Effectiveness of pre-computed knowledge in self-adaptation - A robustness study

Max Korn, Philipp Chrszon, Sascha Klüppelholz, Christel Baier, and Sascha Wunderlich*

Institute of Theoretical Computer Science, Technische Universität Dresden, Germany

Abstract. Within classical MAPE-K control-loop structures for adaptive systems, knowledge gathered from monitoring the system and its environment is used to guide adaptation decisions at runtime. There are several approaches to enrich this knowledge base to improve the planning of adaptations. We consider a method where probabilistic model checking (PMC) is used at design time to compute results for various short-term objectives, such as the expected energy consumption, expected throughput, or probability of success. The variety PMC-results yield the basis for defining decision policies (PMC-based strategies) that operate at runtime and serve as heuristics to optimize for a given long-term objective. The main goal is to apply a robust decision making method that can deal with different kinds of uncertainty at runtime. In this paper, we thoroughly examine, quantify, and evaluate the potential of this approach with the help of an experimental study on an adaptive hardware platform, where the global objective addresses the trade-off between energy consumption and performance. The focus of this study is on the robustness of PMC-based strategies and their ability to dynamically manage situations, where the system at runtime operates under conditions that deviate from the (idealized) assumptions made in the preceding offline analysis.

1 Introduction

The world is full of dynamic systems, which adapt their behavior depending on the prevailing conditions with the simple goal to survive or to self-optimize with respect to various criteria, e.g., energy-efficiency. Nature already provides very efficient and smart ways for the autonomous management of adaption within biological systems and chemical processes. Likewise, there are human-created technical systems that also allow for adaptation to changing conditions either within the system itself or to external changes in the environment. In particular, there is a growing importance and dependence of complex dynamic hardware/-software systems, systems of systems, and cyber-physical systems that are subject

* The authors are supported by the DFG through the TRR 248 (see `https://perspicuous-computing.science`, project ID 389792660), the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy), and the Research Training Groups QuantLA (GRK 1763) and RoSI (GRK 1907) and the DFG-project BA-1679/11-1.

to different types of changes. Recent application areas include, e.g., autonomous driving and flying, where internal and external changes can happen anytime. Internal changes are, e.g., software updates or failures of hardware/software components. External or environmental changes include, e.g., bad road or weather conditions and sudden pedestrian appearance. Adaptations may also be required in case of a shift in the prioritization of different objectives or the rebalancing of trade-offs, e.g., performance vs. energy efficiency. Typical decisions for system adaptation include, e.g., switching on and off of components, switching between management policies, (de)allocating resources, and task scheduling and migration.

The challenge of maintaining and managing complex systems efficiently at runtime often exceeds what can easily be achieved by human beings. In practice, decision making is usually based on heuristics that generally do not provide optimal or close-to optimal results. In case a multitude of demands must be met, e.g., safety, reliability, throughput, and minimizing operating costs, decision making becomes increasingly complex. A recent trend is to direct (parts of) this complex task to learning-based approaches, which may be problematic in safety-critical settings where black-box decision making does not provide insights and hence trust. Alternatively, approaches based on formal methods can be employed to enable both transparent and (close-to) optimal decision making. In this paper, we apply probabilistic model checking (PMC) for supporting the runtime decision-making. The focus will be on the robustness of PMC-supported decision making when the stochastic assumptions on the environment behavior are inaccurate or biased and the system is subject to unforeseen dynamics.

The Approach. The general goal is to optimize runtime decision making in adaptive systems regarding a given *long-term objective*, which is typically a multi-objective property addressing the trade-off between conflicting objective functions. A typical example for such tradeoff is the minimization of energy consumption combined with the maximization of performance or throughput of a system. The goal is then to balance the tradeoff appropriately according to the application specific demands. Unfortunately in practice, the assumption that a detailed system model can be built and optimal solutions regarding a given long-term objective can be computed in time is often not valid. To anyhow achieve this goal, PMC-results for various *short-term objectives* with fixed horizon (such as step-bounded or reward-bounded objectives) are computed at design time. In this step, we use an abstract and discrete-time operational system model (i.e., a Markov decision process (MDP) annotated with costs/rewards) and store PMC-results into a database that can later be queried at runtime. The decision making can adapt the system according to adaptation choices that seem optimal in regards to selected short-term objectives and in a way that potentially approximates optimal solutions for the long-term objective. So called *PMC-based strategies* are query execution plans for the database that specify how exactly PMC-results for short-term results are combined within the decision making at runtime. In this paper we propose different PMC-based strategies and quantify the quality and robustness of PMC-based decision making. For this, we identified the following critical aspects to be addressed within the experimental studies:

(A1) reducing the accuracy of the stochastic environment assumptions, as prior knowledge on the concrete environment behavior at runtime is in general not available, (A2) limiting the analysis horizon, as analyzing the system model to its full depth is practically impossible, and (A3) increasing the time between adaptations, as adaptation in each time step is unrealistic also due to the imposed adaptation costs, e.g. in terms of delays and additional energy.

Contributions. The main contribution of this paper is hence a **robustness study for PMC-supported decision making**. As a starting point we developed a stochastic operational model of a database application that runs on an adaptive hardware platform with alternative computational modes.

1. We suggest different PMC-based strategies for managing the trade-off between the energy consumption and performance (i.e., the amount of database queries that can be processed by the platform within a fixed timeframe),
2. extend Prism's [16] simulator with features to enable an evaluation of PMC-based strategies using statistical model checking,
3. carry out an extensive, simulation-based evaluation of PMC-based adaptation strategies to quantify their quality and robustness regarding aspects (A1–A3) compared to a standard heuristic for resource management, and
4. provide general guidelines on how to deploy PMC-based decision making for robust operation depending on the critical application characteristics.

An artifact is made available under `https://tud.link/8036`, together with an extended version of this paper. The artifact contains the operational model, the full tooling, a demo example and the documentation. Furthermore, it allows for reproducing all experimental results.

Related work. There is a large variety of formal methods addressing different components of the classical MAPE-K loop [21,13], namely for monitoring, analyzing, planning, executing and to gain additional knowledge. Existing approaches are closely related, but do not share the primary goal of robustness decision making. Instead, they are orthogonal and can be combined with PMC-supported decision making. Runtime verification (RV) is for example used for the monitoring of certain properties and their verification at runtime. RV is typically used for safety, reliability, security, fault containment, and recovery as well as for online system repair, but also for aiming at quantitative objectives [17]. Usually, RV does not rely on an abstract system model and model checking at runtime. In contrast, *online model checking* performs classical model checking periodically. The goal is to obtain guarantees on the near future in particular with respect to reliability (often covering safety only) [23,12], even though the model might be inaccurate or of approximate nature only. The limits of any approach that carries out model checking at runtime are determined by the time available for decision making, which is typically very short. *Incremental model checking* [14,18] hence considers an operational model for some base behavior plus models for dynamic aspects or components. This incremental model structure potentially allows for reducing the effort of model checking at runtime, as only the dynamic system parts have to be considered and combined with results from previous verification runs. The

development of approaches for PMC-supported decision making and planning in adaptive software is an emerging research field [6,4,9]. Although there has been some progress on online approaches that call a probabilistic model checker for (bounded) models at runtime [5,20,22,19], PMC is too compute-intensive for a pure on-the-fly approach when decisions cannot afford delays and have to be made almost instantaneously. Incremental approaches (e.g. [10]) try to narrow this gap in performance, but limits on what can be computed at real time while the systems executes (or waits for adaptation) still exist. Hence, there is a clear motivation to carry out any kind of complex PMC analysis at design time rather than at runtime. The work in [8] also relies on a pre-analysis of quantitative measures (only) for reliability. The results are represented and stored as symbolic expressions, namely polynomials with free parameters that are then evaluated at runtime for the actual parameter values. The authors rely on parametric model-checking techniques. The clear benefit of this is, that the model checking results can be stored very compactly (in terms of symbolic expressions), but the approach is only practical when the number of free parameters is reasonably small. The method in [11], which we rely and build on in this paper, uses pre-analyzed quantitative measures to create annotated MDPs for runtime decision making. In contrast to [8], results are stored explicitly rather than symbolically. Our approach is similar in this aspect. Technically, we rely on a database to store results for multiple state properties. The database is then dynamically queried at runtime. Nevertheless, the common assumptions of [8] and [11] are, that the measures computed in the offline analysis are fixed and the exact same as the ones addressed and optimized for at runtime and that the full MDP model is small enough to be analyzed in its full depth. In our work, we drop those assumptions and address the question of how (for large systems in particular) short-term objectives and decision policies with bounded scope on parts or fractions of the system (which can be computed at design time) can be utilized and dynamically combined into complex adaptation policies (called PMC-based strategies) to optimize for a given long-term objective. The main question we address is then the aspect of robustness of different, alternative PMC-based strategies. We also want to stress the fact, that the selected method is compatible with other (formal) methods present in a MAPE-K loop. For instance, RV can be used for monitoring and online model checking can be used periodically and even incremental to update the database in the background.

2 PMC-supported Decision Making

The classical MAPE-K control loop consists of a managed system operating within an uncontrollable environment, monitoring components (M) collecting information on the system and the environment via sensing at runtime, components to analyze the available information (A), components that generate an adaptation plan (P) from the output of the analysis components as well as the knowledge base (K), and components that execute the current plan (E) via actuators on the system under control. In this work, the system under control is typically a hardware/software

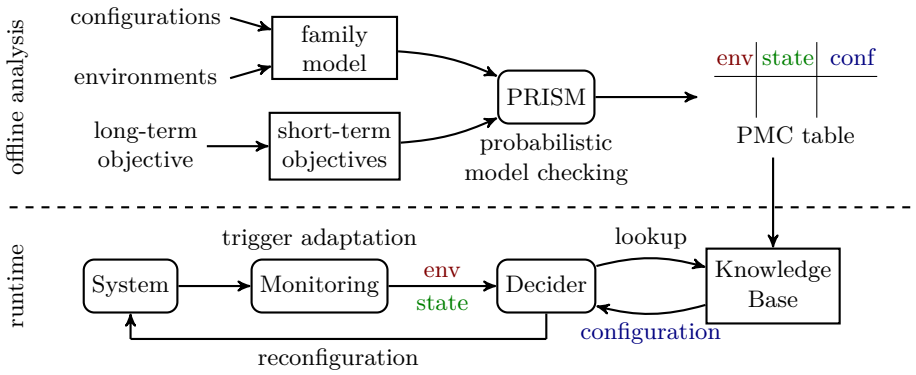

Fig. 1: Schema of PMC-supported decision making

system whereas the environment is defined by the workload assigned to the system. In the following, we outline a PMC-based planning approach for steering the system adaptation as seen in Figure 1. This approach requires an offline pre-analysis of the system for producing the knowledge base (tech. a database) used at runtime.

2.1 Model-based Offline Analysis

Starting point of our procedure is an abstract and discrete-time operational model of the system in question (i.e., a Markov decision process (MDP) annotated with costs/rewards), which combines non-deterministic choices with probability distributions over successor states. The non-deterministic choices encode possible reconfiguration actions and the probability distributions encode stochastic assumptions, e.g., on the number of newly arriving tasks. Cost and reward structures are used to annotate, e.g. energy consumption in the given configuration. The PMC-analysis then takes the model together with a property specification (i.e., in terms of a PCTL* formula (probabilistic computational tree logic)) and computes so called *schedulers*, which resolve the non-deterministic choices in order to maximize (or minimize) expected accumulated costs as well as probabilities of temporal events. [1] In order to optimize decisions regarding a *long-term objective*, which is typically a multi-objective property, the goal is to balance the tradeoff in a way that fulfills the application specific demands. In [11], the MDP model is annotated with the results (per state) which then yields the basis for the decision making at runtime, on how to best achieve the targeted long-term objective. Since this is not always practical, we focus instead on computing optimal decisions regarding *short-term objectives* such as step-bounded or reward-bounded objectives, where the analysis horizon comprises the next $n \in \mathbb{N}$ timesteps rather than the full depth of the model. In our

[1] See Appendix A for the formal definition of MDPs and schedulers.

approach, a multitude of results for short term objectives can be computed in an offline analysis, even for larger system models with restricted analysis depth. The exhaustive analysis of environments, system states, and reconfiguration options results in a PMC-table containing the probabilities and costs for each possible reconfiguration. The PMC-table is stored within a database (implementing the knowledge base). This enables fast and flexible lookups at runtime.

2.2 Runtime Decision Making

At system runtime, it is now possible to dynamically access the precomputed results and search for a reconfiguration action that is most promising for a given temporal objective. We call such a policy that is formulated as query on precomputed PMC-results a *PMC-based strategy*. Technically PMC-based strategies can be implemented as database queries: given a temporal objective Φ, a certain system state, assumptions on the environment, and optionally additional knowledge, context information and environment predictions one can define database queries looking for a reconfiguration policy that tries to optimize either Φ directly (if the results for Φ have been precomputed and stored for a matching situation into the database) or heuristically by combining other precomputed results. Applying such a PMC-based strategy at runtime will cause the resolution of all nondeterministic choices within the MDP (which yields a MC) and can be understood as online construction of an MDP scheduler. As this querying happens at runtime, one can customize and modify PMC-based strategies dynamically according to the current situation, predicted evolution, context information and most importantly the long-term temporal objective. With all the results from the pre-analysis and the ability of dynamically modifying the current reconfiguration according to changes, the hope is to steer the adaptive HW/SW systems very efficiently, outperforming, e.g., related techniques that do not expose such features and in particular simple heuristics such as greedy-based strategies that are currently used due to their simplicity.

3 Robustness of PMC-supported Decision Making

In this section, we investigate the robustness of PMC-supported decision making. In particular, the approach is evaluated to answer the following research questions.

(RQ1) How can the trade-off between conflicting objectives be resolved?
(RQ2) How robust is the decision making in case of inaccurate assumptions?
(RQ3) What impact has the analysis horizon on the quality of adaptations?
(RQ4) What is the influence of different monitoring approaches?
(RQ5) What are general guidelines for configuring a PMC-based control loop?

To address (**RQ1**)–(**RQ5**), we examine an adaptive hardware/software system, whose abstract operational model generalizes to a larger class of producer-consumer systems. The PMC-supported decision making is evaluated using model-based simulation.

Adaptive system example. We have developed an abstract stochastic operational model (an MDP) for an adaptive database system [15] that processes

incoming computational tasks (i.e. queries) over the course of a day. The number of incoming tasks[2] is stochastically distributed and depends on the time of day. Incoming tasks are enqueued into an input buffer for delayed execution. The size of the task buffer is limited and an overflow results in dropped tasks, which in turn incurs an SLA-violation. The platform can be operated in various configurations regarding: the numbers of CPUs used, their frequency levels, and the usage of hyper-threading. Each configuration allows for processing a specific number of tasks per time step while consuming a certain amount of energy. The platform can be reconfigured in reaction to changing workloads or energy constraints. Our parametrized operational model can be instantiated with different resolutions of the time domain and e.g., different distributions for incoming tasks, buffer sizes, configurations and their cost/utility characteristics. Beyond this and due to the selected level of abstraction, the operational model generalizes to a class of producer-consumer systems with alternative operational modes. The specific values used throughout our experiments are based on the ones in [15]. The resolution of the time domain was chosen such that a large number of experiments can still be carried out in a reasonable time.

Long-term objective. Our long-term objective takes reference to the energy/utility tradeoff and are based on the two following measures[3] that refer to the probability of successfully processing tasks and the expected energy consumption.

$$\Pr(\text{Success}) \stackrel{\text{def}}{=} \Pr(\text{“no sla vio” U (“day end” } \wedge \text{ “buffer empty”})) \tag{1}$$

$$\mathbb{E}(\text{Energy}) \stackrel{\text{def}}{=} \mathbb{E}_{\text{Energy}}(\Diamond \text{“day end”}) \tag{2}$$

Here, (1) is the probability of finishing all tasks by the end of the day without SLA-violations, i.e., the *success probability* (which needs to be maximized), and (2) quantifies the expected accumulated energy costs (which should be minimized).

PMC-based strategies. Obviously, the two measures are not independent and even conflicting. Maximizing the probability that all tasks are processed can be trivially achieved by running the system in the configuration with the highest possible throughput, but this also implies maximal energy consumption. Conversely, minimizing the energy consumption means selecting the configuration with the least throughput or even turning off the system, resulting in multiple SLA-violations. To resolve this conflict, we formulate PMC-based strategies, which are gradually applied at runtime (i.e., a PMC-based strategy $\mathfrak{S}$ then induces a Markov chain $\mathcal{M}_{\mathfrak{S}}$). The ability of $\mathfrak{S}$ w.r.t. balancing the energy/utility tradeoff can be judged and quantified and with the help of Pr(Success) and $\mathbb{E}$(Energy) evaluated in $\mathcal{M}_{\mathfrak{S}}$. The strategy $\mathfrak{S}_{\mathbf{Util \geq P}}$ prioritizes the first objective by considering only those configurations where the probability of success[4] (i.e. no SLA-violation occurs) is greater or equal to P. Among the remaining configurations, the one with the lowest energy consumption is selected. $\mathfrak{S}_{\mathbf{Budget:B}}$ tries to maximize

[2] This abstract number and the respective distributions could also be used to characterize the computational weight of larger tasks, e.g., by the number of its subtasks.

[3] Alternative measures for utility could for example address latency.

[4] We simply write $\mathfrak{S}_{\mathbf{Util}}$ when the probability bound P equals 1.

the probability success, but selects only among those configurations that stay on energy budget B. Lastly, strategy $\mathfrak{S}_{\mathbf{Quant:P}}$ relies on a multi-objective property. The idea is to find the minimal energy consumption that keeps the probability of success above threshold P.

Short-term objectives. Since the analysis horizon of the pre-computation is usually limited, the PMC-based strategies are defined in terms of the following *short-term objectives* to approximate the long-term objective.

(SO1) $\mathbb{E}^{\min}_{\text{Energy}}(\Diamond \text{“horizon end”})$

(SO2) $\Pr^{\max}(\text{“no sla vio”} \ \mathrm{U} \ (\text{“horizon end”} \wedge \text{“buffer empty”}))$

(SO3) $\Pr^{\max}(\text{“no sla vio”} \ \mathrm{U}^{\text{Energy}\leq B} \ (\text{“horizon end”} \wedge \text{“buffer empty”}))$

(SO4) $\min e : \Pr^{\max}(\text{“no sla vio”} \ \mathrm{U}^{\text{Energy}\leq e} \ (\text{“horizon end”} \wedge \text{“buffer empty”}))>P$

Here, “horizon end” is an atomic proposition holding in all states where the end of the analysis horizon is reached. Formulas (SO2) and (SO3) are unbounded and bounded versions of PCTL-Until formulas, standing for the maximum probability for reaching the end of the analysis horizon without SLA-violation. The parameter B represents the energy budget for a single day. Formula (SO4) stands for a quantile[2]: given a fixed probability bound $P \in [0, 1]$, the quantile value represents the minimum amount of energy needed such that the energy-bounded version of (S02) holds with at least probability P.

The pre-analysis yields a table containing the PMC results for these objectives, depending on the current system state and the chosen adaptation in that state. At runtime, these results are queried according to the chosen PMC-based strategy. The strategy $\mathfrak{S}_{\mathbf{Util}}$ first maximizes (SO2) and then minimizes (SO1), strategy $\mathfrak{S}_{\mathbf{Budget:B}}$ maximizes (SO3), and strategy $\mathfrak{S}_{\mathbf{Quant:P}}$ minimizes (SO4).

3.1 Experiment Setup

In this section, we give a brief overview of modeling details and the simulation infrastructure that was utilized for the evaluation.

Offline analysis. For the offline analysis step, we make use of a family of MDP models that contains components for the adaptive hardware/software platform itself, the set of stochastic environment assumptions, and additional monitoring components that track runtime information and may trigger adaptations. Technically we rely on the tool ProFeat [7], which supports feature-based modeling of system families and a family-based analysis. For the latter, ProFeat internally relies on Prism, as the input language extends the Prism modelling language and is automatically translated to standard Prism. For the experiments presented in the following, the model consists of 10 abstract time steps, where each time step consists of three phases. The first phase is the *adjust system* phase, where there is a nondeterministic choice among all available reconfiguration actions in the MDP, leading to successor states with respective operation modes. In the succeeding *sample workload* phase, the workload (i.e., 1–6 tasks) is sampled according to a given normal distribution $D(timestep)$ with a given time-dependent mean and a fixed variance. The incoming tasks are stored within an input buffer of size 12. In

the experiments, we used ten synthetic environment assumptions with different shapes and 100 randomly generated environment assumptions. The last phase is the *system operation*, which processes as much tasks as possible within the chosen system configuration[5]. The other components of the family, e.g. environments, can be turned on and off following a set of rules, so we can analyse this family under different environments and configurations, and with different monitors active. The results are written to a database, where the keys are triples of the form $\langle s, m, \alpha \rangle$. Here, s is a state in the MDP that is part of the *adjust system* phase, m stands for monitored state, and α for a reconfiguration action.

Simulation-based evaluation. Rather than executing PMC-based adaptation on the actual system, we follow a simulation-based approach in order to evaluate the PMC-supported decision making. We extended the PRISM simulator to enable the use of PMC-based strategies for resolving all nondeterministic choices at certain points which are determined by an instance-specific monitoring component. The current system state s and monitor states m are then used in a database query to find the "best" adaptation action α w.r.t. the given PMC-strategy. This infrastructure now allows for using the statistical model checking engine of PRISM to compute probabilities and expectations in the Markov chain that results from applying the PMC-strategy to the runtime MDP model. We employ this simulator to gauge the quality of the decision making in various scenarios. For our experiments, we used the confidence interval method of PRISM, with a sample size of 3000 paths and an 0.01 confidence. This means, that in every following experiment, 99% of results were in $\Pr(\text{Success}) \pm 0.02$ and $\mathbb{E}(\text{Energy}) \pm 1.4$ respectively.

3.2 Evaluation of PMC-based Strategies

In the first set of experiments, we compare different PMC-based strategies to illustrate the potential of PMC-supported decision making and we examine their ability to balance long-term objectives (**RQ1**). The strategies are compared under idealized conditions, i.e., the runtime environment behavior agrees with the environment assumptions, the analysis horizon spans the whole day, and adaptations are triggered in every time step. As baseline for the overall adaptation quality, we consider a commonly applied heuristic (which we call $\mathfrak{base}$), which in each time step selects a configuration that uses the least amount of energy but has still high enough throughput to process all incoming tasks immediately. This heuristic has, e.g., been applied in the context of operating systems and the resource management of an adaptive database system [1,15]. We first show how much the PMC-strategies can further reduce the expected energy consumption. This is possible since the pre-analysis provides predictions which allow us to safely delay the processing of tasks without risking immediate SLA-violations. Figure 2 shows the expected costs and the success probability of the different PMC-based strategies compared to $\mathfrak{base}$ when applied to the 100 random and 10 synthetic environments with a variance of 2.0. The strategy $\mathfrak{S}_{\mathbf{Util}}$ (or $\mathfrak{S}_{\mathbf{Util \geq 1.0}}$)

[5] A detailed overview of the PROFEAT model can be found in Appendix B.

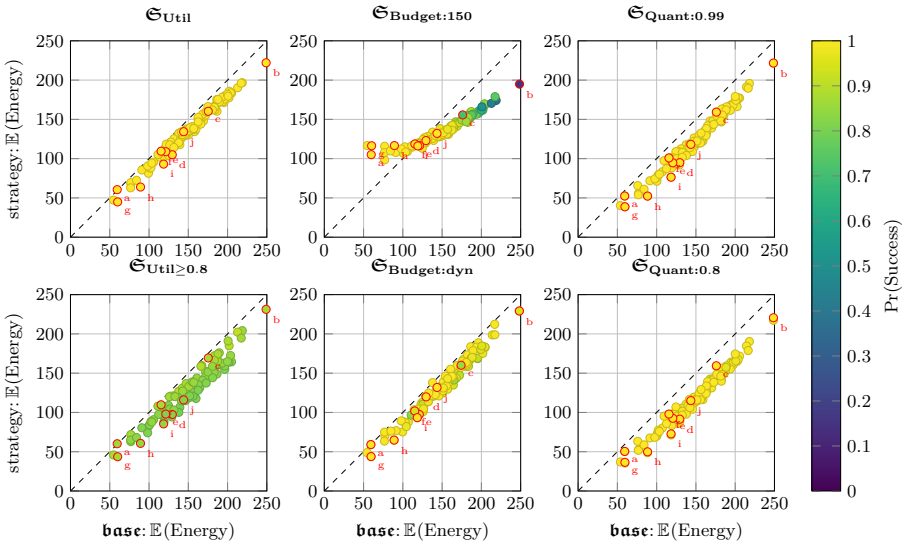

Fig. 2: Comparison: of baseline strategy **base** against PMC-based strategies

guarantees a success probability of 1.0 while reducing the expected costs. For the 100 random environments, the average energy saving was 11.8%. This makes $\mathfrak{S}_{\mathbf{Util}}$ strictly more efficient than the baseline strategy. The expected energy costs can be lowered further by also lowering the bound for the success probability. The application of $\mathfrak{S}_{\mathbf{Util\geq 0.8}}$ gains an average energy saving of 33.1%.

The $\mathfrak{S}_{\mathbf{Budget:B}}$ strategy with a fixed energy budget B completely uses up the given budget even if less energy would have been needed to guarantee a high success probability. Thus, it is important to choose a budget that is close to the overall expected energy consumption for a certain environment. Applying this idea yields the strategy $\mathfrak{S}_{\mathbf{Budget:dyn}}$, where the budget is calculated for each individual environment and set to the expected energy consumption under the **base** heuristic. This leads to an average cost improvement of 13.4% compared to **base**, while still providing a high success probability. The strategy $\mathfrak{S}_{\mathbf{Quant:P_{0.99}}}$ is superior to both $\mathfrak{S}_{\mathbf{Util}}$ and $\mathfrak{S}_{\mathbf{Budget:dyn}}$, as it reduces the energy consumption by 18.5% on average while still providing a success probability of almost 1. Lowering the success probability bound to 0.8 leads to a further reduction of 21.88% compared to **base**. However, unlike the $\mathfrak{S}_{\mathbf{Util\geq P}}$ strategy, lowering the probability bound P of strategy $\mathfrak{S}_{\mathbf{Quant:P}}$ does not affect the success probability as much. In the setting with idealized assumptions, one can conclude that the considered PMC-based strategies can reliably outperform the baseline strategy **base**. In the following, the idealized assumptions are stepwise relaxed. Since the strategies $\mathfrak{S}_{\mathbf{Util}}$ and $\mathfrak{S}_{\mathbf{Quant:0.99}}$ are the most promising of the compared strategies, the following experiments will focus on these two strategies.

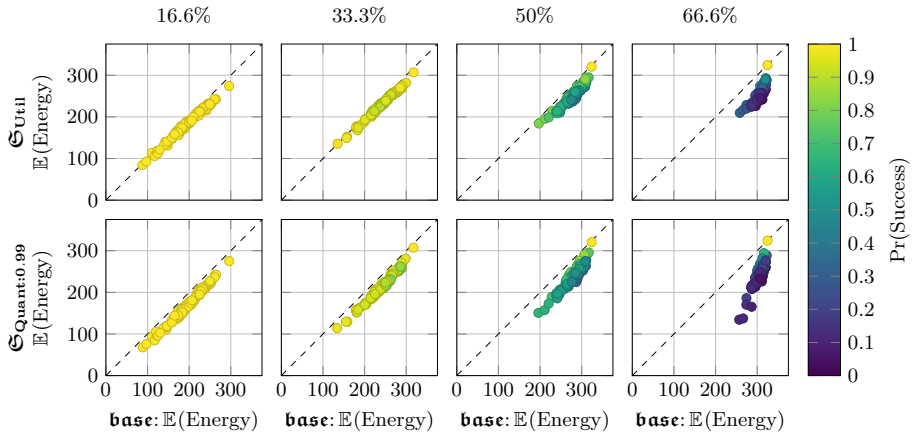

Fig. 3: Robustness of $\mathfrak{S}_{\mathbf{Util}}$ and $\mathfrak{S}_{\mathbf{Quant:0.99}}$ under noise (increased means)

3.3 Impact of Unexpected Workloads

The focus will now be on the robustness of the PMC-based decision making when the actual load put onto the system is higher than assumed in the preceding offline analysis (**RQ2**). For each of the 110 environments, we gradually increase for each time step the means of the distributions $D(t)$ within the runtime environment by 16.6%, 33.3%, 50%, and up to 66.6% of the maximal workload processable in one time step. Figure 3 shows the effect on Pr(Success) and $\mathbb{E}$(Energy) for the PMC-based strategies $\mathfrak{S}_{\mathbf{Util}}$ and $\mathfrak{S}_{\mathbf{Quant:0.99}}$, again with baseline $\mathfrak{base}$.

The average energy consumption increases steadily, which is expected since increasing the overall number of tasks requires a higher throughput. The success probability decreases with an increasing unexpected load on the system. $\mathfrak{S}_{\mathbf{Util}}$ is able to properly deal with an increase of up to 33.3%, without incurring higher costs than baseline ($\mathbb{E}$(Energy) is still 4.9% better compared to the baseline) and without significantly lower success probability (the average Pr(Success) is down to 0.94). From level 50% onward, $\mathfrak{S}_{\mathbf{Util}}$ is no longer able to properly guide the system, since the difference between the expected number of arriving tasks and actually arriving tasks is simply too large. On the other hand, $\mathfrak{S}_{\mathbf{Quant:0.99}}$ reacts slightly different to an unexpected load. Here, we also see a reasonable performance up to 33.3%, but with higher cost improvements of 11.3% at load level 33.3%. The average success probability drops down to 0.92, while the worst probability value is 0.79. For $\mathfrak{S}_{\mathbf{Quant:0.99}}$ with load levels above 50%, costs drop drastically whenever the success probability is getting close to zero. This is due to the fact that $\mathfrak{S}_{\mathbf{Quant:0.99}}$ optimizes for energy consumption only in case SLA-violations can be no longer avoided. From the above results we conclude that in the considered setting, both PMC-based strategies can reasonably tolerate noise levels of 30–40% of the maximum load processable in one time step.

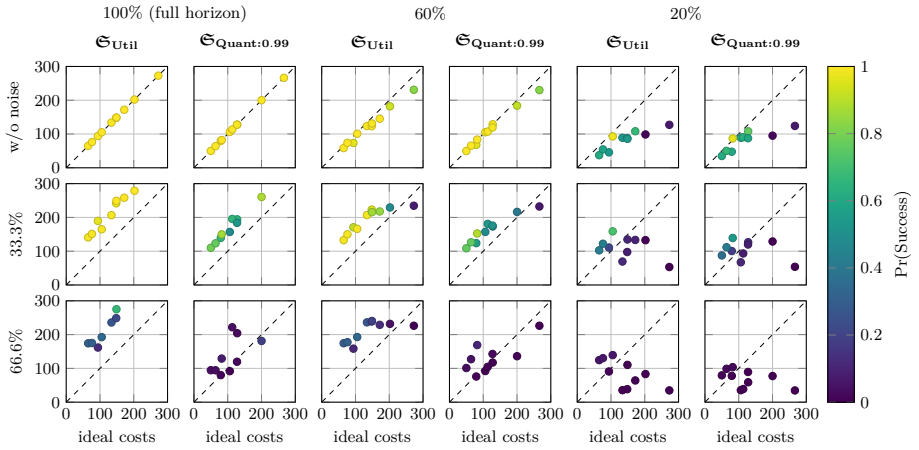

Fig. 4: Expected energy consumption and probability of success under increasing noise and decreasing analysis intervals compared to idealized conditions.

3.4 Controllable Factors of the PMC-supported Decision Making

Our remaining research questions concerned the configuration of the PMC-based control loop ((**RQ3**) and (**RQ4**)). While results under ideal conditions and under reasonable noise looked promising, the goal is to explore the approach under more realistic conditions. For this we carried out experiments, where (1) the analysis horizon does not cover the full day and (2) system reconfiguration is not happening in each timestep, but triggered by different monitors that observe the system and the environment. Results for (1) imply that analysis horizon should be above 40% and that (2) the monitor *env. change* serves best to minimize the number of reconfiguration steps (every second timestep) and maintaining the energy consumption and performance at a reasonable level. For the latter, the monitor *env. change* triggers adaption whenever the number of incoming tasks per timestep has changed by at least 2 since the last decision was made. [6]
Insights from the individual experiments for (1) and (2) went into a combined experimental study (cf. Figure 4) in which we investigated the robustness of the decision making under noise with limited analysis-horizon and using dynamic system reconfiguration (here using the monitor *env. change*).
Here, we see that both PMC-Strategies are still able to produce results of reasonable quality with analysis horizon higher than 40%, and a noise up to 33%. But it is also visible that the quality impacts add up, worsening the adaptation and creating some outliers, where short term goals can no longer be kept.
Finally, we want to answer (**RQ5**), and find some guidelines for choosing analysis horizon and monitoring. Generally, triggering adaptations in every time step yielded always the best results. However, should the cost of adaptation be

[6] Detailed results for (1) and (2) in isolation can be found in Appendix C.

not negligible, an environment-dependent monitor provides a suitable trade-off between the number of decisions and the adaptation quality. The choice of the PMC-based strategy depends on the overall goal, i.e., the long-term objective. As shown in Figure 4, the $\mathfrak{S}_{\mathbf{Util}}$ prioritizes the prevention of SLA violations at the cost of a higher energy consumption, while the $\mathfrak{S}_{\mathbf{Quant:0.99}}$ allows for a greater reduction of energy costs at the risk of additional SLA violations. Finally, the analysis interval length should be chosen as short as possible to decrease the overhead of the pre-computation, but long enough to allow for the prediction of high-load situations. In our experiments, an analysis horizon higher then 40% was sufficient, independent from the noise level.

3.5 Performance and scalability of PMC-based decision making

All experiments have been carried out on a server with Intel(R) Xeon(R) E5-2680 CPUs with in total 16 physical cores and 377GB of DDR3 RAM. For the quantitative analysis as well as for the decision making at runtime, we relied on a modified version of Prism 4.5. For all experiments 8GB of RAM were sufficient. For the database to store and look up PMC-results we relied on SQLite 3.30.1.

Offline Analysis. The base model used consists of 672 085 reachable states, the construction of which required less than one second using the Sparse engine of Prism. The analysis times to compute the results for all 110 environments per property (SO1–SO4) depends on the considered horizon. Considering the maximal horizon, the total computation time (i.e., over all environments) for (S01) was still less than 1s, for (S02) it was less than 2s, and for (S04) not more than 10s. The budget considered in Formula (SO3) demands for an additional counter variable added to the model. This leads to an increased state space size of up to 67 618 570 states, when considering a budget of 125. Due to the increased state space, the total computation time for (S03) was longer, but was still less than 80s. In most cases, the database consists of 3696 rows (i.e., possible configurations) and consumes about 1.3MB of physical storage. Including (S03) and a monitor, this grew up to 6 054 048 rows (1.2GB) of storage.

Runtime Evaluation. The evaluation at runtime relied on statistical model checking with 3000 simulation runs per experiment. Each run involves up to ten decision points where database lookups happen. Each lookup to the indexed database took about 1.1ms on average when using (SO1, SO2, SO4) and about 15ms for (SO3). In the largest setting the average lookup time grew up to 895ms.

Threats to Validity. The proposed method suits well to adaptive producer-consumer systems whenever postponing tasks allows for operating the system in more energy efficient computational modes. Regarding scalability of PMC-supported decision making, it will be crucial that the database querying is fast enough. Reducing query-time was out of scope for this paper. On one hand the databases will easily grow and query time will increase when considering larger system models and/or PMC-based strategies involving complex system or environment monitors. On the other hand there is large room for improvements not yet considered for this article. By reducing the tables to only necessary information and using advanced database techniques, we are confident that even for larger system models, query times can be kept within reasonable bounds.

4 Conclusion

A goal in this paper was to show the general potential of PMC-supported decision making at runtime and in particular to study the robustness of the PMC-based strategies in practice. Our experiments have been carried out for an abstract version of an adaptive database server. As a first result, we were able to keep the time for database lookups and hence the overall delay for decision making reasonably low. The experiments demonstrate, that there exist robust PMC-based strategies, which operate with limited analysis horizon, under noise, and do at the same time, keep the number of reconfiguration steps (and hence cost for reconfiguration) within reasonable bounds. Given these promising results, a natural next step would be to apply the PMC-decision making to the actual adaptive system (rather than the simulation of the system). In this setting additional challenges are to be expected, and in particular with respect to delays and other concrete timing issues, which are due to the selected level of abstraction (regarding time) in the abstract system model as used in the offline-analysis.

References

1. F. M. Arega, M. Hähnel, and W. Dargie. Dynamic power management in a heterogeneous processor architecture. In *Architecture of Computing Systems - ARCS 2017 - 30th International Conference, Vienna, Austria, April 3-6, 2017, Proceedings*, volume 10172 of *Lecture Notes in Computer Science*, pages 248–260. Springer, 2017.
2. C. Baier, M. Daum, C. Dubslaff, J. Klein, and S. Klüppelholz. Energy-utility quantiles. In *NASA Formal Methods - 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 - May 1, 2014. Proceedings*, volume 8430 of *Lecture Notes in Computer Science*, pages 285–299. Springer, 2014.
3. C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
4. R. Calinescu, C. Ghezzi, M. Z. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 55(9):69–77, 2012.
5. R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS management and optimization in service-based systems. *IEEE Transactions on Software Engineering*, 37(3):387–409, 2011.
6. J. Cámara and R. de Lemos. Evaluation of resilience in self-adaptive systems using probabilistic model-checking. In *7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2012, Zurich, Switzerland, June 4-5, 2012*, pages 53–62. IEEE Computer Society, 2012.
7. P. Chrszon, C. Dubslaff, S. Klüppelholz, and C. Baier. Profeat: feature-oriented engineering for family-based probabilistic model checking. *Formal Aspects of Computing*, 30(1):45–75, 2018.
8. A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In R. N. Taylor, H. C. Gall, and N. Medvidovic, editors, *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, pages 341–350. ACM, 2011.
9. A. Filieri, G. Tamburrelli, and C. Ghezzi. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. *IEEE Transactions on Software Engineering*, 42(1):75–99, 2016.

10. V. Forejt, M. Z. Kwiatkowska, D. Parker, H. Qu, and M. Ujma. Incremental runtime verification of probabilistic systems. In *Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers*, volume 7687 of *Lecture Notes in Computer Science*, pages 314–319. Springer, 2012.
11. C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli. Managing non-functional uncertainty via model-driven adaptivity. In D. Notkin, B. H. C. Cheng, and K. Pohl, editors, *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 33–42. IEEE Computer Society, 2013.
12. M. Güdemann, F. Ortmeier, and W. Reif. Safety and Dependability Analysis of Self-Adaptive Systems. In *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (isola 2006)*, pages 177–184, Nov. 2006.
13. M. Hachicha, R. B. Halima, and A. H. Kacem. Formal verification approaches of self-adaptive systems: A survey. In *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES-2019, Budapest, Hungary, 4-6 September 2019*, volume 159 of *Procedia Computer Science*, pages 1853–1862. Elsevier, 2019.
14. K. Johnson, R. Calinescu, and S. Kikuchi. An incremental verification framework for component-based software systems. In *CBSE'13, Proceedings of the 16th ACM SIGSOFT Symposium on Component Based Software Engineering, part of Comparch '13, Vancouver, BC, Canada, June 17-21, 2013*, pages 33–42. ACM, 2013.
15. T. Kissinger, D. Habich, and W. Lehner. Adaptive energy-control for in-memory database systems. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 351–364. ACM, 2018.
16. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.
17. M. Leucker and C. Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, May 2009.
18. M. Lochau, S. Mennicke, H. Baller, and L. Ribbeck. Incremental model checking of delta-oriented software product lines. *Journal of Logical and Algebraic Methods in Programming*, 85(1):245–267, 2016.
19. M. A. Nia, M. Kargahi, and F. Faghih. Probabilistic approximation of runtime quantitative verification in self-adaptive systems. *Microprocessors and Microsystems*, 72, 2020.
20. J. Rinast. *An online model-checking framework for timed automata.* PhD thesis, Hamburg University of Technology, 2015.
21. D. Weyns, M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad. A survey of formal methods in self-adaptive systems. In *Fifth International C* Conference on Computer Science & Software Engineering, C3S2E '12, Montreal, QC, Canada, June 27-29, 2012*, pages 67–79. ACM, 2012.
22. Y. Zhao. *Online Model Checking Mechanisms and Its Applications.* PhD thesis, Universität Paderborn, 2016.
23. Y. Zhao and F. Rammig. Online Model Checking for Dependable Real-Time Systems. In *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 154–161, Apr. 2012. ISSN: 2375-5261.

A Markov Decision Processes (MDPs)

These definitions are based on [3].

Definition 1 (Markov Decision Process). *A* Markov Decision Process *(MDP) is a tuple* $M = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ *where*

- S *is a countable set of states,*
- Act *is a set of actions,*
- $\mathbf{P} : S \times Act \times S \to [0,1]$ *is the transition probability function such that for all states* $s \in S$ *and actions* $\alpha \in Act$*:*

$$\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{0, 1\}$$

- $\iota_{init} : S \to [0,1]$ *is the initial distribution such that* $\sum_{s \in S} \iota_{init}(s) = 1$*, and*
- AP *is a set of atomic propositions and* $L : S \to 2^{AP}$ *a labelling function mapping a combination of atomic propositions to each state.*

An action α *is enabled in state* s *if and only if* $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1$ *. Let* $Act(s)$ *denote the set of enabled actions in state* s*. For any state* $s \in S$*, it is required that* $Act(s) \neq \emptyset$*. Each state* $s' \in S$ *for which* $\mathbf{P}(s, \alpha, s') > 0$ *is called an* α*-successor of* s*, and* $\alpha(s) = \{s' \in S \mid \mathbf{P}(s, \alpha, s') > 0\}$ *the set of all* α*-successor of* s*.*

Given a set $S' \subseteq S$*, we note* $\mathbf{P}(s, \alpha, S') = \sum_{s' \in S'} \mathbf{P}(s, \alpha, s')$*.*

In this paper, we utilize MDPs to represent the abstract system model, where nondeterministic choices stand for the possible reconfiguration actions and probability distributions encode the stochastic environment assumptions.

Definition 2 (Scheduler for MDP and Induced Markov Chain). *Let* $M = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ *be an MDP. A scheduler for* M *is a function* $\mathfrak{S} : S^+ \to Act$ *such that* $\mathfrak{S}(s_0 s_1 \dots s_n) \in Act(s_n)$*.*

The induced Markov chain $M^{\mathfrak{S}}$ *is a tuple* $(S^+, \mathbf{P}^{\mathfrak{S}}, \iota_{init}, AP, L')$ *where*

$$\mathbf{P}^{\mathfrak{S}}(s_0 s_1 \dots s_n, s_0 s_1 \dots s_n s_{n+1}) = \mathbf{P}(s_n, \alpha, s_{n+1})$$

with $\alpha = \mathfrak{S}(s_0 s_1 \dots s_n)$*, and* $L'(s_0 s_1 \dots s_n) = L(s_n)$*.*

Since in our approach nondeterministic choices correspond to reconfiguration actions for adapting the system, a scheduler corresponds to an adaptation strategy. A *PMC-based strategy* is then defined in terms of the results for the short-term objectives which were gathered during the offline analysis of the system model. In the analysis of the runtime model, all nondeterministic choices are resolved by a database lookup according to the PMC-based strategy. Thus, the PMC-based strategy acts as the scheduler in the runtime model. Using the resulting induced Markov chain, we can compute the relevant measures of the global objectives.

B Details on the Modeling and Analysis

In this appendix we provide a detailed description of the family-based modeling and the analysis for the PMC-based decision making. Starting point is the general structure of the family model that serves as a template to create models needed for the offline analysis as well as the runtime evaluation.

B.1 Feature Model for the Offline Analysis and Runtime Evaluation

The template model from which all relevant model instances are generated consists of a common part and additional components that can either be included into a particular instance or not. We rely on a feature-based modeling approach to describe the entire system family. Figure 5 shows the feature diagram which provides an overview on which features exist and the (parts of the) underlying logic that determines the instances that can be composed. Each instance of our model family consists of

- the MAPE-K phase model,
- the adaptive DBMS system itself,
- the current configuration of the system (optionally),
- one selected environment, and
- one or more monitors collecting additional information.

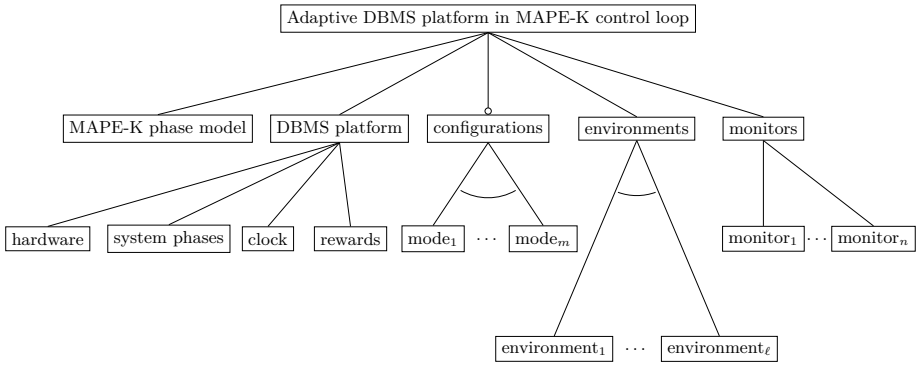

Fig. 5: Feature Diagram for Adaptive DBMS Platform in MAPE-K Control Loop

From this, we can create all the instances used for the offline analysis by fixing one particular environment and one persistent operational mode in each respective instance. The resulting models are Markov chains (bound to the analysis horizon) that are analysed w.r.t. the provided short-term objectives. The PMC-results for each short-term objective, environment and operational mode are stored into our database. In case that no operational mode is fixed, an MDP is created which preserves nondeterministic choices and allows for computing maximizing

or minimizing schedulers, i.e., optimal resolution of these choices w.r.t. a given objective. Similarly, one can derive the model(s) for the runtime evaluation. Here, the operational mode is initially disabled. The choice of monitors to be included depends on the information required by the respective PMC-strategy. With this, the resulting MDP is ready to be used in the statistical model checking. Our plugin then loads the PMC-strategy along with the relevant data (i.e. PMC-result tables) and ensures the resolution of nondeterministic choices, by activating the best operational mode accordingly.
For the concrete modeling we rely on ProFeat [7], which supports feature-based modeling for families of MDPs. The input language of ProFeat is an extension of the Prism modeling language which ProFeat then internally translates into a family model (or alternatively each instance separately) in the standard Prism modeling language (without features, arrays, etc.). The semantics of this Prism model is an MDP that yields the basis for a (family-based) analysis and the evaluation via statistical model checking. Figure 6 shows the realization of the above feature diagram within the ProFeat language.

```
root feature
  all of phases, system, configurations, environments, monitors;
endfeature
```

Fig. 6: ProFeat Realization of the Family Model

B.2 ProFeat Model

In the following section we will briefly explain all relevant components of the model (i.e. implementations of the leaf nodes in the feature diagram) by providing (fragments of) the ProFeat model code. The full model is part of the artifact available under `https://tud.link/ovau`.

MAPE-K Phase Model. This part of the model can be seen as a global clock or an orchestrating component that defines phases in which the envolved MAPE-K components can take their next step. The the three phases (analyze, plan, execute) alternate. The next phase is entered by an entry action (e.g. `plan`) and left with an exit action (e.g. `plan_end`). These actions synchronize with the components relevant in the respective step. The first phase (analyze) allows for dynamically changing the environment assumptions used in the PMC-based analysis and/or the concrete environment used in evaluation with statistical model checking methods. This dynamics has not been used in the experiments presented in this paper, and therefore has no functionality. The second phase (plan) enables our Prism plugin to interface with the model, adapting the configurations depending on our PMC-Strategy. The last phase (execute) then enables all actions in the DBMS System, until a new decision is triggered. Figure 7 shows the ProFeat

code for the MAPE-K phases. `f_signal` stands for a formula which evaluates to true if a corresponding monitor is in a specific state, signaling the transition to the next phase.

```
feature phases
  modules phases_impl;
endfeature

module phases_impl
  op_phase: [0..2] init 0;
  op_step:  [0..1] init 0;

  [analyze_env]   (op_phase=0) & (op_step=0) -> (op_step'=1);
  [analyze_end]   (op_phase=0) & (op_step=1) -> (op_step'=0) & (op_phase'=1);

  [plan]          (op_phase=1) & (op_step=0) -> (op_step'=1);
  [plan_end]      (op_phase=1) & (op_step=1) -> (op_step'=0) & (op_phase'=2);

  [execution]     (op_phase=2) & (op_step=0) -> (op_step'=1);
  [execution_end] (op_phase=2) & (op_step=1)
                   & (f_signal)              -> (op_step'=0) & (op_phase'=0);
endmodule
```

Fig. 7: ProFeat Code for MAKE-K phases

Adaptive DBMS System. The adaptive system itself (cf. Figure 8) consists of four mandatory subfeatures.
The first subfeature `hardware` describes the configurations of the hardware. It contains variables for the currently enabled number of cores and the frequency level. The second subfeature is `system_phases`. The adaptive system itself takes five microsteps: it starts the hardware reconfiguration (action `adjust`), sets the number of cores (action `cores_on[j]`) and the frequency level (action `setfreq[i]`) before moving to the working step (action `working`) that decreases the workload according to the available computational power. The action `sampling` synchronizes with the environment to increase the load to be processed in the next timestep(s) according to what the environment adds to the current load. The current load variable along with the increase and decrease operations is captured in a subfeature of the environment. Details will be provided in the paragraph "Environments". The `in_system` guard synchronises the system with the phases feature, and assures that there is only activity in the execution phase. The third subfeature is the system clock (cf. Figure 9) which synchronizes in each `working` step and increases the variable `timestep` by one.
The last subfeature contains the energy/utility rewards (cf. Figure 10). It contains the definition of transition rewards that synchronize with the `working` step. Figure 10 shows for example the reward structure for the total energy consumption.

```
feature system
  all of hardware, system_phases, system_clock, energy_utility_rewards;
endfeature

feature hardware
  cores:     [1..c_number_cores] init 1;
  freq_step: [0..c_socket_number_frequencies-1] init 0;
endfeature

feature system_phases
  modules system_phase_impl;
endfeature

module system_phase_impl
  phase_steps: [0..4] init 2;

  [sampling]  (phase_steps=0) & (in_system) -> (phase_steps'=1);
  [adjust]    (phase_steps=1) & (in_system) -> (phase_steps'=2);

  // set Number of cores
  for j in [1..c_number_cores]
    [cores_on[j]] (phase_steps=2) & (in_system) -> (phase_steps'=3);
  endfor

  // set frequency
  for i in [0..c_socket_number_frequencies-1]
    [setfreq[i]] (phase_steps=3) & (in_system) -> (phase_steps'=4);
  endfor

  [working] (phase_steps=4) & (in_system) -> (phase_steps'=0);
endmodule
```

Fig. 8: ProFeat Code for the Adaptive DBMS System

```
feature system_clock
  modules system_clock_impl;
endfeature

module system_clock_impl
  timestep: [0..number_of_timesteps];

  [working] (timestep<number_of_timesteps) -> (timestep'=timestep+1);
  [working] (timestep=number_of_timesteps) -> (timestep'=0);
endmodule
```

Fig. 9: ProFeat Code for System clock

The concrete values for energy consumption (and maximum gained utility) for each configuration are stored in large tables (ProFeat arrays) and accessible via functions. E.g., for the energy, we rely on the function `f_energy_from_hardware`.

```
feature energy_utility_rewards
  rewards "energy"
    [working] true : f_energy_from_hardware(hardware.cores,
                                            hardware.freq_step);
  endrewards

  rewards "sla_violation"
    [working] true : max(0, f_load - max_load*2);
  endrewards
endfeature
```

Fig. 10: ProFeat Code Energy/Utility Rewards

Configurations. The configuration feature will automatically be generated by our tooling. It contains variables that store the current operational mode. The rules in the transition definitions describe which of the nondeterminism in the DBMS model are available(and which are blocked). Figure 11 is not exactly the automatically generated ProFeat code, but optimized here for the sake of readability. The feature provides the interface to the external PMC-decider. The

```
module configurations_impl
   operational_mode_1: [1..c_number_cores];
   operational_mode_2: [0..socket_number_frequencies-1];

   for j in [1..c_number_cores]
     [cores_on[j]] (operational_mode_1 = j) -> true;
   endfor

   for i in [0..c_socket_number_frequencies-1]
     [setfreq[i]] (operational_mode_2 = i)  -> true;
   endfor
endmodule
```

Fig. 11: ProFeat Code for Configurations

latter externally communicates its adaptation decision by externally manipulating the mode variable values of `operational_mode_1` and `operational_mode_2` in the plan phase. The transition definitions ensure, that the only possible

reconfiguration actions will be the ones that lead to the operational mode selected by the decider. The other actions will be blocked.

Environments. An environment feature (cf. Figure 12) consists of one module which contains the variable for the current load and synchronous actions `sample` and `working` that increase and decrease the load to be processed. The formula `next_load` here generates a new load value out of a previously defined normal distribution, with mean dependent on the current timestep. This mean over time function can either be previously defined for every timestep (like with the 10 synthetic environments in Figure 13), or randomly generated using the python random number generator. For our evaluation experiments with noise we manipulate the means of these distributions in the runtime model. The full details of sampling incoming workload and introducing noise are part of the artifact available under `https://tud.link/ovau`.

```
feature environment;
  modules environment_impl;
endfeature

module environment_load_impl
  load: [0..max_load*2+1];

  [sampling] true -> (load'= max(min(load + next_load, max_load*2+1), 0));
  [working]  true -> (load '= min(f_remaining_load, max_load*2));
endmodule
```

Fig. 12: ProFeat Code for Environments

Monitors. Lastly, the family model consist of an arbitrary number of monitor features, which are are used for purely technical purposes (e.g., counting the number of steps between decision points tracking the remaining energy budget) or to store and deliver additional information that will be made available to PMC-based strategies (e.g., counting the number of dropped tasks or the number of SLA-violations). Monitors can trigger the end of the execution phase of the MAPE-K loop by "sending signals", e.g., to switch into the next operational phase. Figure 14 shows such an example of such a monitor component with signal `f_decision_reach_3_signal`. The formula `f_signal` is a disjunction of the signal functions belonging to all existing monitor features and triggers the next phase once one of them activates (which would be every 3 timesteps with `f_decision_reach_3_signal`).
The hidden details of the model, such as additional functions, energy/utility characteristics, the model instances considered in the experiments along with the tooling are part of the artifact available under `https://tud.link/ovau`.

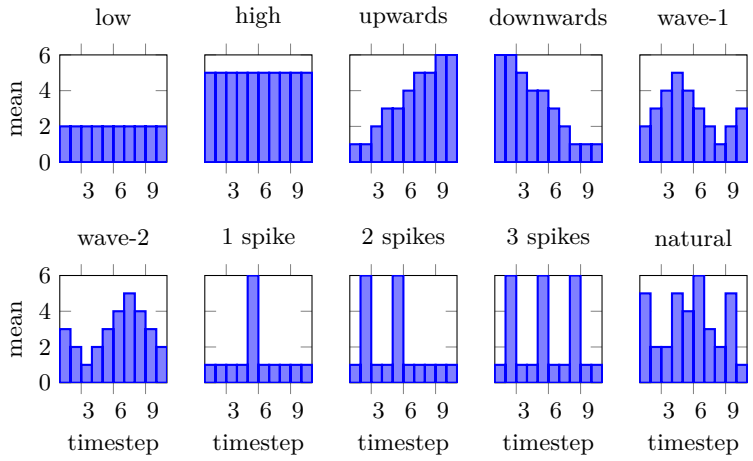

Fig. 13: Synthetic Load Distributions

```
module decision_reach_3_impl
  count: [0..3] init 0;

  [working] (count < 3) -> (count'=count+1);
  [working] (count = 3) -> true;

  [execution] true -> (count'=0);

endmodule

formula f_decision_reach_3_signal = (decision_reach_3.count = 3);

formula f_signal = ...| f_decision_reach_3_signal | ...;
```

Fig. 14: ProFeat Code for an Example Monitor

C Further Experiments

C.1 Influence of the Analysis Horizon

(**RQ3**) concerns itself with the impact, limited knowledge of the future has on the quality of our decision making. To research this, we consider an arbitrary but fixed *analysis horizon*, i.e., an upper bound of the number of time steps that are analyzed in the pre-computation, and evaluate its influence on the quality of the adaptations.

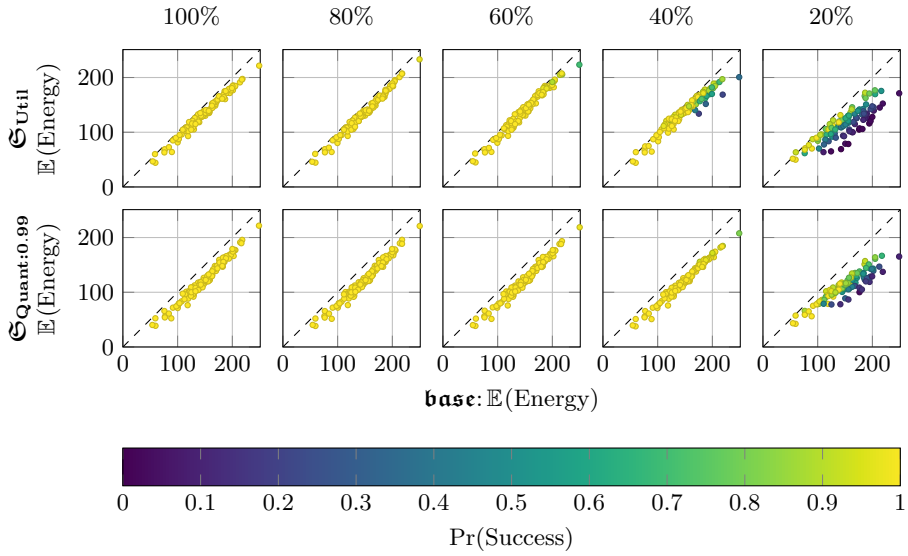

Fig. 15: Impact of the analysis horizon on expected costs and the probability of SLA violations under $\mathfrak{S}_{\mathbf{Util}}$ and $\mathfrak{S}_{\mathbf{Quant:0.99}}$ compared to the baseline strategy

For the strategies $\mathfrak{S}_{\mathbf{Util}}$ and $\mathfrak{S}_{\mathbf{Quant:0.99}}$, we computed the expected energy consumption and the probability for SLA violations for decreasing analysis interval lengths. The results are shown in Figure 15. For an analysis horizon below 40%, both strategies can no longer avoid SLA violations since they cannot anticipate peak loads occurring at the end of the day. If we assume a continuous operation of the system and further allow shifting tasks beyond the end of the day, even the 20% analysis interval is viable. In conclusion, for PMC-based strategies that are able to meet the long-term objective by using localized, mostly independent decisions, a close analysis horizon is sufficient.

C.2 Comparison of Monitoring Approaches

In previous experiments, we assumed that adaptations are triggered in each time step. However, in a MAPE-K loop adaptations are usually initiated by the monitoring component in case there have been significant changes in the environment or the system itself. In this section, we evaluate the quality of PMC-based decision making for dynamically triggered adaptations (**RQ4**). We tested a number of different monitoring approaches:

- trigger adaptations on every third time step (denoted by *3-step*)
- trigger if there are more than 5 tasks pending (*threshhold*)
- trigger if the number of incoming tasks per time step has changed by at least 2 since the last decision (*env. change*)
- trigger if the number of pending tasks has changed by at least 2 since the last decision (*load change*)

The monitor components are not only used in the runtime model, but are already present within the model used for the pre-computation. Thus, the computed predictions take the behavior of the monitoring into account.

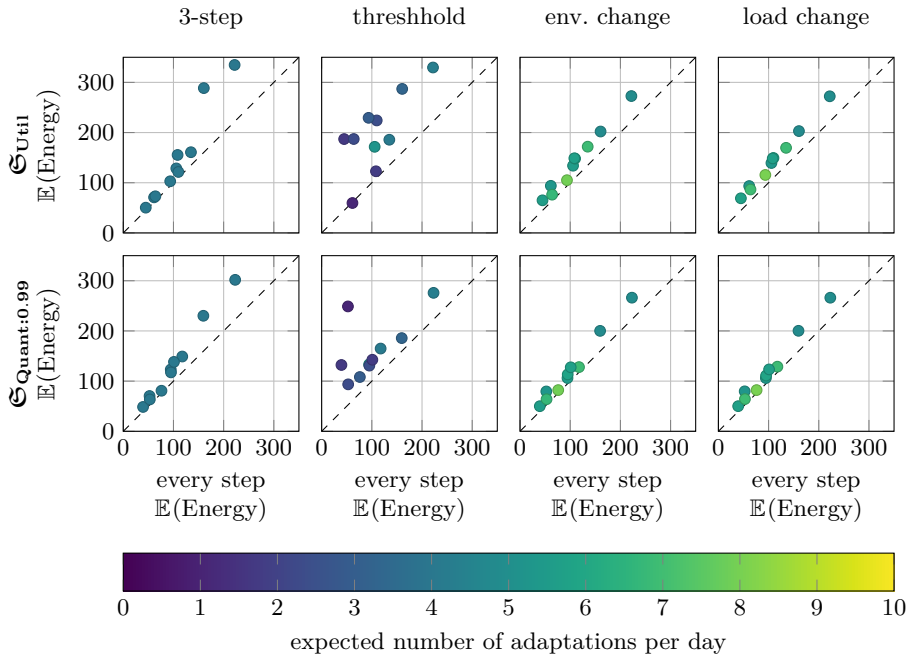

Fig. 16: Expected energy consumption under different monitors compared to adaptations triggered on every step.

Figure 16 shows the expected energy consumption under the described monitors compared to a static triggering of adaptations on every time step. The shading of the data points indicates the expected number of adaptations that are triggered by the respective monitor in a single day. Both PMC-based strategies are able to prevent additional SLA violations (compared to the every step monitor) by overcommitting resources which increases the overall energy consumption. As expected, triggering adaptations depending on changes in the environment (*env. change*) is the most successful approach, as it is able to cut the number of adaptations in half while still maintaining a reasonable energy consumption.