the robustness of our TVA approach.

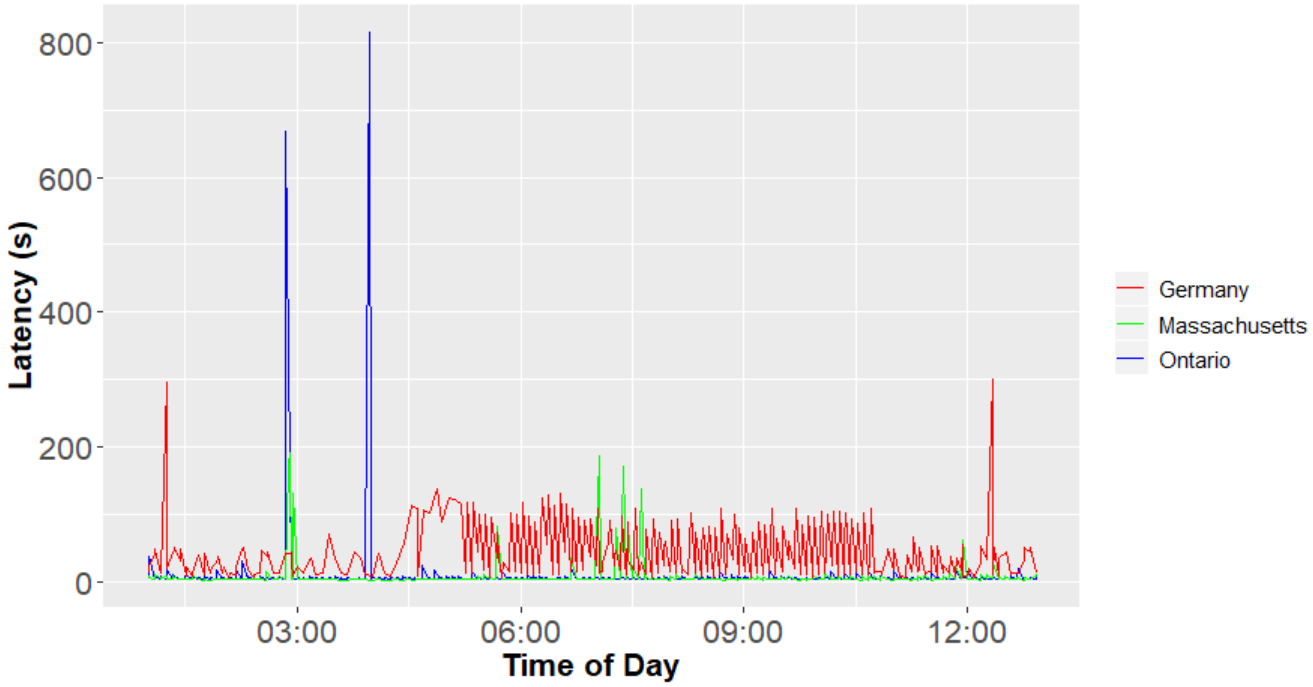

Figure 5.1: 9-Hour+ Snippet of Latency Data

Although the steps that VALET took to gather data seem rather intuitive, they were vitally important for our evaluation process. This is because they emulate realistic, usable adaption scenarios in real-world self-adaptive systems. For example, searching for specific information could represent the the self-adaptive web service example querying for user

account info, other types of content to show, and any other scenarios where it has to “look” for something to aid the self-adaptive process. Furthermore, since these operations are being performed using actual servers over the internet, this helped us to create unpredictable real-world latency volatility, as the time it takes to download a file over the internet can depend heavily on factors out of the systems control.

Chapter 6

Evaluation

To systematically evaluate TVA, we conducted multiple experiments that emulated real-world adaptation scenarios. In doing this, we were able to gather real-world data with real-world volatility. Furthermore, this means our evaluation and conclusions/analysis are not based on theoretical data, providing our work with a much more robust evaluation process. In this evaluation of TVA, addressed the following research questions:

RQ 1. 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 and tactic cost on the self-adaptive process. To accomplish this, we simulated a scenario where tactics did not behave consistently and the system did not have the ability to predict how each tactic would behave.

RQ 2. How effective is ARIMA in allowing the system to monitor system requirements? In our experiments involving time series analysis, we demonstrate that using time series forecasting with ARIMA helps the system to become more proactive in maintaining requirements defined in the SLA. To accomplish this, we examined the time series data that was collected and ARIMA's ability to predict future values.

RQ 3. How effective is MRA in predicting tactic latency and cost at run-time? In evaluating our regression model, we demonstrate that strong predictive power exists throughout our experiments when estimating tactic latency and tactic cost, even when faced with varying amounts of volatility in the gathered data. We also show that this predictive power remains when there exists little to no volatility.

RQ 4. 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.

6.1 Experimental Design

To thoroughly evaluate TVA, we created an experimental process that enabled us to gather real-world tactic latency and cost, while also experiencing real-world volatility. Furthermore, to automate and enhance these experiments, we created the previously discussed tool *VALET*. This allowed us to modify and develop these experiments with relative ease and make the data it collects available for future research. To reiterate, the data used for the evaluation of TVA was collected through the following two experiments in VALET:

Raspberry Pi Experiments

The first way we gathered real-world data was through a two-day Raspberry Pi experiment. In this experiment, we downloaded an identical file at one minute intervals from three different download mirrors across the world and considered the latency of this action to be how long it took the file to successfully download. Not only were we able to collect tactic latency data in these experiments, but also equal amounts of tactic cost data. In self-adaptive systems the cost of executing a tactic can be of significant concern to the decision-making process, thus playing a crucial role in the adaptation process. Furthermore, just like latency, it too can also experience volatility in real-world systems. Therefore, these experiments were of equal value to our work as those performed for obtaining latency data.

Grepping Experiments

The second way we gathered data with VALET was through a one-day experiment that created over 1,500 records of tactic latency data for our evaluation of TVA. In these experiments, we again used three different download mirrors from across the world (Germany, Ontario, United States). However, instead of downloading a single file, we downloaded a ZIP. Furthermore, we also searched it's contents for specific keywords to emulate a tactic that gathers information from an outside source and then searches for the specific pieces of information it needs.

6.2 Experimental Data Analysis

To analyze the results found from our experiments, we used the statistical tests of Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) to evaluate the systematic benefits of our TVA approach. These tests allowed us to determine TVA's ability to reduce uncertainty and increase the effectiveness of the decision-making process through prediction accuracy of our time series and machine learning models. The methods used are discussed in this section.

Root Mean Squared Error

To analyze and further quantify the results found from our experiments, we used the Root Mean Squared Error (RMSE) statistic as a measure of predictive power. Specifically, we used RMSE to quantify prediction error within our response variables, since RMSE puts more weight on larger prediction errors, while also making smaller prediction errors even smaller. RMSE also quantifies this error regardless of the error being in a positive or negative direction.

$$RMSE = [\sum_{i=1}^{N}(y_i - t_i)^2/N]^{1/2} \tag{6.1}$$

As shown in Equation 6.1, RMSE represents the quadratic mean of the differences between the estimated values and observed values, or the *residuals*, across multiple testing data points. It serves to aggregate the magnitudes of prediction errors, so the resulting values can be used as a measure of predictive power. Possible values for RMSE range from $0 - \infty$ due to the squared summing of residuals. Therefore, in our analysis, we use RMSE to concretely tell us how our models perform in terms of predictive ability.

Mean Absolute Error (MAE)

The final analysis method we used in evaluating our approach is Mean Absolute Error (MAE). Like RMSE, MAE can be used as another measure of predictive power. However, the main difference in these tests is that MAE examines the average absolute error, suggested by its name.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{6.2}$$

As shown in Equation 6.2, we can see the main difference between it and RMSE. Where RMSE would take the squared sum of differences over a dataset to determine prediction accuracy, MAE looks at the absolute value of these differences. Furthermore, MAE is much more commonly used when looking at forecasting errors in time series analysis [15]. Therefore, we use MAE as our measure of prediction accuracy when examining forecasting errors within our ARIMA model.

Chapter 7

Results

The results from evaluating our proposed TVA approach are the following:

RQ 1: How does not accounting for tactic volatility effect the decision-making process?

We first explored how not accounting for tactic volatility can negatively impact the self-adaptive system. This was accomplished by performing a proof of concept evaluation using the statistical tool R, in which we emulated the negative effects of poor decision-making by a self-adaptive system. We began by defining two tactics that had arbitrary costs associated with their latency times. Meaning, so long as the tactic took X amount of time to execute, it cost the system C units of cost. For these simulations, the "cost" values can represent the same resources mentioned throughout this work that a self-adaptive system may consume during tactic execution. For example, the cost of executing a tactic may be the consumption of a certain amount of RAM, which may end up slowing other processes running in the system.

Table 7.1: 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 the normal distribution, while the tactic with the lower cost 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, however, that both tactic latency and cost do not follow either a normal distribution or skewed distribution, but because these distributions are two extremes, it was sufficient for our proof of concept simulations. Table 7.1 shows the characteristics of each tactic.

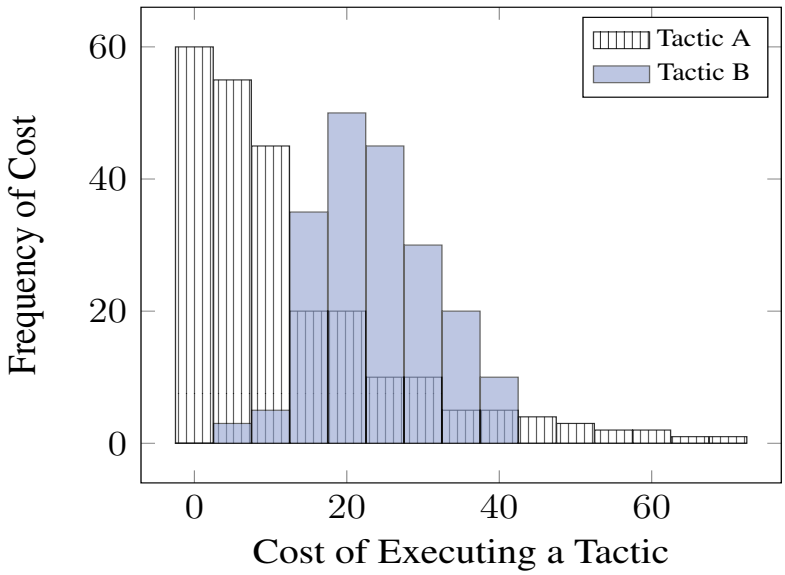

Figure 7.1: Overall Costs of Executing a Tactic in R Simulations

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 the latency times differ. This also allowed us to gather overall cost values that could be used to compare how varying data distributions can effect 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 7.1). However, tactic B's data

followed a normal distribution, which is highly unlikely occurrence for 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 were able to successfully demonstrate that accounting for tactic volatility is essential in self-adaptive systems, especially when the system is known to have unpredictable behavior.*

RQ 2: How effective is ARIMA in allowing the system to monitor system requirements?

To evaluate our ARIMA model for time-series forecasting, we used the energy data collected from the Raspberry Pi experiments. Although we had been using a self-adaptive web 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 the Raspberry Pi was downloading the file. Therefore, the data used in this research question is strictly the energy usage fluctuations when the device was idle and *not* downloading the file.

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 setting up 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 atleast once. After performing this validation, we then calculated both RMSE and MAE values to determine the predictive ability of the ARIMA model.

Shown in Figure 7.2 are the MAE results from our ARIMA model compared to our initial Hidden Markov Model. As shown, using an ARIMA model on the time series data that we gathered was not only successful by itself, but also better than that of the HMM model. We believe this occurred since the data we collected does not allow for a full treatment of a Hidden Markov Model to be properly implemented. Meaning, we did not

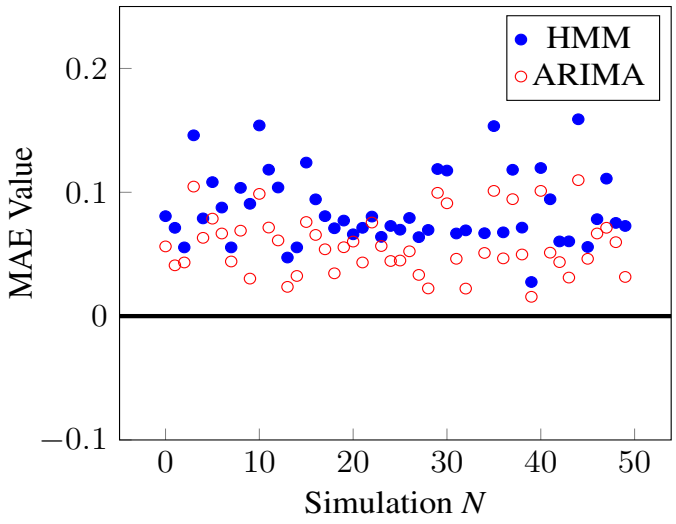

Figure 7.2: MAE Values Over 50 Experiments Using ARIMA vs HMM for Time Series Forecasting

have enough features in our model to obtain the full capabilities of HMM.

Since each experiment in our evaluation is independent from the others, no patterns should be inferred nor concluded from the graph. For example, from roughly simulation 20 to 40 the MAE values for the HMM model form a near horizontal line. However, this does not mean there are any underlying reasons for this. The same 50 experiments could be run again and the shape of the graph could look entirely different. What we care about most in the graph is how wide the "spread" of these MAE values are, as this tells us how stable our predictions were. Over the course of the experiments in Figure 7.2 we saw fairly stable MAE values, represented by the small range between the lowest MAE value and the highest. Furthermore, not only were MAE values consistent but they were also considerably low. Therefore, this tells us that the predictive ability of our model was both strong and consistent across all the data we gathered.

As discussed previously, when using RMSE, larger prediction errors will become "larger" and smaller prediction errors will become "smaller" – a result of the squared term. Due to this, we would expect the RMSE graph to be a little more spread out compared to the MAE graph. In examining the RMSE differences between HMM and ARIMA, our findings were

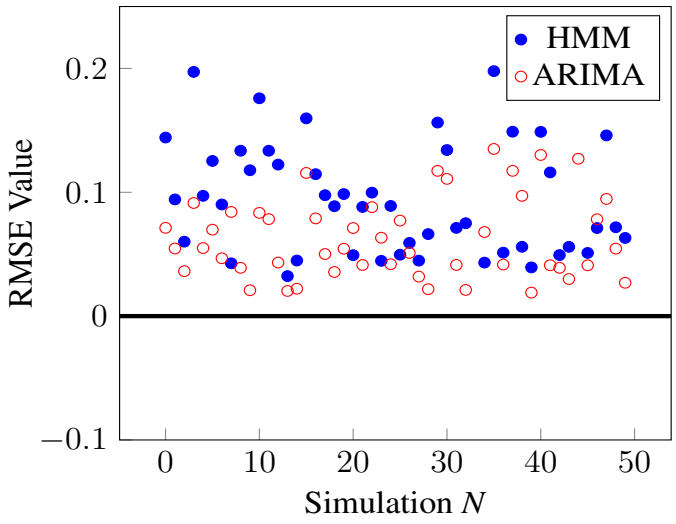

Figure 7.3: RMSE Values Over 50 Experiments Using ARIMA vs HMM for Time Series Forecasting

exactly this. As expected, ARIMA still had lower RMSE values than the HMM model. Furthermore, looking at Figure 7.3 closely, we can see how the RMSE values have a wider spread than the MAE values reported in Figure 7.2. However, this does not mean there was less predictive power or that the model is less useful. The reason we report on both is 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. Therefore, they may deem RMSE a more powerful statistic for determining predictive ability of the model. On the other hand, 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 that examined time series data, TVA demonstrated it's ability to allow for proactive adaptations. TVA accomplishes this by applying an ARIMA time series model to predict future values for requirements, thus allowing the system to start adapting before they may become broken.*

RQ 3: 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 [33, 37, 23, 8]. 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 belief that RMSE is more appropriate, this research question does not report both RMSE and MAE values.

Unlike Research Question #2, where we only examined the energy usage for when the Raspberry Pi was idle, this research question looked only at the energy usage when the device *was* performing the file download. Had we 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.

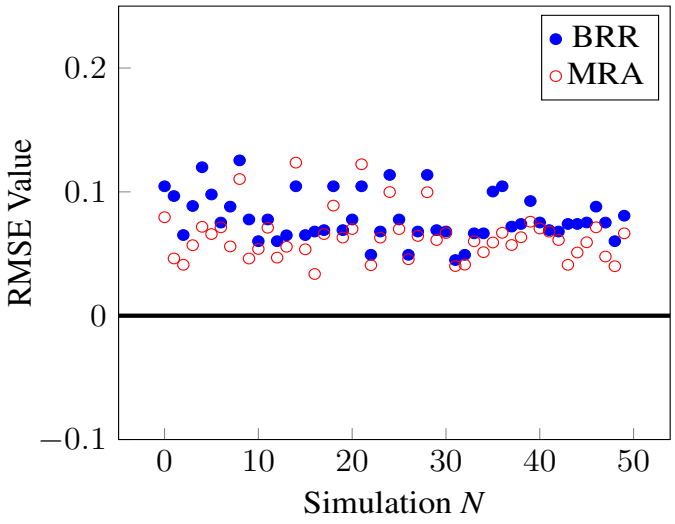

Figure 7.4: RMSE Values for Tactic Cost Predictions of MRA vs BRR

As demonstrated in Figure 7.4, we can see why MRA was 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.

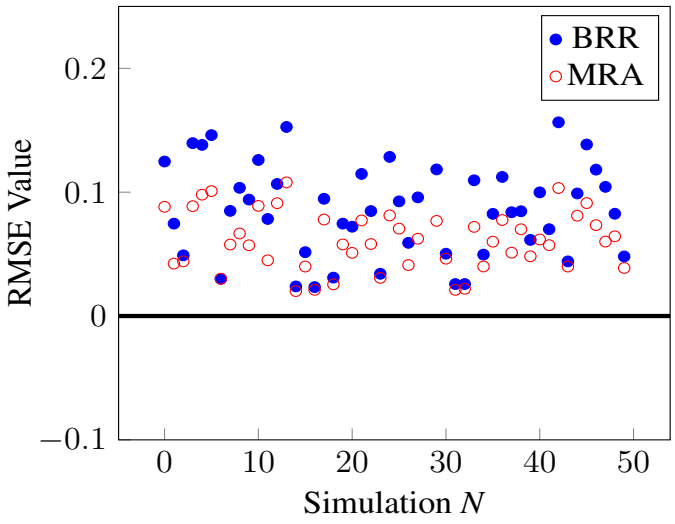

Figure 7.5: RMSE Values for Tactic Latency Predictions of MRA vs BRR

As demonstrated in Figure 7.5, we saw very similar results between the RMSE values calculated for tactic latency predictions and those calculated for tactic cost predictions. 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 seen. As mentioned, 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 (Figure 5.1). This can partially be seen in Figure 7.5 since more volatility will likely lead to larger prediction errors. Thus, 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 that MRA may be generalizable to other types of data sets, seeing consistent RMSE values regardless of the data it was modeling was imperative. Represented in both Figure 7.4 and Figure 7.5, we can see that not only were RMSE values on the lower side for both data sets, but the values were consistent. Therefore, MRA was able to handle both the volatile latency data and cost data collected for these experiments.

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.

RQ 4: Does using TVA provide substantial improvement to the self-adaptive process over simply using static values for latency and cost?

We have thus far demonstrated both the benefits and predictive ability of using an ARIMA time series model and a MRA model in a self-adaptive process. However, to provide further confidence in our TVA approach, we also compared it to the current self-adaptive processes in order to determine its concrete benefit. 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 it's prediction process for estimating future tactic behavior*

Although looking at the average for previous latency and cost values may seem like a naive comparison, this provides us with a better comparison than using a static value like current self-adaptive processes. This is because doing something as simple as taking the average of a given set of latency values can provide more predictive ability and information than considering latency to be a static value for every execution. This also takes out any ambiguity or subjectivity in defining what the static value should be for comparison. Therefore, comparing our TVA process to taking the average of previous values is appropriate for evaluation purposes. As with Research Question #2, we also report both MAE and RMSE values for model comparison.

As shown in Figure 7.6, TVA was able to substantially outperform the baseline approach. 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 stayed fairly consistent for TVA, the baseline approach saw a much wider spread; a direct result of only using

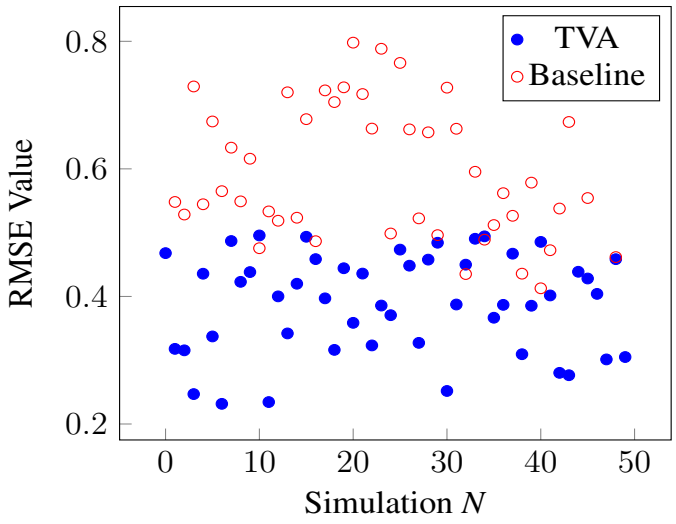

Figure 7.6: Baseline Value Approach vs TVA - RMSE Differences

a static value for latency and cost. Although there were cases where the baseline approach did outperform TVA (seven in total out of 50 experiments), the differences in these values were fairly negligible.

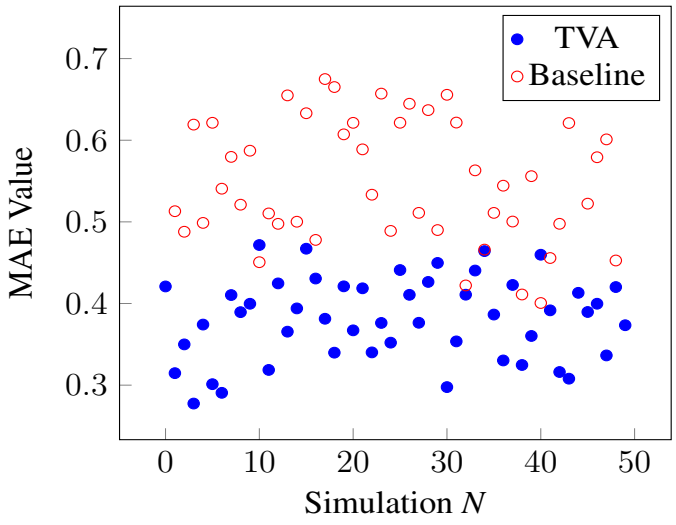

Figure 7.7: Baseline Value Approach vs TVA - MAE Differences

In examining the MAE values shown in Figure 7.7, we found that MAE was not as much of a "man-in-the-middle" result as we would have expected. In Research Question #2, it is fairly evident that RMSE made larger prediction errors larger, and smaller errors smaller.

However, in this research question, those kinds of results are not as evident. Looking at 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 happened because the baseline model's prediction method mentioned previously 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: *Not only do these findings show that TVA is beneficial for decision-making processes in self-adaptive systems, but that it is significantly more useful than assuming tactic latency and tactic cost to be static values. TVA accomplishes this through a combination of time series forecasting with ARIMA and a MRA-based prediction model for tactic latency and cost.*

Chapter 8

Discussion

Building off of the results discussed in Chapter 7, not only can our work be summarized with the following two categories, but so can its benefit for future research and development in self-adaptive systems.

Predictable Behavior

Being able to accurately estimate how tactics are going to behave is crucial for the decision-making process. Without this ability, self-adaptive systems could behave erratically possibly leading to the wrong tactics being executed. However, being able to accurately estimate how a tactic is going to behave is much more than simply monitoring previous tactic latency and cost values. For example, a self-adaptive web service could monitor ten response time values and then make a prediction of what the next value is going to be. Yet, if the prediction mechanism is something as simple as taking the average of the previous values, it is likely that there will be significant error in the estimate. Thus, it is imperative that we give self-adaptive systems the ability to model its surrounding environment. Not only will this allow the system to more accurately estimate tactic latency and tactic cost, but it also provides a "hands-free approach" since these kinds of models can train themselves. This means that so long as a system records and stores data for the model to build from, there is no human intervention required to update the model periodically. In our work, we were able to demonstrate this through the use of Multiple Regression Analysis, a regression model that proved it was capable of predicting tactic latency and cost values, but was also able to handle volatility that existed within previously experienced data.

Reliability

For self-adaptive systems that are driven by their requirements, the decision-making process must be tailored to them. Otherwise, the system will continue to experience less than optimal performance by infrequently addressing the most important requirements of the system. For some systems, maintaining these requirements may be fairly intuitive because some may depend on others. For example, in the self-adaptive web service there could be the aforementioned requirement that states that response time is not allowed to go over a certain threshold, along with one that states that 90% of available content must be shown to the user at all times. In reality, if content levels were dropping below 90% due to response time issues from the system's servers, the service would address the response time requirement first. Then, if there are still problems with the levels of content being shown to the user, the system could perform further adaptions to maintain the required 90% level of content. However, for systems where maintaining requirements is not as intuitive as this example, being able to appropriately prioritize, monitor, and analyze them becomes that much more imperative.

Fortunately, prioritizing requirements can be done through the system's SLA, where the system's engineers can clearly define what requirements have priority over others. Furthermore, monitoring and analyzing these requirements can be done through time series forecasting, as monitoring certain values over a period of time results directly in the collection of valid time series data.

In our work, we demonstrated the need for tailoring decision-making processes to the requirements defined in the SLA by including time-series forecasting (ARIMA model) in our approach. By doing this, we were able to show that by accurately knowing the potential future values for requirements defined in the system, the system is able to make more informed adaptations. Furthermore, because this prediction is done with time series forecasting, *i.e.*, the predicted value is in some determined time interval T in the future, it also allows the system to become more *proactive*, instead of *reactive* when monitoring requirements. For example, a web service that is reactive would recognize that response

time is over a certain threshold as it happens and adapt accordingly. However, a web service that acts more proactively through the use of an ARIMA model as done in this work, has the ability to start adapting before response time actually becomes an issue/surpasses a threshold. Thus, the gives the system even more of an ability to make more informed decisions and perform more reliable adaptations.

From evaluating our work through different types of experiments and varying real-world datasets, we have shown that TVA can significantly improve the decision-making processes of a self-adaptive system. Additionally, these improvements are capable of reducing any uncertainty that may exist, which has already been shown to be detrimental [3, 29, 10]. Therefore, this has significant implications for both researchers and developers in the realm of self-adaptive systems. As improvements of predictable behaviour and reliability become more necessary, we believe our work provides a solid foundation and solution to the limitations that exist in current self-adaptive system processes.

Chapter 9

Limitations 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 the initial phases of incorporating our TVA approach, using machine learning models may not be of substantial improvement over current processes. This is due to the fact that machine learning depends heavily on the existence of data. Therefore, tactic attributes may still be predetermined due to a lack of prior data to consider. Additionally, observing tactic volatility will not be practical for addressing several types of infrequently utilized tactics. For example, tactics that execute infrequently will not have adequate data available for machine learning prediction models. This means that traditional processes will still be necessary due to a lack of relevant available data.

In many systems, tactic cost may be an ambiguous and tough-to-define measure. 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 the physical in the device. Such a cost is hard to quantify in most situations. Therefore, using TVA the notion of cost must be limited to a value that is reasonably easy to quantify.

Although TVA provides a way of monitoring requirements defined in the SLA for self-adaptive systems, not all requirements are necessarily measurable with time-series analysis. For example, a system may have the requirement that it must be available 99.99% of the

time. This is not something that could be measured or predicted with time-series analysis, rather it would have to be handled through fault detection techniques in the systems architecture. Therefore, adopting our TVA process would mean taking into consideration that requirements monitoring with TVA must be of time-series form.

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 requirement 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 requirements 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.

Tactic-based decisions frequently do not occur independently from other tactics. Often, tactics occur simultaneously with other tactics and this may impact other tactic-based decisions [28, 30]. Future work should be done to examine how poor tactic-based decisions due to tactic volatility can affect both subsequent and concurrent tactic-based decisions. These results have the possibility of further demonstrating the importance of considering tactic volatility in the decision-making process.

Although we have demonstrated the benefits of our TVA approach with real-world experimental data, we have yet to implement it into any physical devices. This is separate from using physical devices to collect data, however. 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 services.

ARIMA demonstrated its ability to perform time series forecasting, however, no mechanisms are in place to quantify any uncertainty within these predictions. Future work could be done to include confidence intervals around predictions made by ARIMA. For example, instead of just predicting a single point value for a requirement, we could predict a range that states something like 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 it's predictions, therefore providing the decision-making process with more information.

Now that we have shown the benefits of not using static values for tactic latency and cost, future work must be done to compare TVA to more machine-learning and time-series forecasting models. In this investigation some of the important factors to examine may be: computing resources (as more complex machine-learning models will tend to require more computational resources) and models that perform well regardless of the availability of data. Furthermore, examining the use of time series models that have "longer memory" should also be examined to determine the concrete differences between the two and their possible uses in self-adaptive systems.

Chapter 10

Conclusion

In this work, we proposed a Tactic Volatility Aware (TVA) process that is able to account for tactic volatility in multiple ways. TVA first uses time-series forecasting to monitor system requirements, ensuring that the system can be proactive when adapting to its environment while maintaining it's most important requirements defined in the SLA. Next, TVA uses regression analysis to predict tactic latency and tactic cost at run-time helping to improve the decision-making process of the system. As the world begins to use more and more autonomous systems everyday, addressing tactic volatility must become a first-class concern. Without this ability, systems will not be able to perform at their peaks, possibly resulting in severe consequences.

Bibliography

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

[2] Yuriy Brun, Giovanna Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Software engineering for self-adaptive systems. chapter Engineering Self-Adaptive Systems Through Feedback Loops, pages 48–70. Springer-Verlag, Berlin, Heidelberg, 2009.

[3] Javier Cámara, David Garlan, Won Gu Kang, Wenxin Peng, and Bradley Schmerl. Uncertainty in self-adaptive systems. *MONTH*, 2017.

[4] Javier Cámara, Gabriel A Moreno, and David 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.

[5] Javier Cámara, Gabriel A Moreno, David Garlan, and Bradley Schmerl. Analyzing latency-aware self-adaptation using stochastic games and simulations. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(4):23, 2016.

[6] 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.

[7] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, December 2006.

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

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

[10] Naeem Esfahani and Sam Malek. Uncertainty in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*, pages 214–238. Springer, 2013.

[11] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. Adaptive, model-driven autoscaling for cloud applications. In *11th International Conference on Autonomic Computing ({ICAC} 14)*, pages 57–64, 2014.

[12] David Garlan, Bradley Schmerl, and Shang-Wen Cheng. *Software Architecture-Based Self-Adaptation*, pages 31–55. Springer US, Boston, MA, 2009.

[13] S Hall and Dimitros Asteriou. Arima models and the box–jenkins methodology. In *Applied Econometrics*, pages 265–286. Palgrave MacMillan, 2011.

[14] Nikolaus Huber, André Hoorn, Anne Koziolek, Fabian Brosig, and Samuel Kounev. Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments. *Serv. Oriented Comput. Appl.*, 8(1):73–89, March 2014.

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

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

[17] Pooyan Jamshidi, Amir M. Sharifloo, Claus Pahl, Andreas Metzger, and Giovani 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.

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

[19] M Jeroen Van Der Donckt, Danny Weyns, M Iftikhar, and Ritesh Singh. Cost-benefit analysis at runtime for self-adaptive systems applied to an internet of things application, 01 2018.

[20] Gueyoung Jung, Kaustubh R Joshi, Matti A Hiltunen, Richard D Schlichting, and Calton Pu. A cost-sensitive adaptation engine for server consolidation of multitier applications. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, page 9. Springer-Verlag New York, Inc., 2009.

[21] Gueyoung Jung, Kaustubh R. Joshi, Matti A. Hiltunen, Richard D. Schlichting, and Calton 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.

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

[23] Cody Kinneer, Zack Coker, Jiacheng Wang, David Garlan, and Claire Le Goues. Managing uncertainty in self-adaptive systems with plan reuse and stochastic search. 2018.

[24] Cody Kinneer, David Garlan, and Claire Le Goues. Information reuse and stochastic search: Managing uncertainty in self-* systems. 2019.

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

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

[27] 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.

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

[29] Gabriel A Moreno, Javier Cámara, David Garlan, and Mark Klein. Uncertainty reduction in self-adaptive systems. 2018.

[30] Gabriel A Moreno, Javier Cámara, David Garlan, and Bradley 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.

[31] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. Flexible and efficient decision-making for proactive latency-aware self-adaptation. *ACM Trans. Auton. Adapt. Syst.*, 13(1):3:1–3:36, April 2018.

[32] Gabriel A. Moreno, Alessandro V. Papadopoulos, Konstantinos Angelopoulos, Javier Cámara, and Bradley 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.

[33] Arthur Rodrigues, Ricardo Diniz Caldas, Genaína Nunes Rodrigues, Thomas Vogel, and Patrizio Pelliccione. A learning approach to enhance assurances for real-time self-adaptive systems. *arXiv preprint arXiv:1804.00994*, 2018.

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

[35] Bradley R. Schmerl, Javier Cámara, Jeffrey Gennari, David Garlan, Paulo Casanova, Gabriel A. Moreno, Thomas J. Glazier, and Jeffrey M. Barnes. Architecture-based self-protection: composing and reasoning about denial-of-service mitigations. In *HotSoS*, 2014.

[36] 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.

[37] Sven Tomforde, Stefan Rudolph, Kirstie Bellman, and Rolf 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.