Simulation-based Retrieval of Adaptation Knowledge

Alexander Frömmgen    Patrick Wagner    Alejandro Buchmann
DVS, TU Darmstadt, Germany
{froemmgen, wagner, buchmann}@dvs.tu-darmstadt.de

ABSTRACT
Today’s networking applications have to operate under changing environmental conditions. Web browsers on mobile devices, for example, are facing rapidly changing network conditions. Even though browsers and network stacks provide fine grained configuration options, it is challenging to find configurations which perform well in these changing environments.

In this paper, we propose to adapt the configuration to changing network conditions at runtime. Based on monitored network properties, a k-nearest neighbour classifier predicts suitable configurations. For a first evaluation, we trained the classifier for a web browsing scenario with Firefox in Mininet simulations. The predicted configurations outperform all static configurations and reduce the average page load time by 1.5 seconds, realizing 68% of the possible improvement of an optimal prediction.

1. INTRODUCTION
Users expect network applications to work under different network conditions. Especially applications on mobile devices with multiple access technologies are facing rapidly changing network conditions regarding bandwidth, latency, and packet drops. Optimizing the application for all possible environments at the same time is challenging. Applications such as modern web browsers provide a multitude of configuration parameters [14], e.g. the maximum number of parallel TCP connections and the usage of HTTP pipelining [6]. However, as these configurations are static, today’s applications are statically configured in a highly dynamic environment.

In this paper, we propose to leverage the already existing configuration parameters and adapt the application at runtime to always execute a suitable configuration. Therefore, we predict the optimal configuration out of a huge configuration space based on the currently monitored behaviour of the environment (Figure 1 top). To manage the inherently complex dependencies, we propose to use a classifier for the prediction and automatically train the classifier offline for a wide range of environments. For a first evaluation of our approach, we executed a typical web browsing example with the Firefox browser in 175,500 Mininet [11] simulations with different network conditions to train a k-nearest neighbour classifier (Figure 1 bottom).

Using simulations to learn desired behaviour offline was successfully applied by Winstein et al. [20], who automatically generated congestion controls which outperform analytically derived algorithms. Our presented approach follows the MAPE-K cycle of autonomous computing [15] for managing adaptive behaviour. Compared with existing approaches, such as the Fossa [7, 8] framework, our approach uses configuration parameters of real world applications and is not restricted to a rule based representation of the adaptation logic.

2. PREDICTION ENGINE
The prediction engine maps a set of monitored attributes \( \langle \text{mon\_attr}_1, \ldots, \text{mon\_attr}_n \rangle \) to the target configuration. The monitoring attributes should be chosen carefully, as they are essential for a high classification performance. For applications which use TCP, connection details such as the round trip time and the experienced packet drops are obvious candidates. They are easily accessible from the socket state [16].
Regarding the choice of the classifier, first evaluations showed that a k-nearest neighbour classifier outperforms both tree based classifiers and support vector machines. The k-nearest neighbour classifier uses a distance function to choose the target configuration based on the majority of the k nearest trained samples [18].

3. TRAIN THE ENGINE

The classifier uses training samples with a set of attributes and their target label for training. In the web browser example, the classifier minimizes the time from starting the HTTP request until the document and all dependent resources are loaded. Thus, the label is the configuration which leads to the lowest page load time regarding the Navigation Timing interface [9].

As the classifier requires a huge amount of training data, we automatically generate these in Mininet simulations with different network conditions. Mininet allows to run unmodified binaries and supports all relevant network parameters for bandwidth, delay, and loss. We developed a mass simulation environment to run simulations on a multitude of server instances in parallel. Additionally, we extended Mininet’s Python API\(^1\) to natively support both Apache 2.4.7 and Firefox 39.0. Our extension supports the manipulation of configuration parameters, e.g., `setFirefoxParameter(h1, ‘network.http.pipelining’, ‘true’), and to control a headless running Firefox from the Mininet API.

4. EVALUATION

For the evaluation, we implemented the presented frameworks and optimized the page load time of the front pages of three popular web pages (Google, Amazon, and Wikipedia). As configuration parameters, we used three congestion controls (Reno, Cubic, and Vegas [10, 2]) and the HTTP pipelining modes of Firefox: `off`, `on`, and `aggressive`. HTTP pipelining allows the client to send multiple requests over a persistent connection without waiting for a response and thereby reduces the number of required round trip times for multiple requests [17]. We tested 650 network conditions for access networks as reported in the literature: bandwidth between 0.384 and 100 Mbit/s, latency between 0 and 300ms, and packet drop rates between 0 and 5% [1, 3, 4, 5, 12, 13, 19]. Each simulation was repeated 10 times, leading to $3 \cdot 3 \cdot 3 \cdot 650 \cdot 10 = 175,500$ simulations.

The evaluation shows that `no pipe/cubic` is the optimal configuration for 26% of all tested network conditions (Figure 2a). `No pipe/vegas` and `pipe/vegas`, however, have the lowest average page load time 13.7s of all configurations (Figure 2b). Thus, they should be chosen for a static solution. The trained prediction engine has an average page load time of 12.2s. The optimal solution, which always executes the optimal configuration, would lead to 11.5s. The prediction engine easily outperforms static solutions, and reduces the average page load time by 1.5s of the possible 2.2s, realizing 68% of the possible improvement of an optimal prediction. The CDF shows that the predicted solution improves especially medium and high page load times (Figure 3).

5. CONCLUSION AND OUTLOOK

This paper proposed a simulation-based retrieval of adaptation knowledge for networking applications. We showed that runtime adaptations between Firefox configurations and congestion control algorithms reduce the average page load time. For future work, we plan to apply our framework on more applications and to evaluate the classifiers in real world scenarios.

Acknowledgments

This work has been funded by the German Research Foundation (DFG) as part of projects A02 within the Collaborative Research Center (CRC) 1053 – MAKI.

---

\(^1\)Available at http://dvs.tu-darmstadt.de/simunu
6. REFERENCES


