Instance-based Learning for Hybrid Planning

Ashutosh Pandey, Bradley Schmerl, and David Garlan
School of Computer Science
Carnegie Mellon University, Pittsburgh, PA
ashutosp@cs.cmu.edu · schmerl@cs.cmu.edu · garlan@cs.cmu.edu

***Abstract*—Due to the fundamental trade-off between quality and timeliness of planning, designers of self-adaptive systems often have to compromise between an approach that is quick to find an adaptation plan and an approach that is slow but finds a quality adaptation plan. To deal with this trade-off, in our previous work, we proposed a hybrid planning approach that combines a deliberative and a reactive planning approach to find a balance between quality and timeliness of planning. However, when reactive and deliberative planning is combined to instantiate a hybrid planner, the key challenge is to decide which approach(es) should be invoked to solve a planning problem. To this end, this paper proposes to use a data-driven instance-based learning to find an appropriate combination of the two planning approaches when solving a planning problem. As an initial proof of concept, the paper presents results of a small experiment that indicate the potential of the proposed approach to identify a combination of the two planning approaches to solve a planning problem.**

I. Introduction

Self-adaptive software systems make decisions at run time that seek to change their behavior in response to faults, changing environments, and attacks. Having an appropriate planning approach to find adaptation plans is critical to successful self-adaptation. To determine adaptation plans, researchers in the self-adaptive community have suggested various approaches, such as model-checking [9], fuzzy-logic [18], reinforcement learning [19], stochastic search [15], automated planning [27] and case-based reasoning [28].

For any planning approach, there is a fundamental trade-off between quality and timeliness of planning: quality of planning refers to the likelihood of a (determined) plan meeting adaptation goals and timeliness of planning indicates the ability to find a plan in a timely fashion. Planning could be considered as a search/optimization process to determine a plan in a state space; a more complete search provides better quality guarantees about a plan, but requires more time to complete. In fact, the $NoFreeLunchTheorem$ formally proves that for any search/optimization algorithm, any elevated performance over one class of problems is paid for in performance over another class [5].

However, modern systems such as Amazon Web Services [1] and Netflix [2], not only need to find the adaptation plans quickly but also require the plans to be optimal. Ideally, for such systems, a planning approach is needed that can find optimal adaptation plans in a timely manner.

To find a balance between quality and timeliness, in our previous work, we proposed the idea of hybrid planning that combines a reactive and a deliberative approach [16]. In the hybrid planning approach, a self-adaptive system uses reactive planning to provide a quick (but potentially sub-optimal) response to a planning problem, but simultaneously uses deliberative planning to find a higher quality plan. Once the deliberative plan is ready, it takes over execution from the reactive plan.

Specifically, in our previous work, we combine a reactive approach with a deliberative planning approach. Reactive planning is invoked only in case of an emergency situation i.e., a constraint violation. For a constraint violation, reactive planning is able to provide a quick response since planning time is reduced by ignoring parts of the operating domain state space (e.g., states that arise as a result of uncertainty); however, this reduced state space is likely to result in low quality plans. Our approach simultaneously invoked deliberative planning to consider more of the state space while beginning the execution of the reactive plan. Once a deliberative plan is ready, it takes over from the reactive plan. Our experiments in the context of a cloud-based application demonstrated the effectiveness of this hybrid planning approach [16].

Invoking reactive planning only on constraint violations, as in our original work, suffers from two limitations. First, it limits the applicability of hybrid planning to cases involving constraint violations: so, the approach is effective for self-healing, but it is not clear how it would apply to other self-* properties that are not triggered conditionally. For example, self-optimization attempts to improve utility continuously [21]. Second, for complex systems, it can be difficult to determine a fixed set of predefined conditions at design time that capture all possible constraint violations.

In this paper, we propose to address these limitations by applying hybrid planning to non-conditional self-* properties using a data-driven instance-based learning (IBL) approach that, broadly speaking, helps to solve a new problem based on similar problems seen in the past [29]. It is called instance-based since it maps a problem space to the solution space using training instances; this mapping is eventually used to solve a new problem instance. The application of IBL is based on the assumption that, for a self-adaptive system the decision to invoke reactive planning for two similar planning problems would also be similar; in other words, an effective combination of reactive and deliberative planning for one problem should also work for another "closely related" problem.

IBL has benefits over condition-based invocation of reactive planning. First, using IBL a system could apply reactive planning to a broader range of situations compared to specific (predefined) conditions. Second, system designers do not need to (explicitly) identify conditions that trigger reactive planning.

As a preliminary validation of IBL, we performed experiments for a simulated cloud-based application. Specifically, we manually identified pairs of similar planning problems and investigated if the decision to invoke reactive planning for one problem in a pair also applies to the other problem in the pair. The experiments indicated the potential of IBL to decide

whether reactive planning should be invoked (or not) for a planning problem.

The idea of applying instance-based learning (in the context of hybrid planning) is inspired by the work done in the field of hyper-heuristics. A hyper-heuristic can be considered as a high-level heuristic that, given a particular search problem instance and a number of low-level heuristics to solve the problem, can select and apply an appropriate low-level heuristic at each decision point. Broadly speaking, hybrid planning falls into the category of hyper-heuristics since a planning problem (i.e., a search problem) is solved by selecting appropriate planners and applying them. To solve search problems (e.g., SAT solving), IBL is one of the approaches commonly proposed by the hyper-heuristics community [25].

The rest of the paper is organized as follows: in Section II, we introduce a motivating example that will be used throughout the paper to explain our IBL approach; Section III describes the IBL approach; in Section IV, we present preliminary evaluation results; Finally, we conclude in Section V.

II. Motivating Example

This section presents the motivating example, which was also used in our previous work [16]. Suppose we have a cloud-based web application, which has a typical three layered architecture: a presentation layer, an application layer, and a database layer. When the application layer receives a client request from the presentation layer, it processes the request and exchanges data with the database layer if required. We will assume that the system has servers of different capacity;[1] however, the cost of a server increases with an increase in capacity. The workload on the system depends on the request arrival rate, which is uncertain as it depends on external demand.

The system needs to optimize profitability by maximizing revenue and minimizing operating cost; the system has various adaptation tactics to achieve these objectives. To maximize revenue, it is desirable to maintain the response time for user requests below some threshold (say T), since higher perceived user response time results in revenue loss [4][6]. Typically, an increase in request arrival rate would cause a higher response time perceived by clients. In such situations, the system can add more servers (using the *addServer<type>* tactic) to handle the workload; however, adding servers would increase operating cost. To manage cost, the system has an adaptation tactic *removeServer<type>* to deactivate an extra server.

In addition to adding servers, system response time can be controlled through *brownout*, which reduces the amount of optional content (e.g., advertisements or product recommendations) [8]. Such optional content generates revenue, but requires more computation power and network bandwidth, increasing response time [7]. The system provides a way to control this by providing tactics *increaseDimmer* and *decreaseDimmer* which raise or lower the probability that a request will contain optional content – if the value of the dimmer setting is high, a majority of requests will be served optional content; if it is low, fewer requests will have that content.

Since the system has servers of different capacity, a round-robin strategy for assigning client requests to active servers would not be efficient. The number of client requests delegated to a server depends on its capacity. The load-balancer uses queueing theory [14] to decide on the optimal load-distribution among the active servers. To distribute the load efficiently, there is a tactic, *divert_traffic*, which helps the load-balancer manage the percentage of client requests assigned to each server.

We assume there is a penalty, say P, for each request having a response time above the threshold. Therefore, in case of a high average response time, the system needs to react quickly either by adding servers or decreasing the dimmer value. However, once response time is under control, the system should execute adaptation tactics to bring down the operating cost in order to maximize overall long term utility.

The goal of the system is to maximize the utility, which depends on revenue generated, the penalty for response time above the threshold, and the cost of active servers. If the system runs for duration L, utility can be defined as:

$$U = R_O x_O + R_M x_M - P x_T - \sum_{i=1}^{n} C_i \int_0^L s_i(t)dt \qquad (1)$$

where R_O and R_M is revenue generated by a response with optional and mandatory content respectively; x_O, x_M, and x_T are the number of requests with optional content, only mandatory content, and having response time above the threshold; C_i is cost of server type i and s_i is number of active servers of type i; n is number of different types of servers.

III. Approach

There are two key challenges to realize hybrid planning for decision-making in self-adaptive systems. First, ensuring a seamless transition from a reactive plan to a (possibly) higher quality deliberative plan. Second, finding an effective arrangement of reactive and deliberative planning to solve a planning problem.

To find a balance between quality and timeliness, hybrid planning relies on transitioning from a low-quality reactive plan to a higher quality deliberative plan. To ensure such a transition, we rely on two features of the formal model defining the problem of hybrid planning [17]. These features are: (a) deliberative planners generate a universal plan i.e., a plan contains state-action pairs corresponding to all the reachable states from the initial state, where a mapping from a state (say s) to an action (say a) suggests a be executed in s; and (b) the operating domain is *Markovian* i.e., the state after a transition only depends on the current state – not on the sequence of states that preceded it [20]. The combination of these two features ensures that if reactive and deliberative planning have the same initial state, once the deliberative plan is ready, it can take over the plan execution from the reactive plan because any state in the reactive plan will be in the deliberative plan.

When reactive and deliberative planning is combined to instantiate a hybrid planner, the other key challenge is to decide whether reactive planning should be invoked for a planning problem. Assuming that deliberative planning provides better plans compared to reactive planning, when an adaptation is required a self-adaptive system has one of the two choices: (a) use reactive planning to provide a quick response, but switch to a deliberative plan once it is ready, or (b) invoke deliberative planning and wait until a plan is ready (i.e., do not use reactive

[1] As measured by average number of requests handled per second.

planning). Both choices have contexts in which they might be appropriate. In the context of cloud-based system (introduced in Section II), the first choice could be useful in dealing with emergency situations such as response time constraint violation. However, for some situations (e.g., sudden short-term drop in the request arrival rate), reactive planning might suggest a low-quality plan (e.g., remove a server), which would negatively impact the system's utility; therefore, the second choice might be preferred over the first one. For hybrid planning to be broadly applicable, an effective approach is needed to select between the two choices.

One approach is to invoke reactive planning only when a constraint is violated [13]. In the context of a cloud system, on a response time violation the system would invoke reactive planning to provide a quick response (say *addServer*) to the violation. However, while a new server becomes active, deliberative planning would determine a higher quality plan that would take over execution to improve long term utility.

As demonstrated by our previous work, invocation of reactive planning based on predefined conditions (e.g., constraint violations) can be useful. However, it limits the scope of hybrid planning to self-* properties such as self-healing. In contrast to self-healing systems, for instance, self-optimizing systems continuously improve utility (i.e., adaptation not limited to constraint violations) [21]. For self-optimizing systems, identifying a fixed set of conditions at design time could be difficult. Moreover, for such systems, invoking reactive planning only on constraint violations would limit the use of reactive planning, and thus the potential of hybrid planning would not be fully realized.

To support a broad application of hybrid planning, this paper proposes an instance-based learning (IBL) approach to decide whether reactive planning should be invoked for a planning problem. In this approach, for a planning problem, the decision to invoke reactive planning depends on the performance of reactive planning on a similar planning problem seen earlier.s

The proposed IBL approach has an offline and an online phase. During the offline phase, we evaluate the hybrid planner against a set of planning problems similar to the ones expected at run time. The offline phase helps to determine when it is effective to invoke reactive planning in combination with deliberative planning compared to just invoking deliberative planning. When a system faces a planning problem at run time (i.e., the online phase), it invokes deliberative planning since it is likely to provide a high-quality plan when the planning is complete. Meanwhile, to decide whether reactive planning needs to be invoked, a problem similar to the current problem is found from the offline phase. For this similar problem, if reactive planning improved the utility (compared to waiting for deliberative planning to complete), reactive planning is invoked for the current problem too, otherwise the system waits until a deliberative plan is ready.

A. The offline phase

The offline phase helps to build the performance model of a hybrid planner by profiling it against a set of planning problems that the system expects to observe at run time; here, for a planning problem, profiling refers to evaluating the hybrid planner for two cases i.e., whether reactive planning is invoked (in combination with deliberative planning) or not. For a planning problem realized at run time (i.e., the online phase), this performance model helps the system to decide whether to invoke the reactive planner or wait until a deliberate plan is ready. More formally, the goal of the offline phase is to approximate the function $Y : \Xi \rightarrow \{useReactive,\ notUseReactive\}$ suggesting whether reactive planning should be invoked for a planning problem $\xi \in \Xi$. Here Ξ is the set of all planning problems for the system.

The offline phase has two steps: (a) identify sample planning problems to profile the hybrid planner; and (b) profile the hybrid planner against these sample problems.

1) Identifying Sample Problems: For a self-adaptive system, comprehensive coverage of a planning problem space is critical to get a better estimation of the performance model of a hybrid planner; however, identifying a good set of representative problems is challenging due to a potentially infinite problem space. Unfortunately, there is no ideal solution to this problem as it requires an appropriate abstraction of the problems space; this abstraction depends on system specific requirements.

For our preliminary investigation for IBL approach in the context of the cloud-based system in Section II, we classified the problem space based on the current response time and the future workload pattern. Since the cloud system has a penalty for the response time constraint violation (i.e., maintaining response time is a primary concern than reducing operating cost), we divide the problem space along the dimensions of current response time and future workload pattern.

2) Profiling the Hybrid Planner: After determining a set of planning problems that reasonably cover the planning problem space, the next step is to evaluate the hybrid planner against these problems. For evaluation, first we solve a sample planning problem (say ξ_s) using the deliberative planner; suppose a plan π_d is determined in time t_d. Second, we solve the same problem with the reactive planner, which determines a plan π_r in a negligible time.

To evaluate the hybrid planner against the sample problems, we use a probabilistic model-checker – in this case Prism [26]. For each sample problem, there are two evaluations corresponding to the two possible choices (i.e., use or not use reactive planning) for the hybrid planner. In the first evaluation, for a sample problem we model-check the combination of the reactive plan and the deliberative plan. In the second evaluation, for the same problem we model-check the case when the system waits until the deliberative plan is ready. Then we compare the two results to decide if reactive planning should have been applied to the problem.

The first evaluation has two steps: (a) until time t_d, reactive plan π_r is executed, and (b) from time t_d onwards, deliberative plan π_d is executed. Suppose on simulating this combination of the reactive and the deliberative plan, we get utility U_c. This evaluation represents the case when, initially, reactive planning is invoked to solve ξ until the deliberative plan (i.e., π_d) is ready to take over the execution.

The second evaluation represents the case when a system does not invoke reactive planning to solve ξ_s, but rather waits for the deliberative plan to be ready. This evaluation has two steps: (a) no action until time t_d, and (b) from t_d onwards, execute the deliberative plan π_d.

Finally, we compare the utilities U_c and U_d to decide if reactive planning should be invoked (or not) to solve ξ_s; if $U_c > U_s$ then invoke the reacting planning (i.e., $Y(\xi_s) = useReactive$) otherwise wait for the deliberative plan to be ready (i.e., $Y(\xi_s) = notUseReactive$).

B. The online phase

When a system senses an adaptation opportunity (i.e., a planning problem, say ξ_r) at run time, it needs to decide if invoking reactive planning would improve utility. In the context of IBL, this decision-making problem consists of two sub-problems: (a) find planning problems from the profiling stage that are similar to ξ_r, and (b) decide whether to invoke reactive planning based on its performance on the matching problems.

Often the system will not encounter precisely the problems that were covered in the offline phase. Therefore, we need a way to determine if the current state of the system is similar to a state we observed in the offline phase. Formally, we need to define a function $S : \Xi \times \Xi \rightarrow \mathbb{R}$ that takes two planning problems as an input and returns a real number indicating the similarity between the planning problems. To calculate similarity between two planning problems, one can develop a metric that considers various aspects (e.g., the similarity between current states of the system). We can then use this metric to find training instances that are similar to the current problem and use some heuristic to decide whether to invoke reactive planning. For example, in the simplest case, we could choose based on the instance closest to the current instance, or we could choose based on whether the majority of the N-closest instances invoked reactive planning, or something else.

Defining a good similarity metric is itself a challenging problem, and depends on the operating domain. In our experiments using the cloud-based system, we use current response time (i.e., above or below the threshold) and the predicted trend of request arrival.

IV. Proof of Concept

The purpose of our experiments is to demonstrate that if two planning problems are similar, then the decision to invoke reactive planning for one should also apply to the other. Here is an overview of our methodology to evaluate the proposed IBL approach: First, in the context of the cloud-based system in Section II, we synthesize a pair of planning problems (i.e., adaptation opportunities) that are similar (but not exactly same) in terms of current response time (e.g., above or below the threshold) and future workload pattern. We choose one problem from the pair to conduct profiling of hybrid planner as discussed in Section III-A2. Suppose, profiling of the problem shows that a combination of reactive and deliberative planning provides a higher utility compared to the case when the system waited until a deliberative plan is ready. Then, for the second problem in the pair, we simulate both the cases (i.e., reactive planning invoked and not invoked) on the actual system. If simulation results are consistent (i.e., invoking reactive planning provides higher utility) with the profiling result for the first problem, then it is an indication that IBL could be effective in deciding whether reactive planning should be invoked.

A. Experimental Setup

To bring self-adaptive capability to our cloud-based system, we implemented a MAPE-K loop [12] using a discrete event simulator, *OMNeT++* [3]. In the simulator, we implemented various architectural components such as a load-balancer, and different types of servers. Besides the fundamental functionality, these components also have logic to support the tactics described earlier. For instance, servers allow increments/decrements of dimmer values and the load-balancer allows addition/removal of servers.

Similar to our previous work, we instantiated hybrid planning using deterministic (i.e., reactive) and MDP (i.e., deliberative) planning [16]. To model environmental uncertainty for MDP planning, we use a time-series predictor to anticipate future workload on the system [10].

The system has three types of servers: A, B and C. Server type-C is the costliest, but has the highest capacity in terms of its ability to handle requests: server type-A is the cheapest, but has the least capacity. We assign a cost per minute to be \$0.5, \$0.7, and \$1 for server type-A, type-B, and type-C, respectively. The capacity of server type-A is 50 when serving the optional content and 150 without serving the optional content; the capacity of server type-B is 130 when serving the optional content and 200 without serving the optional content; and the capacity of server type-C is 150 when serving the optional content and 300 without serving the optional content. Table I summarizes the simulation parameters for the three types of server.

Server-type	Cost ($ per minute)	Capacity (requests per second)	
		With Optional Content	Without Optional Content
A	0.5	50	150
B	0.7	130	200
C	1.0	150	300

TABLE I. Simulation parameters for the three servers

Suppose the cost of a server can be covered by the revenue of handling 1/10 of its maximum capacity with optional content and the revenue of handling 2/3 of its maximum capacity without optional content. If the server cost per minute is C, capacity with optional content is c_O and without optional content is c_M, then the revenue for a server with optional content is $R_O = \frac{10}{c_O}C$ and without optional content would be $R_M = \frac{3/2}{c_M}C$. For each request having response time above the threshold of 1 second, there is a penalty of -3 units.

For the experiments, we have 3 dimmer levels and 1 server of each type i.e. the total number of servers is 3. We assume a fixed boot-up time of 2 minutes for each server type. Since at any point in time, we can have a maximum of 2 inactive servers, and each server requires 2 minutes of boot up time, our look-ahead horizon for MDP planning is chosen to be 5 minutes. This heuristic gives a long enough horizon to go from 1 active server to 3 active servers plus 1 additional evaluation cycle to observe the resulting utility.

To formulate pairs of similar planning problems, as discussed later we use two sections (highlighted as Region-1 and Region-2 in Fig. 1) of the request arrival trace from the World Cup '98

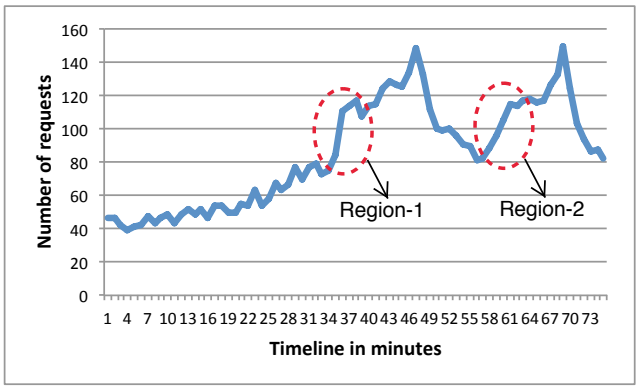

Fig. 1: Request arrival rate for a trace used as an input.

website [11]. This trace is scaled down so the request arrival rate does not exceed the capacity of the simulation. Notice that the workload pattern for these sections is similar, where load increases from about 80 requests per second to about 120 requests per second over 5 minutes.

The cloud-based system operates in a self-optimization mode. To explain further, suppose the system takes the trace shown by Region-2 as an input. The goal of the system is to maximize the aggregate utility (for Region-2), which is the sum of utility accumulated at the end of each minute; since the trace duration is 5 minutes, the utility of the system is calculated 5 times for the trace and finally these utilities are summed up to get the aggregate utility. To find an adaptation plan, the system can either use the combination of reactive and deliberative planning or use only deliberative planning.

We conducted the experiments on a Ubuntu 14.04 virtual machine having 8GB RAM and 3 processors at 2.5 GHz. The state space for the MDP planning varies approximately between 1.6 million and 2.2 million and the MDP planning time varies between 35-45 seconds. The state space for the deterministic planning is less than 100K states with planning time less than a second, which is considered negligible in the experiments.

B. Results

To show the usefulness of IBL, we experimented with two scenarios. Each scenario has a source planning problem that we profile and a similar planning problem (i.e., the target problem) that we apply the results to. If IBL is useful, it should help to decide whether reactive planning should be applied to the target problem.

1) Scenario-1: In this scenario, the source planning problem (Pb_1) plans for a situation where the response time is above the constraint threshold, and the workload is predicted to continue increasing. This problem corresponds to Region-1 in Fig. 1. In this planning problem, the current state of the system has an active server of type A and dimmer level of 0 (i.e., optional content is being served).

We profile the planner in the cases of doing reactive planning and not doing it. For this situation, the reactive planner suggests that increasing the dimmer and adding a server of type-C will sufficiently bring down the response time. The deliberative plan (which takes time to compute) suggests decreasing the dimmer since the newly added type-C server (as suggested by the reactive plan) is sufficient to handle the expected increase in workload. When we model check the profile for this situation, we find that combining reactive and deliberative planning results in better aggregate utility than waiting for the deliberative plan and doing that alone.

For the target planning problem (Pb_2), we use the second region of Fig.1 which also has high response time and a predicted increase in workload, but has different a number of requests. We simulate this, and in Fig. 2 it can be seen that if we were to use deliberative planning only in this case, we get worse aggregate utility (normalized, it is -1) than if we did what IBL suggests and use reactive planning in combination with deliberative planning, with a normalized utility of -0.5.

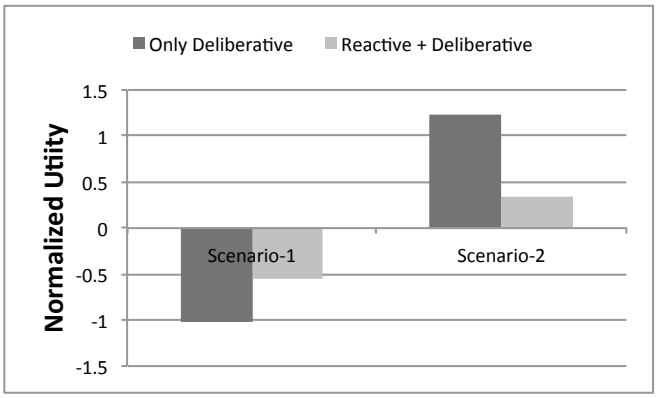

Fig. 2: Normalized aggregate utility for different approaches when handling the second problem in the two scenarios.

2) Scenario-2: In this scenario, the source planning problem (Pb_3) plans for a situation where the response time is below the constraint threshold, but the workload is predicted to continue increasing. This problem corresponds to Region-1 in Fig. 1. Although the same region (i.e., Region-1) is used for both Pb_1 and Pb_3, in contrast to Pb_1, problem Pb_3 has response time under the threshold since more resources are initially deployed by the system. For Pb_3, the current state of the system has 2 active servers (one each of type-A and type-B) and the dimmer level is at 3 (i.e., no users get optional content).

Similar to Scenario-1, we profile the planner in the cases of doing reactive planning and not doing it. For Pb_3, the reactive planner suggests removing the server of type-B, which is the wrong decision considering the predicted increase in workload going forward. When we model check the two situations, we find that combining reactive and deliberative planning results in worse aggregate utility than waiting for the deliberative plan and doing that alone.

For the target planning problem (Pb_4), we use Region-2 of Fig.1. Like Pb_3, problem Pb_4 has response time below the constraint threshold and a predicted increase in workload. For problem Pb_4 the system has 2 active servers (one each of type-A and type-C) and the dimmer level is at 3. We simulate this, and in Fig. 2 it can be seen that if we were to use reactive planning in combination with deliberative planning in this case, we get worse aggregate utility (normalized, it is about 0.4) than if we did what IBL suggests i.e., use only deliberative planning, with a normalized utility of 1.20.

V. Conclusion

As a proof of concept, we provide two specific instances that indicate that IBL could be useful in the context of hybrid planning. Although our experiments indicate the potential of IBL approach, further investigation is required to determine the effectiveness of the approach. To this purpose, first we need to find a reasonable set of profiling problems that broadly cover the problem space. For this, we plan to use the traces used by Gandhi et al. [22] that are claimed to cover major types of load patterns for cloud-based systems; different load patterns could help to cover (a large part of) the problem space.

The next challenge is to develop a similarity metric that helps in identifying similar planning problems. This metric would be used to automate the process of finding similar problems during the online phase. Specifically, we plan to explore techniques to calculate similarity between time-series (i.e., workload) patterns in order to identify similar planning problems [23]. However, a major challenge for the online phase is to keep it efficient both in terms of correctness and time consumed to find similar problems. Although, going forward, we plan an extensive validation for IBL approach, this paper lays a foundation for a meaningful discussion among the self-adaptive community.

Acknowledgments

This work is supported by award FA87501620042 from the Air Force Research Laboratory (AFRL). Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the AFRL or the U.S. Government.

References

[1] https://aws.amazon.com/ec2/sla/

[2] http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using- aws.html

[3] https://omnetpp.org/

[4] G. Linden. Make Data Useful. Amazon, 2009

[5] D. H. Wolpert and W. G. Macready. Coevolutionary free lunches. IEEE Transactions on Evolutionary Computation,, vol. 9, pp. 721-735, 2005

[6] W. Lloyd et al. Stronger semantics for low-latency geo-replicated storage. In 10th USENIX Symposium on Networked Systems Design and Implementation, pages 313-328. USENIX Association, Apr. 2013

[7] M. B. Dias et al., The value of personalised recommender systems to e-business. In Proceedings of the 2008 ACM Conference on Recommender Systems - RecSys '08, page 291, New York, New York, USA

[8] C. Klein et al., Brownout: building more robust cloud applications. In Proceedings of the 36th International Conference on Software Engineering - ICSE 2014, pages 700-711, New York, New York, USA

[9] D. Sykes et al., Plan-Directed Architectural Change For Autonomous Systems. Sixth International Workshop on Specification and Verification of Component-Based Systems (SAVCBS 2007), September 3-4, 2007, Cavtat near Dubrovnik, Croatia, 2007

[10] G. Moreno et al., Proactive Self-Adaptation under Uncertainty: A Probabilistic Model Checking Approach. ESEC/FSE-15, August 30 - September 4, 2015, Italy

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

[12] J. O. Kephart and D. M. Chess, "The vision of autonomic computing", Computer, vol 36, issue 1, Jan 2003. doi:[10.1109/MC.2003.1160055]

[13] O. Cheng et al., Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. 1st International Conference on Autonomic Computing (ICAC 2004)

[14] Mor Harchol-Balter, Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge University Press

[15] Z. Coker et al. SASS: Self-Adaptation Using Stochastic Search. Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems Florence, Italy 2015, pp. 168-174

[16] A. Pandey et al., Hybrid Planning for Decision Making in Self-adaptive Systems. In International Conference on Self-Adaptive and Self-Organizing Systems, ser. SASO 2016, 2016, pp. 12-16

[17] A. Pandey et al., Towards a Formal Framework for Hybrid Planning in Self-Adaptation. In 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS-17

[18] P. Jamshidi et al., Autonomic resource provisioning for cloud-based software. In 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS-14

[19] Barry Porter and Roberto Rodrigues Filho, Losing Control: The Case for Emergent Software Systems using Autonomous Assembly, Perception and Learning. In International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2016

[20] Mausam and Andrey Kolobov, Planning with Markov Decision Processes: An AI Perspective. Synthesis Lectures On Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, June 2012

[21] Andrew Berns and Sukumar Ghosh, Dissecting Self-* Properties. Proceedings of the 2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO '09), Pages 10-19

[22] Gandhi et al., Autoscale: Dynamic, robust capacity management for multi-tier data centers. TOCS, 2012

[23] Pablo Montero and Joše A. Vilar, TSclust: An R Package for Time Series Clustering. Journal of Statistical Software, November 2014, Volume 62

[24] E. Burke et al., Hyper-Heuristics: An Emerging Direction in Modern Search Technology. Handbook of Metaheuristics, Vol. 57, July 2003

[25] Lars Kotthof, Algorithm Selection for Combinatorial Search Problems: A Survey. AI Magazine. Fall 2014, Vol. 35 Issue 3, p48-60. 13p

[26] M Kwiatkowska et. al., PRISM 4.0: Verification of Probabilistic Real-time Systems. In Proc. 23rd International Conference on Computer Aided Verification (CAV'11), volume 6806 of LNCS, pages 585-591, Springer, 2011

[27] Arshad et al., Deployment and Dynamic Reconfiguration Planning For Distributed Software Systems. Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03)

[28] Sima Soltani et al., QuARAMRecommender: Case-Based Reasoning for IaaS Service Selection. 2014 IEEE International Conference on Cloud and Autonomic Computing 978-1-4799-5841-2/14 315.00 DOI 10.1109/ICCAC.2014.26220

[29] D. Aha et al., Instance-Based Learning Algorithms. Machine Learning, 6, 37-66 (1991)