Thesis Proposal
Hybrid Planning in Self-Adaptive Systems

Ashutosh Pandey
Institute for Software Research
School of Computer Science
Carnegie Mellon University
ashutosp@cs.cmu.edu

December 2017

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Thesis Committee

Dr. David Garlan (Chair)
Institute for Software Research
Carnegie Mellon University

Dr. Hausi Müller
Department of Computer Science
University of Victoria

Dr. John Dolan
The Robotics Institute
Carnegie Mellon University

Dr. Jonathan Aldrich
Institute for Software Research
Carnegie Mellon University

Copyright © 2017 Ashutosh Pandey
Abstract

Self-adaptive software systems determine adaptation plans at run time that seek to change their behavior in response to faults, changing environments and attacks. Therefore, having an appropriate planning approach to find an adaptation plan is critical to successful self-adaptation.

For many realistic systems, ideally one would like to have a planning approach that finds quality plans in a timely manner. However, due to the fundamental trade-off between quality and timeliness of planning, today designers often have to compromise between an approach that is quick to find a plan and an approach that is slow but finds a quality plan.

To deal with this trade-off, we propose a hybrid planning approach for self-adaptive systems that combines deliberative and reactive planning to find a balance between quality and timeliness. The key idea is to use reactive planning to provide a quick (although potentially a sub-optimal) response, but simultaneously invoke deliberative planning to determine quality plans. Once the deliberative plan is ready, it takes over the execution from the reactive plan to provide a higher quality adaptation thereafter.

The proposed thesis work will demonstrate through case-studies that a combination of reactive and deliberative planning can improve adaptation effectiveness over using either alone as measured by a multi-dimensional utility function capturing different dimensions of a system’s goal. In the process, the thesis will make contributions to both the theory and the practice of hybrid planning in self-adaptive systems. Specifically, the thesis will provide: (a) a formal framework defining the problem of hybrid planning; (b) a practical approach (grounded on the formal model) to apply hybrid planning to self-adaptive systems; and (c) concrete examples bridging the gap between theory and practice.
## Contents

1 Introduction 1

2 Thesis 4
   2.1 Thesis Statement 5
   2.2 Expected Contributions 5

3 Motivating Example 6

4 Problem Description 9

5 Proposed Approach 12
   5.1 Constructing a Reachability Graph 14
      5.1.1 Restricting the Number of Nodes 14
      5.1.2 Connecting the Nodes 15
   5.2 Finding a Path in a Reachability Graph 17
      5.2.1 The Offline Phase 18
      5.2.2 The Online Phase 21
   5.3 Discussion 21

6 Preliminary Work 22
   6.1 Hybrid Planning for Self-healing 22
   6.2 Formalization of the Hybrid Planning Problem 23
   6.3 Hybrid Planning for Self-optimization 23

7 Related Work 23
   7.1 Automated Planning 23
   7.2 Frameworks to Support Automated Planning 24
   7.3 Hyper-Heuristics 25
   7.4 Planning in Self-Adaptive Systems 25

8 Research Plan 26
   8.1 Validation 26
   8.2 Risks 27
   8.3 Timeline 28

References 30
1. Introduction

A typical control loop in many self-adaptive software systems has four fundamental computational components: Monitoring-Analysis-Planning-Execution (MAPE) [36]. Based on information collected by the monitoring component, if the analysis component decides that system needs to adapt to meet its goals, then the planning component determines an adaptation plan, which is executed by the execution component.

The planning component determines an adaptation plan based on various factors such as the current state of a self-adaptive system and its operating environment, a set of possible adaptation actions, and the adaptation goal; such factors together constitute a planning problem (as formally explained in Section 4). In other words, the planning component takes a planning problem as an input and returns an adaptation plan.

For the planning component, researchers in the self-adaptive community have proposed various approaches to determine plans. Frameworks such as Rainbow [15] use expert knowledge to determine adaptation plans; when adaptation is needed, Rainbow chooses an adaptation strategy (i.e., a plan) from a predefined repertoire, which was created at design time by domain experts based on their past troubleshooting experience. In addition, researchers have demonstrated the potential of other techniques, such as reinforcement learning [37, 57], case-based reasoning [63], genetic algorithms [16], and fuzzy logic [33] that (similar to using expert knowledge) generate adaptation decisions offline but choose them at run time. In contrast to generating adaptation decisions offline, various automated planning techniques have been explored to generate adaptation plans at run time [13, 49, 65].

For the design of a MAPE-based system, an appropriate instantiation of the planning component is both critical and non-trivial. An appropriate instantiation is critical since it impacts the ability of a planning component to determine adaptation plans, and thus a system’s potential to meet adaptation goals. An appropriate instantiation of the planning component is non-trivial since there are numerous planning approaches, each having its own set of characteristics [26]; expertise is needed to identify and implement an approach that best meets the requirements.

For many self-adaptive systems, quality and timeliness are two particularly important requirements to be considered when planning. Here “quality” of planning refers to the likelihood of a plan meeting the adaptation goals under the assumption that the plan is available instantaneously, when required. For many domains, such as safety-critical systems, quality of planning is important, especially since a bad plan could lead a system to an irreparable failure state that endangers lives. In other domains such as enterprise system, poor quality plans can hinder in meeting business goals.

In addition to quality, finding an adaptation plan in a timely manner is another important requirement for planning [59]. For instance, after a security threat detection, if an enterprise system fails to determine a defense plan in a timely manner, there is a risk that the system might be compromised, resulting in failure to meet the goal of self-protection.

Many systems need both – quick planning under urgent circumstances, but over the long term performance should be optimized. Ideally, for such systems, a planning approach is needed that can find optimal adaptation plans in a timely manner. For instance, Amazon Web Services (AWS) are required to maintain an up-time of at least
99.95% in any monthly billing cycle as per the service level agreement, balancing it with other concerns such as cost minimization.\(^1\) The perceived effectiveness of such systems will drop drastically if their service-level constraints are violated. In the case of a constraint violation, a rapid response is required to keep the system in a desirable state (for AWS, maintaining availability). However, to maintain long-term quality, an adaptation plan should be as close to optimal as possible, by considering other metrics (e.g., operating cost) as well. Netflix is another example of such a system, where managing the overall latency of response to clients is critical to good user experience, in spite of the desire to minimize resource usage, and thus to lower operating cost.\(^2\)

Unfortunately, for a planning approach, quality and timeliness are potentially conflicting requirements. Planning, in essence, is a search/optimization process performed over the space of possible plans – more complete searches provide better quality guarantees, but require more time to complete. Hence, for urgent situations a planner can either provide a sub-optimal plan at the moment when it is needed, or provide a higher-quality plan, risking it being late. Moreover, this imbalance between quality and timeliness increases significantly with the increase in a search space that arises in the presence of large numbers of components, adaptation options, and multiple qualities of interest.

As a consequence, when choosing an off-the-shelf planning approach, self-adaptive systems today must compromise between one of the two requirements leading to systems that typically can either respond quickly, or provide a high-quality adaptation but not both (refer to Figure 1). Within the self-adaptive systems community, research has primarily focused on the quality of planning; timeliness of planning, in general, has not been treated as a first class concern [45].

![Figure 1](https://aws.amazon.com/ec2/sla/)

**Figure 1.** A notional representation of a space for planning approaches in domains with uncertainty. Ideally, one would like to move towards the desired region i.e., high plan quality with low planning time

One direction, explored by the artificial intelligence (AI) community, is to develop customized planning solutions applicable to a particular domain or a narrow class of

---

\(^1\)https://aws.amazon.com/ec2/sla/

\(^2\)http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html
planning problems, since such solutions can exploit specific knowledge about the search space. While there are successful instances of such customized solutions [9, 32, 34], developing them is a non-trivial task since it requires deep understanding of the operating domain and experience in planning technology. Thus, even though such customized solutions can provide strong support for adaptation tailored to a particular domain they are time-consuming, and hence costly to construct. Furthermore, the success of such solutions varies from one problem to another, limiting their generality. This fact is consistent with the No Free Lunch Theorem: for any search/optimization algorithm, an elevated performance over one class of problems is paid for in performance over another class [68, 69, 70].

In contrast to inventing a domain/problem-specific planning approach, in this thesis we propose a novel idea of hybrid planning for self-adaptive systems that combines an off-the-shelf deliberative and reactive approach to address the trade-off between quality and timeliness of planning. The key idea is to use reactive planning to provide a quick (but potentially sub-optimal) response to a problem, but simultaneously invoke deliberative planning that is likely to provide a higher-quality plan (compared to reactive planning). Once the deliberative plan is ready, it takes over execution from the reactive plan to provide a higher-quality adaptation thereafter. Intuitively, using deliberative planning alone introduces a gap in response time that can lower utility of a system. On the other hand, using reactive planning alone provides a quick response but it may not be nuanced enough to provide quality plans, particularly in the long run.

Hybrid planning has a number of potential advantages over custom planning solutions. Instead of going through the non-trivial process of developing a new algorithm/heuristic, hybrid planning combines off-the-shelf planning approaches. Using existing approaches is likely to reduce development time and cost since software engineers don’t have to deal with the complexity of developing new algorithms/heuristics and do not need to become experts in AI planning. Moreover, by combining a reactive and a deliberative planning approach, hybrid planning raises the level of generality at which a planning problem is solved [12]. To explain further, hybrid planning relies on finding an appropriate combination of existing planning approaches depending on the domain/problem. In a sense, hybrid planning can be thought of as meta-planning, which operates on a set of off-the-shelf planning approaches; therefore, hybrid planning is more general compared to developing domain/problem-specific solutions.

Our preliminary work has already demonstrated the potential of hybrid planning [52]. Specifically, for the cloud-based system outlined in Section 3, we instantiated hybrid planning using a particular choice of reactive and deliberative planners. Reactive planning was invoked in case of an emergency situation, specifically, a constraint violation. In that context, the experiments demonstrated that the combination of reactive and deliberative planning performs better than deliberative planning used in isolation. Then, we went a step further to formally define the hybrid planning to describe its general nature [53]. In Section 5, this formalism is used to explain our approach to apply hybrid planning in a realistic context; solutions grounded on a formal model give us confidence that all relevant challenges are addressed. Moreover, this formal model sets the stage for going beyond the solution proposed for this thesis to find even better solutions to hybrid planning. Furthermore, this model serves as a unifying evaluation framework for different such solutions.
Based on our initial work, we realized that applying hybrid planning more generally (i.e., different domains) and flexibly (i.e., different combinations of reactive and deliberative planning) requires addressing two research challenges:

- **Planning Coordination** \( (\text{PlnCr}) \): to find a balance between quality and timeliness, hybrid planning relies on transitioning from a low-quality plan to a higher-quality plan. However, guaranteeing a seamless transition between two plans is a challenge.

- **Planning Selection** \( (\text{PlnSel}) \): when a reactive and a deliberative planning is combined to instantiate hybrid planning for a particular system, it is important to figure out which approach(es) should be invoked to solve a planning problem observed at run time. Assuming deliberative planning provides better plans (over reactive planning), it could always be invoked to solve a problem. However, reactive planning cannot be blindly invoked since a potentially sub-optimal reactive plan could lead a system to a permanent failure state; therefore it is crucial to decide when to invoke reactive planning (along with deliberative planning).

The rest of the proposal is organized as follows: Section 2 presents the thesis statement; Section 3 outlines a motivating example that will be used throughout the proposal to explain challenges and the proposed approach; Section 4 provides an overview of the formal model defining the problem of hybrid planning; Section 5 describes our solution approach to hybrid planning; Section 6 highlights our preliminary work; Section 7 discusses the related work; finally, Section 8 outlines our research plan going forward.

2. **Thesis**

This research will improve the current state-of-the-art for planning in self-adaptive systems. Due to the trade-off between timeliness and quality of planning, when choosing a single planning approach, designers have two choices: (a) make an offline (i.e., at design time) compromise between finding adaptation plans quickly and finding quality plans demanding longer computation times, or (b) deal with the complexity of developing a customized planning approach that is likely to consume time and resources. Therefore, rather than choosing a single approach, we propose a hybrid planning approach that combines off-the-shelf deliberative and reactive planning to find a balance between quality and timeliness without incurring the overhead of developing a customized planning approach. Moreover, in the context of self-adaptive systems, we will show that a hybrid planning approach could be applied with the following three qualities:

- **Effectiveness**: Hybrid planning will improve the effectiveness of a self-adaptive system compared to reactive or deliberative planning approach used alone. Effectiveness is a measure of a system’s ability to meet its adaptation goal. In this thesis, we assume that the system’s adaptation goal is encoded in a multidimensional utility function that captures various quality dimensions of the goal.

To validate effectiveness, we will demonstrate that hybrid planning provides higher utility. While the amount of improvement depends on the context, we expect to show a significant improvement in a variety of natural contexts as we have shown in our preliminary work (refer Section 6).
• **Generality**: Hybrid planning will be general enough to be applied to self-adaptive systems operating in domains that differ in: (a) quality dimensions of concern, (b) the cost of delayed action, and (c) the reversibility of adaptation actions.

To validate generality, we will evaluate hybrid planning on two kinds of system: (a) AWS/Netflix-like systems, where even if the system fails to maintain a critical constraint (e.g., availability/response-time) due to a poor or a delayed decision, it can still recover back to a desired state later, and (b) safety-critical systems such as unmanned aerial vehicles (UAV), where a failure to avoid a crash (i.e., safety constraint) due to a low-quality or a delayed decision, could lead to a mission failure.

• **Flexibility**: Hybrid planning will be flexible enough to be instantiated using different combinations of reactive and deliberative planning approach. Two combinations will be considered different if either the reactive or the deliberative approaches are different in the combinations.

To demonstrate flexibility, a different combination of reactive and deliberative planning will be used for the two systems (as discussed above). For a combination, we will choose between: (a) deterministic and rule-based adaptation approaches for reactive planning, and (b) Markov Decision Processes (MDP) [47] and Partially Observable Markov Decision Processes (POMDP) [35] based planning for deliberative planning; the choice will depend on nature of uncertainty in the operating domain.

2.1. Thesis Statement

My thesis is:

*We can improve effectiveness of self-adaptive systems by using a hybrid planning approach, which is general and flexible. This approach has the following elements:*

- the use of off-the-shelf deliberative and reactive planning approaches to instantiate hybrid planning that can take advantage of both planning approaches to find a balance between quality and timeliness of planning;
- the ability to dynamically decide which constituent planning approach(es) of hybrid planning should be invoked to solve a planning problem – specifically, whether to invoke reactive planning along with deliberative planning.

2.2. Expected Contributions

Our main contribution will be to show that hybrid planning can be used to improve the current state-of-the-art for planning in self-adaptive systems by finding a balance between quality and timeliness of planning. More specifically, the proposed research will make contributions to both the theory and the practice of hybrid planning in MAPE-K based self-adaptive systems.

The contribution to theory is:

- a formal model characterizing the general problem of hybrid planning.

The contributions to practice are:
• a practical approach to apply hybrid planning under certain assumptions/restrictions that nonetheless apply to many self-adaptive systems.
• a demonstration of effectiveness, generality, and flexibility of hybrid planning for self-adaptive systems using the proposed solution approach;
• methods/tools to apply hybrid planning to self-adaptive systems;
• concrete examples bridging the gap between theory and practice;
• an informal taxonomy and guidelines to help software engineers to select planning approaches for instantiating reactive and deliberative planning.

3. Motivating Example

This section presents an example of a self-adaptive system, which will be used throughout the document to explain the research challenges and our proposed approach.

![High-level view for the cloud-based system](image)

Consider a cloud-based self-adaptive system, as shown in Fig. 2, with a typical N-tiered architecture: a presentation tier, an application tier, and a database tier. Using the presentation tier, a client sends a request to the application tier, which interacts with the database tier to process the request. We assume the system has different types of servers with varying capacity (i.e., ability to handle requests per second) and cost. Not surprisingly, the operating cost of a server increases with increase in capacity. The workload on the system depends on the request arrival rate, which is uncertain as it depends on external demand. Moreover, we assume that there is uncertainty in the system itself because of the possibility of random server crashes, which adversely impact the load bearing ability of the system.

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 [42, 43]. Typically, an increase in request arrival rate causes a higher response
time perceived by clients. In such situations, the system can add more servers (using tactic \texttt{addServer\{type\}}) to handle the increased workload; however, adding servers will also increase operating cost. To manage cost, the system has an adaptation tactic (i.e., \texttt{removeServer\{type\}}) to deactivate a server.

In addition to adding servers, system response time can be controlled through “brownout”, which reduces the amount of optional content (such as advertisements or product recommendations) [38]. Such optional content generates additional revenue, but requires more computation power and network bandwidth that increases response time [18]. The system provides a way to control this by providing the tactics \texttt{increaseDimmer} and \texttt{decreaseDimmer} which raise or lower the probability that a request will contain optional content — the number of requests with optional content decreases as the value of the dimmer setting increases.

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 [29] to decide on the optimal load-distribution among the active servers. To distribute the load efficiently, there is a tactic (i.e., \texttt{divert\_traffic\{traffic\_server\_1 \ldots traffic\_server\_n\}}, where \texttt{traffic\_server\_k} is the traffic for the \textit{k}th server and \textit{n} is the total number of servers), 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 response time, the system needs to react quickly either by adding servers or increasing 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 utility, which depends on the revenue generated, the penalty for response time above the threshold, and the cost of active servers. If the system runs for duration \( L \), the utility function can be defined as:

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

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

To determine adaptation plans for such a cloud-based system, researchers have suggested a variety of planning approaches such as rule-based adaptation (RBA) [15], case-based reasoning (CBR) [63], fuzzy-logic [33], reinforcement learning [57], stochastic search (e.g., genetic algorithms) [16], that, generally speaking, fall into the category of reactive planning. These approaches determine an adaptation plan quickly because the plan is not generated at run time, but rather selected from an existing set of plans; however, the selected plan might not be optimal because it is difficult to have an optimal plan that was determined offline. For instance, a system with a rule-based adaptation might have a hard-coded rule saying that whenever the response time constraint is vio-
lated, add a server with the highest capacity. This plan could be sub-optimal if the spike in client requests is temporary; by the time the newly added server is active, response time would be below the threshold, but the system ends up paying an additional cost for this server.

In contrast, for self-adaptive cloud systems, researchers have also proposed various deliberative approaches such as planning based on Markov decision processes (MDP) to dynamically generate plans [52]. MDP planning at run time can be slow but often provides high-quality adaptation plans for unexpected situations since the planning takes into consideration factors such as the current state of the system and its environment, predicted (but uncertain) values of future request arrival rate, and timing of tactic latency [49].

However, for this cloud-based system, using a single (i.e., either a reactive or a deliberative) planning approach can be problematic. A reactive approach such as RBA might provide a quick response to a response time constraint violation, and thus improve the system’s utility in short-term. However, the plan is likely to be sub-optimal due to uncertainty in the request arrival rate, which is difficult to predict at design time (i.e., when formulating the rule). On the other hand, if a deliberative approach such as MDP is used for planning, it is likely to provide high-quality plans but at the cost of slow response, which would be an issue, particularly for situations such as a response time constraint violation.

To find a balance between quality and timeliness of planning, a hybrid planner could be instantiated using a reactive (e.g., RBA) and a deliberative (e.g., MDP) planning approach. Balancing quality and timeliness would help in improving utility of the system. However, elaborating what was said earlier, there are two key challenges in combining different planning approaches:

- **Planning Coordination (PlnCrD):** To balance quality and timeliness, hybrid planning requires a smooth transition from a reactive plan to a possibly higher-quality deliberative plan. Suppose the system observes a response time constraint violation. As a result, assume reactive planning is invoked to provide a quick response to this problem. Now, for a seamless transition from the reactive plan to the deliberative plan, the latter needs to have provisioned for the system’s state after executing the reactive plan. This is challenging for two reasons: (a) uncertainty about deliberative planning time makes it difficult to predict when the deliberative plan will be ready to take over, (b) uncertainty in the system’s environment makes it difficult to predict the expected system state after executing the reactive plan.\(^3\)

- **Planning Selection (PlnSel):** Assuming that deliberative planning provides better plans compared to reactive planning, when an adaptation is required, a self-adaptive system has one of 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 planning).\(^4\) Both choices have contexts in which they might be appropriate. For instance, if reactive planning is based on rule-based adaptation, as discussed earlier

\(^3\)A state consists of system state and environment state.

\(^4\)The choice of using only reactive planning is not considered since if a deliberative plan is ready to take over, it will provide a higher utility compared to a plan determined by reactive planning.
there can be situations (e.g., constraint violations) when invoking reactive planning (i.e., the first choice) could improve system utility by providing a quick response. In contrast, as already exemplified, there can be situations (i.e., a temporary spike in request arrival rate) when invoking such a reactive planning could lower utility; in such situations, it is better to wait until the deliberative plan is ready, i.e., the second choice is preferred over the first one. For hybrid planning to be broadly applicable, an effective approach is needed to select between the two choices.

4. Problem Description

This section provides an overview of the hybrid planning problem that helps to explain its general nature. This overview is based on our work [53] that formalizes the problem of hybrid planning in the context of domains whose behavior is Markovian – the conditional probability distribution of future states of a system depends only upon the present state, not on the sequence of states that preceded it [47]. The formalism decomposes the hybrid planning problem into three computational sub-problems. Such a decomposition not only helps to explain the complex problem of hybrid planning in terms of the simpler sub-problems but also provides a framework to solve a hybrid planning problem since the problem can be solved by tackling the sub-problems.

A Planning Problem: Informally, in the context of a self-adaptive system, given a description of the initial (i.e., current) state of the system and environment, a description of the desired adaptation goals, and a set of possible adaptation actions, the planning problem is to synthesize a plan that is guaranteed (when applied to the initial states) to generate a state which meets the desired goals. Formally, a planning problem $Pb$ is a tuple $(S, s'_i, A, o, T, U_e)$, where $S$ is a set of states, $s'_i \in S$ is the initial state, $A$ is a set of actions controlled by a system (e.g., addServer), $o : S \rightarrow Z$ is a function that maps the set of states ($S$) to the set of actions ($Z$) controlled by environment (e.g., request arrival rate), $T$ is a transition function such that $T : S \times A \times Z \rightarrow S$, and $U_e$ is utility (i.e., a real number indicating quality of a plan) of executing a plan ($P : S \rightarrow A$) determined by a planner that takes a planning problem as an input. Solving a planning problem (using a planner) means optimizing $U_e$ by providing a plan for given $S$, $s'_i$, $A$, $T$, and $o$.

Hybrid Planning and a Hybrid Plan: Given a planning problem and a set of planners, informally, hybrid planning refers to determining a sequence of plans, possibly generated by different planners, that optimizes $U_e$; this sequence is known as a hybrid plan. Formally, hybrid planning is equivalent to finding a path consisting of nodes and edges in a reachability graph. An example of such a path is illustrated in Figure 3.

A Reachability Graph: A reachability graph $\Gamma$ is a directed graph defined as a tuple $(V, E, V^i)$, where $V$ is a set of nodes, $E$ is a set of edges such that $\mathcal{E} \subseteq V \times V$, and $V^i$ is a set of initial nodes of the graph such that $V^i \subseteq V$. Figure 3 shows a reachability graph

---

5 This definition of a plan is general enough to capture different types of plan. For instance, universal plans such as MDP policies could be linked directly to the state-action mapping provided by the function $P$. This definition also captures sequential plans, since the time as a state variable can be used to maintain the ordering of actions during execution. Some planners (e.g., deterministic planners) generate plans that do not have an action corresponding to every state in a state-space. In our formalism, we map such states to a special action referred to as empty action.

6 $U_e$ is referred to as a posteriori utility, i.e., utility yielded after executing a plan.
with the nodes (e.g., $N_4$), the edges (e.g., $E_1 = (N_2, N_4)$), and the initial nodes (i.e., $N_1$, $N_2$, and $N_3$).

A Node in a Reachability Graph: A node is a tuple $(Pb, Pl, Dl)$ consisting of a planning problem $(Pb)$, a compatible planner $(Pl)$ that can solve $Pb$, and deadline $(Dl)$, which is the worst-case planning time for $Pl$ to solve $Pb$. A problem-planner node (say $(Pb', Pl', Dl')$) in a graph indicates that $Pb'$ is compatible (i.e., could be solved) with $Pl'$.

An Edge in a Reachability Graph: An edge originating from a node $(Pb', Pl', Dl')$ represents the complete or a partial execution of the plan (say $P'$) determined by planner $Pl'$ for problem $Pb'$. In the context of the cloud system, assuming $P'$ has two actions (e.g., add a server then increase the dimmer), an example of a partial execution is to execute only the first action (i.e., add a server), whereas the complete execution would refer to executing both the actions in the plan. The utility of an edge is the same as the utility of the corresponding (full or partial) execution. Initial nodes $V^i$ indicate the potential starts of executions in a reachability graph. Informally, an edge between a pair of nodes $N_a$ and $N_b$ indicates that the plan for the problem-planner pair in $N_b$ can take over execution from the plan (after full/partial execution) for the problem-planner pair in $N_a$; therefore edges in a graph solve the planning coordination problem $(PlnCrd)$. An edge can be constructed between $N_a$ and $N_b$ if and only if the two reachability conditions are met:

- **Preemption:** after executing the plan from $N_a$, the system should reach the initial state of the planning problem in $N_b$. Only then the plan for $N_b$ takes over from the plan for $N_a$.  

---

**Figure 3. An example of a Reachability Graph**
• **Timing:** the plan in $N_b$ should be ready once the execution comes to it. Hence, the planner for $N_b$ has to be invoked at least the worst case planning time (i.e., deadline) for the planner to solve the problem.

Given a planning problem (say $Pb_0$) and a set of planners (say \{Pl_0, Pl_1, Pl_2,...\}), here is a process to build a reachability graph. Suppose $Pb_0$ is compatible with planners $Pl_0$ and $Pl_1$ that solve $Pb_0$ in (the worst-case) time $dl_{00}$ and $dl_{01}$ respectively. Therefore, the reachability graph (in Figure 3) has initial nodes $N_1$ and $N_2$, indicating that $Pb_0$ could be solved by $Pl_0$ and $Pl_1$.

Figure 3 shows another initial node (i.e., $N_3$) consisting of a modified problem (say $Pb_1$) of $Pb_0$, a planner (say $Pl_1$) compatible with $Pb_1$, and deadline $dl_{11}$. A planning problem could be relaxed (i.e., modified) to reduce the state space to be searched for planning; in other words, only a smaller part of the state space is considered when planning for a modified problem. Therefore, such a modification helps reduce planning time. However, solving the modified low-fidelity problem is likely to result in a sub-optimal plan, since the optimal plan might exist in the part of the state space that has not been considered while planning. To exemplify, in the case of a constraint violation for the cloud system, suppose $Pb_0$ considers all the adaptation actions for planning. In contrast, assume $Pb_1$ considers only a subset of actions (e.g., `addServer`, `increaseDimmer` and `divert_traffic`). In this example, planning with fewer actions will reduce planning time due to the reduced search space; however, the plan is likely to yield a lower utility compared to the one determined using all the actions.

Using node $N_2$ as an example, we explain how a graph is extended from an initial node. Suppose planner $Pl_0$ determines a plan $P_{00}$ for problem $Pb_0$. An edge originating from node $N_2$ would indicate a full or a partial execution of plan $P_{00}$. Since different partial executions from node $N_2$ are possible for plan $P_{00}$, there are multiple outgoing edges from $N_2$ indicating various executions such as $E_1$ and $E_2$. Suppose execution $E_1$ takes the system to the initial state of problem $Pb_2$, which has two compatible planners $Pl_2$ and $Pl_3$. To capture a combination of $Pb_2$ with both the compatible planners, the graph has nodes $N_4$ and $N_5$. However, an execution (e.g., $E_2$) could also lead a system to different planning problems. Suppose execution $E_2$ of plan $P_{00}$ takes the system to planning problem $Pb_3$. Node $N_6$ represents the combination of problem $Pb_3$ with a compatible planner $Pb_4$. Since problem $Pb_3$ could be modified, there is another reachable node $N_7$ that represents a modified problem (i.e., $Pb_3$) along with a compatible planner (i.e., $Pb_3$). Using these possibilities the graph is extended from all the nodes (including the initial nodes $N_1$ and $N_3$).

A path in a graph consists of nodes and edges that refers to a combination of plan executions for different problem-planner nodes in the path. Such a combination that optimizes a posteriori utility (i.e., $U_e$) is a hybrid plan. In other words, hybrid planning is about finding a sequence of problem-planner nodes that yields optimal $U_e$. Since a node in the optimal path informs which planner to be invoked to solve a planning problem, the path solves the planning selection problem ($Pl_NSel$).

To explain a reachability graph, for example, in the context of a cloud system, suppose there are two planners (e.g., planners based on MDP and Case-based reasoning) to solve the problem of a constraint violation. In the reachability graph, there could be different paths representing various combinations of the two planners. For example, paths using MDP planning alone, or initially using CBR planning and later switching to
MDP planning. Assuming the combination of CBR and MDP planning yields highest utility, this path would be selected, thus improving the system’s utility compared to other paths.

To deal with the complexity of constructing a reachability graph and identifying an optimal path, the formal model breaks the problem of hybrid planning into three sub-problems.

- **Path Selection** \((\text{PthSel})\): The Path Selection sub-problem is, informally, to find a path in a reachability graph that yields the highest utility. As discussed earlier, this path implicitly solves the planning selection problem \((\text{PlnSel})\).

- **Reachability Graph Construction** \((\text{GphCon})\): The Graph Construction sub-problem is to evaluate the reachability conditions between each pair of nodes in a reachability graph. If the reachability conditions are satisfied for a pair, the nodes are connected through an edge to construct the graph, eventually, to be used by \(\text{PthSel}\).

- **Planner Assessment** \((\text{PlrAst})\): The Planner Assessment sub-problem is, given a planning problem and a set of compatible planners, rate the performance of these planners on that problem. The metrics for the rating are execution utility (i.e., quality) and planning time (i.e., timeliness). This rating process is repeated for all the possible (i.e., original, intermediate, and modified) planning problems. These ratings are used by \(\text{PlrAst}\) and \(\text{PthSel}\) to construct the edges between nodes and to find an optimal path in a reachability graph, respectively.

To summarize, this formal model explains the general nature of the problem of hybrid planning; this model is general in the sense that it explains the problem for any number of planners and for planning problems with an infinite planning horizon. The formalism assumes the operating domain to be *Markovian*. Many of the domains explored by the self-adaptive community are assumed to be Markovian [37, 57, 65] (even though not always explicitly stated). Therefore, even with this assumption, the proposed formal model is applicable to various domains, particularly those investigated by the self-adaptive community. As discussed earlier, this formalism has a number of benefits. First, solutions grounded on such a formal model give us confidence that all relevant challenges are addressed. Second, this formal model sets the stage for going beyond the solution proposed in this thesis to find even better solutions to hybrid planning. Third, this model serves as a unifying evaluation framework for different such solutions; due to the complexity of the hybrid planning problem, the implementations will vary in algorithms that address the three sub-problems.

### 5. Proposed Approach

Although the theoretical model discussed in Section 4 helps to explain the general nature of the hybrid planning problem, using this model is not practical for two principal reasons:

- In theory, since a planning problem could be modified in an unlimited number of ways and have an infinite planning horizon (i.e., no explicit goal/end state), a reachability graph (even with discretized time) could potentially have an infinite
number of nodes and edges (connecting these nodes). Finding an optimal path in an infinitely large reachability graph is an intractable problem since it requires comparing utilities for an infinite number of infinitely long paths.

- The model ignores the time to solve the three sub-problems (i.e., PrnSel, GphCon, and PlrAst). Even with a finite number of problem-planner nodes, the time to construct a reachability graph is unlikely to be negligible, since the process requires solving PlrAst (i.e., rating the planner against the problem for each problem-planner node) and GphCon (i.e., evaluating the reachability conditions for each pair of a problem-planner node). Moreover, finding an optimal path (i.e., PrnSel) in the graph can take non-negligible time. Since hybrid planning aims at dealing with the run-time planning delay, an additional delay (due to solving the three sub-problems) would further increase the complexity and decrease the effectiveness of applying hybrid planning to realistic self-adaptive systems.

This section presents a practical approach that finds an approximate solution to the hybrid planning problem, particularly in the context of self-adaptive systems. For an application of hybrid planning, the first challenge is to constrain the size of a potentially infinite reachability graph. We plan to achieve this objective by making three assumptions. As discussed later in Section 5.3, in this thesis, we hypothesize that even with these assumptions hybrid planning is beneficial for realistic self-adaptive systems.

- **Finite-horizon**: A planning problem has a finite planning horizon. In other words, the problem has an explicit goal/end state. This assumption restricts the number of problem-planner nodes.

- **Reactive-deliberative-only**: Hybrid planning is instantiated using a reactive and a deliberative planning approach, where reactive planning determines a plan (using the reactive planner) in negligible time compared to deliberative planning (using the deliberative planner) since the former considers a smaller part of the operating domain state space. As explained later, this assumption reduces the number of problem-planner nodes in a graph, making the problem of hybrid planning tractable in practice.

- **Deliberative-preferred**: For a planning problem, the deliberative plan provides a higher (or at least equal) expected utility compared to the reactive plan. This implies that whenever the deliberative plan is ready for a planning problem, it is preferred over the reactive plan. This assumption ensures that there can never be a path in a reachability graph that has deliberate planning followed by reactive planning, and thereby restricts the number of paths in the graph.

In addition to constraining the size of a reachability graph, for a practical application of hybrid planning, the other challenge is to minimize the delay due to solving the three sub-problems i.e., PlrAst, GphCon and PrnSel. As explained below in Section 5.1.2,

---

7 As discussed later in Section 8.3, if time permits, we might relax some of these assumptions.

8 Although reactive planning can take non-negligible time for complex state spaces, it is assumed to be orders of magnitude faster than deliberative planning, particularly, for domains under uncertainty. We assume reactive planning time to be negligible since many of the potential reactive planning approaches (e.g., reinforcement learning, case-based reasoning, rule based adaptation, fuzzy logic) suggested for self-adaptive systems determine plans in a negligible time.
our approach does not require us to explicitly solve $P_{LR^A}$ and $G_{PHC}$. However, to solve the $P_{HR Sel}$ (i.e., to find a path in a reachability graph that yields the highest utility), we propose instance-based learning (IBL), which helps to solve a new problem based on similar problems seen in the past [48]. 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. For a reachability graph, compared to explicitly finding an optimal path in the graph, IBL is expected to approximate a solution to $P_{HR Sel}$ quickly since most of the computation to solve $P_{HR Sel}$ is done offline (explained later in Section 5.2). In the context of hybrid planning, the inductive bias (i.e., learning bias) behind application of IBL is that reachability graphs for two “closely related” planning problems should also be similar. In other words, for two similar planning problems, an effective combination of reactive and deliberative planning for one problem would also work for the other problem.

Now, we present our approach to approximate a solution to a hybrid planning problem. In the process, we also link the approach back to the formal model.

5.1. Constructing a Reachability Graph

As explained earlier, hybrid planning is about constructing a reachability graph and finding an optimal path in the graph. As already discussed, the first step towards constructing a reachability graph is to limit the possible problem-planner nodes, and then to connect a pair of nodes if the reachability conditions are satisfied between the nodes.

5.1.1 Restricting the Number of Nodes

Assumptions $\text{Finite-horizon}$ and $\text{Reactive-Deliberative-only}$ help to restrict the number of nodes in a reachability graph corresponding to a planning problem. Assumption $\text{Finite-horizon}$ is about restricting the number of nodes by assuming the planning horizon to be finite (e.g., having an explicit goal/end state). However, even with a finite horizon, theoretically a planning problem can have a reachability graph with an infinite number of problem-planner nodes since (a) there is an unlimited number of ways to modify the problem, and (b) an infinite number of planners could be used to solve these modified problems. To address this problem, we constrain the number of problem-planner nodes using assumption $\text{Reactive-Deliberative-only}$, i.e., hybrid planning is instantiated using a reactive and a deliberative planning approach. This implies that when instantiating hybrid planning for a self-adaptive system, we restrict the number of problem-planner nodes by restricting the number of planners (i.e., reactive and deliberative planner) and how a planning problem should be modified for each of these planners.

Because of assumption $\text{Reactive-Deliberative-only}$, for a planning problem (say $Pb$), three kinds of problem-planner nodes are possible in a reachability graph. The first kind of node, representing deliberative planning, consists of the deliberate planner and a modified version (say $Pb_d$) of $Pb$ that is compatible with the planner. To explain further, suppose $Pb$ has uncertainty in both – action outcomes and observations of an underlying system state. Now, if an MDP planner is used for deliberative planning, the

POMDP-based planners can handle both kinds of uncertainty; however, these planners could be slow
input problem specification to the planner needs to ignore uncertainty in observations since MDP planners can handle uncertainty only in action outcomes.

The second kind of node, representing reactive planning, consists of the reactive planner and the modified version (say \( Pb_r \)) of \( Pb \) that is compatible with the reactive planner. As an example, if a deterministic planner is used for reactive planning, then the input problem specification cannot have uncertainty since deterministic planners cannot handle uncertainty. Such a simplification of the planning problem will decrease the search space, thereby decreasing the reactive planning time compared to deliberative planning.

The third kind of node consists of \( Pb \) and a special planner known as a \textit{noAction} planner. As discussed earlier, while deliberate planning is solving a planning problem, there are two choices: (a) invoke reactive planning, or (b) wait until a deliberate plan is ready. The first choice is represented by the reactive planning node as discussed above. The second choice is represented by the node consisting of \( Pb \) and the \textit{noAction} planner; this planner always returns an empty action suggesting no action be taken for the planning problem. In a way, this third kind of node represents a different kind of reactive planning (compared to the second type of node) since \( Pb \) pairs with the \textit{noAction} planner to formulate a problem-planner pair.

5.1.2 Connecting the Nodes

Once the number of problem-planner nodes is restricted to a finite value, the next step is to connect the nodes to build a reachability graph. To this end, we need to solve \texttt{PlRAsT} and \texttt{GphCon}. Given a planning problem-planner node, a solution to \texttt{PlRAsT} would return the deadline and the partial utility function for a plan; the deadline is used by \texttt{GphCon} to evaluate reachability between nodes and the partial utility function (returning utility of a full/partial execution) is used by \texttt{PhSel} to find an optimal path. However, solving \texttt{PlRAsT} in negligible time is infeasible since, for each problem-planner node, one needs to rate the planner with respect to the planning problem. Fortunately, as discussed later, our solution approach to \texttt{GphCon} and \texttt{PhSel} helps us to solve the hybrid planning problem without solving \texttt{PlRAsT}.

As discussed in Section 4, an edge between two problem-planner nodes guarantees a smooth transition between plans for the two nodes; in other words, connections between nodes help to solve the planning coordination problem (\texttt{PlnCrD}). To connect nodes, we need to solve \texttt{GphCon}, i.e., evaluate the reachability conditions between each pair of nodes in a reachability graph; if the reachability (i.e., preemption and timing) conditions are satisfied for two nodes, they are connected through an edge. However, this computation is likely to take a non-negligible time for realistic systems.

Instead of explicitly evaluating the reachability conditions between each pair of nodes, we rely on two features of the formal model to ensure a seamless transition from a reactive plan (or an empty plan generated by the \textit{noAction} planner) to a deliberative plan. These features are: (a) planners generate a universal plan (also known as a policy), i.e., a plan \( P \) is a total function \( P : S \rightarrow A \), where a mapping from a state \( s \in S \) to an action \( a \in A \) suggests \( a \) be executed in \( s \) [26]; and (b) the operating domain is \textit{Markovian}.

Fig. 4 explains our approach to support the transition of execution from a reactive
plan to a deliberative plan. Explaining in the context of the exemplar system, suppose at time $t_0$ in state $s$, there is a response time constraint violation. To deal with this situation, both reactive planning and deliberative planning are invoked simultaneously. Suppose reactive planning is designed such that it ignores uncertainty in the external environment by assuming the future request arrival rate to remain the same as in the current state. Since reactive planning time is negligible, suppose it suggests an action $a_1$ to be executed at time $t_0$. Meanwhile, using a time-series predictor, deliberate planning takes predicted, but uncertain, values of future request arrival rate into consideration and comes up with a deliberate plan, suppose at time $t_1$. On executing the action $a_1$, due to uncertainty in the client request arrival rate, the system could reach one of the three possible outcome states (i.e., $s_1$, $s_1'$ or $s_1''$). If the predicted values for the request arrival rate (used for the deliberate planning) are correct, these states will be found in the deliberate plan, because the plan contains the state-action pair for all the reachable states from the initial state $s$. Therefore, once the deliberate plan is ready (suppose at time $t_1$), it can take over the plan execution from the reactive plan because any state in the reactive plan will be in the deliberative plan.

Moreover, due to the Markovian nature of the operating domain, optimality of the action prescribed by the deliberative plan for states such as $s_1$, $s_1'$ and $s_1''$, depends only on that state, and not on any of the previous states. This implies that once deliberate planning solves the planning problem corresponding to the state space shown in Figure 4, the resulting plan would suggest an optimal action for each state reachable from the initial state $s$.

The structure of a deliberate plan and the Markov property increase chances of a transition from a reactive plan to a deliberative plan; however, there is still a possibility that this transition might fail due to violation of either the preemption or the timing condition between two nodes. As an example of an unsatisfied preemption condition, if a prediction of the request arrival rate is incorrect, the resulting states, such as $s_1'$ and $s_1''$,
might not exist in the deliberative plan; therefore, transition from the reactive plan to the deliberative plan would be infeasible. In such a situation, if the system still needs to adapt, deliberative planning needs to be restarted; however, the decision to invoke reactive planning depends on the solution to PhSel as discussed in Section 5.2.

Continuing with the example, the timing condition is violated if a deliberate plan is not ready by the time the system observes one of the states $s_1, s'_1$ or $s''_1$. In such cases, there is no need to restart deliberate planning since, due to the “universal” nature of the plan, once the deliberative plan is ready it can take over the plan execution; meanwhile, the system continues with reactive planning.

5.2. Finding a Path in a Reachability Graph

To this point, we have discussed our approach to deal with PlRAsT and GpHCon. The last sub-problem is PhSel, which is about finding a path consisting of problem-planner nodes that would maximize the system’s utility.

Here is the high-level overview of our approach to select planners for a planning problem. When a system observes a planning problem, it (by default) invokes deliberative planning since it is supposed to provide a high-quality plan (i.e., assumption DELIBERATIVE-PREFERRED). Since deliberative planning is likely to be time-consuming, meanwhile the system must decide if reactive planning should be invoked to provide a quick response. To make this decision, we use instance-based learning (IBL) as discussed later in this section.

Now, we explain our approach (to solve PhSel) in the context of the formal model. As explained earlier, given assumptions FINITE-HORIZON and REACTIVE-DELIBERATIVE-ONLY, for a planning problem only three kinds of node are possible in a reachability graph i.e., nodes corresponding to reactive, deliberative, and noAction planning. Therefore, initially when a system observes a planning problem, one of these three nodes has to be selected as the first node of the path corresponding to the hybrid plan for the planning problem. Given assumption DELIBERATIVE-PREFERRED, if a deliberate plan is ready for a planning problem, the deliberative planning node gets precedence over the other two nodes. Due to the structure of a plan and the Markovian property, once the deliberative plan is ready no more planning is required; in other words, no further nodes need to be selected to construct a path. However, due to the time-consuming nature of deliberative planning, initially the deliberative plan is unlikely to be ready (i.e., the timing condition for reachability is violated) therefore, the deliberative planning node cannot be selected.

While deliberative planning is in process, a decision is required to decide whether to apply reactive planning (i.e., either select the reactive or the noAction planning node). Since reactive planning time is negligible; reactive planning nodes are always available (i.e., reachable) to construct a path.

To choose between reactive planning and noAction planning, one solution is to invoke reactive planning only when a constraint is violated as we did in our preliminary work [52]. This approach could be useful for self-adaptive systems that trigger adaptation when a constraint is violated [15]. To explain in the context of 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 (possibly higher) quality plan that will take over the execution once it is ready.
Invoking reactive planning only on constraint violations, however, suffers from two limitations. First, it limits the applicability of hybrid planning to self-healing systems where adaptation is triggered in response to predefined error conditions such as constraint violations: the approach is effective for self-healing, but it is not clear how it would apply to other self-* properties (e.g., self-optimization) that are not triggered conditionally. For example, maintaining the self-optimization property requires improving utility continuously [6]. 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.

To address these limitations, we propose using a data-driven instance-based learning (IBL) approach that, broadly speaking, helps to solve a new problem based on “closely related” problems seen in the past. 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 don’t have to (explicitly) identify conditions that invoke reactive planning.

The proposed IBL approach has an offline and an online phase. During the offline phase, we profile the hybrid planner against a set of planning problems similar to the ones expected at run time. The offline phase helps to know when it is effective to invoke reactive planning in combination with deliberative planning compared to invoking deliberative planning alone. When a system faces a planning problem at run time (i.e., the online phase), it invokes the deliberative planner 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 also; otherwise, the system waits until a deliberative plan is ready.

5.2.1 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. As discussed later in Section 5.2.1.2, we use a probabilistic model-checker to evaluate the hybrid planner. 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. Formally, the goal of the offline phase is to approximate function $Y : \Xi \rightarrow \{useReactive, notUseReactive\}$, where $\Xi$ is the set of all planning problems for the system. Figure 5 provides an overview of the offline phase.

The offline phase has two steps: (a) identify a finite set of sample planning problems that a system expects to observe at run time; and (b) profile the hybrid planner against these sample problems.

5.2.1.1 Identifying Sample Problems

For a self-adaptive system, a comprehensive coverage of the planning problem space (i.e., the space consisting of planning problems that the system is likely to witness)
is critical to get a good approximation of the performance model of a hybrid planner; however, identifying a finite set of representative problems is challenging due to a potentially infinite sample problem space. Since state variables defining a state space could be real-valued (e.g., request arrival rate for a cloud-based system), the state space could be infinite; therefore, the space of planning problems consisting of states from this state-space will be infinite. While building a set of sample problems from such a problem space, it is difficult to know which problems should be sampled and when the sampling process should be stopped. Unfortunately, there is no general solution to handle this sampling problem. In this section, we explain our intuitive sampling approach to build a finite set of sample problems; this approach will be refined in the actual thesis work.

Typically, the first step to deal with this issue (i.e., an infinite sample problem space) is to find an abstract representation (of the space) that divides the space on the basis of high-level concerns (e.g., quality attributes) rather than state variables. Structuring the space along high-level concerns not only helps in abstracting relevant regions (in the space), but also assists in tracking (un)explored regions.

Finding an abstraction for a state space is a two-step process: (a) find a set of independent dimensions for abstractly defining a sample space; and (b) discretize these dimensions to construct non-overlapping regions in the space. We plan to use quality attributes for abstracting a sample space. As an example, for the cloud-based system discussed in Section 3, we could use quality attributes such as response time, server cost, and revenue as the dimensions to abstract the sample space. In addition to quality attributes, researchers have proposed using external factors such as workload (i.e., request arrival rate) as a dimension for an abstraction [33] [44].

However, the final choice of dimensions depends on the system’s requirements. For instance, in the context of cloud-systems, smaller businesses that do not have many end users, usually prefer to reduce the resource cost, while larger businesses that have many end users, treat cost as a secondary concern (compared to SLA violations such as response time constraint violation). Therefore, for a cloud-based system for bigger businesses, one can ignore resource cost as a dimension (for abstraction) but a system
supporting smaller businesses cannot ignore the resource cost as a dimension.

Once the dimensions for abstraction are decided, discretization of the sample space along these dimensions could be done using one of two approaches: (a) using requirements specification, or (b) using expert knowledge. As an example, a system requirement may explicitly provide a correlation between the range of a quality attribute and its impact on the system’s utility [14]; such a requirement specification could be used to discretize the quality attributes on the basis of range.

At this point, we have divided the abstract sample space into non-overlapping regions. The next step is to construct a set of sample problems representing each region in the space ensuring a comprehensive coverage of the sample space. However, while constructing the set, a key challenge is to optimize the size of the set; a large set could slow down the process of finding similar planning problems during the online phase. To have a good coverage of different regions while keeping the set size small, we plan to investigate ideas from the domain of software testing [61] and instance-based learning [67].

5.2.1.2 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 \(P_b_s\)) using the deliberative planner; suppose a plan \(P_d\) is determined in time \(t_d\). Second, we solve the same problem with the reactive planner, which determines a plan \(P_r\).

For each sample problem, there are two evaluations corresponding to the two possible choices (i.e., to use or not use reactive planning) for the hybrid planner. To evaluate these choices, we plan to use a probabilistic model-checker – in this case PRISM\(^{10}\) [55]. The model-checker takes a planning problem specification and a plan as an input, and returns the utility expected on executing the plan for the problem. 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 \(P_r\) is executed, and (b) from time \(t_d\) onwards, deliberative plan \(P_d\) is executed. Suppose on evaluating the combination of the reactive and the deliberative plan, the model-checker returns the expected utility as \(U_c\). This evaluation represents the case when, initially, reactive planning is invoked to solve \(P_b_s\) until the deliberative plan (i.e., \(P_d\)) is ready to take over the execution.

The second evaluation represents the case when a system does not invoke reactive planning to solve \(P_b_s\), 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 \(P_d\).

Finally, we compare the expected utilities \(U_c\) and \(U_d\) to decide if reactive planning should be invoked (or not) to solve \(\xi_s\); if \(U_c > U_d\) then invoke the reacting planning.

\(^{10}\)A probabilistic model-checker such as PRISM helps to capture uncertainty in the operating domain.
(i.e., $Y(\xi_i) = \text{useReactive}$) otherwise wait for the deliberative plan to be ready (i.e., $Y(\xi_i) = \text{notUseReactive}$).

### 5.2.2 The Online Phase

When a system senses an adaptation opportunity (i.e., a planning problem, say $Pb_r$) at run time, it needs to decide if invoking reactive planning will improve utility. In the context of instance-based learning, this decision consists of solving two sub-problems: (a) find planning problem(s) from the profiling stage that is similar to $Pb_r$, and (b) decide to invoke (or not) reactive planning based on its performance on the matching problem(s). Figure 6 provides an overview of the online phase.

![Figure 6. Overview of the online phase](image)

It will typically be the case that the system will not encounter precisely the problems that were profiled in the offline phase. Therefore, we need a way to determine if a particular state of the system is similar to a state we observed in the offline phase. Formally, we need to define a function $S : \Xi \times \Xi \to \mathbb{R}$ that takes two planning problems as an input and returns a real number indicating the similarity between the planning problems. To find similarity between two planning problems, one can develop a similarity metric that considers various aspects (e.g., the similarity between current states of the system) to calculate the similarity. We can then use this metric to find instances from training 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 [48]. Figure 6 provides an overview of the online phase.

### 5.3. Discussion

The approach outlined above solves the hybrid planning by making three assumptions: (a) a planning problem has a finite planning horizon, (b) hybrid planning is instantiated using a reactive and a deliberative planning approach, and (c) for a planning problem, the deliberative plan provides a higher (or at least equal) expected utility compared to the reactive plan. An important question is whether these assumptions are too restrictive to
support realistic self-adaptive systems. In this section, we argue that these assumptions are likely to be realistic and hybrid planning will be useful even with these assumptions.

- **Finite-horizon**: This assumption restricts the planning horizon to a finite value. For realistic self-adaptive systems such as cloud-based systems, planning for an infinite horizon is not (typically) recommended since as the planning horizon increases, the planning time increases exponentially, while the quality of planning decreases due to decrease in accuracy of predictions (e.g., request arrival rate). Researchers from the self-adaptive community have demonstrated that planning even with a finite horizon is effective [22, 49, 64].

- **Reactive-deliberative-only**: To instantiate hybrid planning, we restrict the number of constituent planning approaches to two i.e., reactive and deliberative planning. Even with this restriction, using specific instantiations of reactive and deliberative planning in different domains, researchers (including us) from the self-adaptive community have demonstrated the potential of hybrid planning [25, 52, 65, 66]. However, the existing work is limited to invoking reactive planning only on faults (i.e., for self-healing [6]). In contrast, as discussed earlier, in this thesis we will extend the idea of hybrid planning to support other self-* properties such as self-optimization. Moreover, we will consider different kinds of instantiation (different reactive-deliberative) combinations, thereby broadening the effectiveness/generality of a 2-planner approach.

- **Deliberative-preferred**: We assume that, for a planning problem, deliberative planning provides better plans compared to reactive planning. This is a realistic assumption since, as discussed earlier, reactive planning ignores parts of the operating domain state space; this reduced state space is likely to result in lower quality plans compared to ones determined by deliberative planning.

6. Preliminary Work

6.1. Hybrid Planning for Self-healing

We introduced the idea of hybrid planning in a paper published at SASO-2016 [52]. In this work, we combined a reactive, deterministic planning approach with a deliberative, MDP planning approach. Reactive planning was invoked in case of faults, specifically, a constraint violation. In other words, reactive planning was used for self-healing i.e., to restore system from a fault [6]. As an initial feasibility study for the hybrid planning approach, we conducted experiments in a simulated cloud-based self-adaptive setting (outlined in Section 3) using a realistic workload pattern from the WorldCup 98 website [3]. In that context, the experiments demonstrated that the combination of deterministic and MDP planning performs better than using MDP planning alone.

Further analysis of the experimental data revealed that, in addition to invocation of reactive planning for self-healing (e.g., constraint violations), reactive planning could be useful for self-optimizing systems in which a system continuously tries to improve utility (rather than just responding to faults). Therefore, in this thesis our approach has been extended to the application of hybrid planning to support self-* properties such as self-optimization. However, this requires a more general approach to invoke reactive
planning.

6.2. Formalization of the Hybrid Planning Problem

For a broad application of hybrid planning and to lay foundations for a variety of possible approaches to hybrid planning, one needs to understand the general nature of the problem. To this purpose, we developed a formal framework to define the problem of hybrid planning; this work was published at SEAMS-2017 [53]. As discussed in Section 4, this formal model helps to explain the problem in its general form and breaks it down to the three sub-problems.

6.3. Hybrid Planning for Self-optimization

Although the formal framework helps to explain the hybrid planning problem, as discussed in Section 5, there are practical challenges to apply hybrid planning using this theoretical model. Therefore, we propose an approach that will help to approximate a solution to a hybrid problem even for self-optimizing systems. The key part of the proposed approach is to use instance-based learning (IBL) to decide whether reactive planning should be invoked to solve a planning problem.

As a preliminary validation of IBL, we performed experiments for a simulated cloud-based application. 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. Specifically, 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. The experiments indicated the potential of IBL to decide whether reactive planning should be invoked (or not) for a planning problem; this work was published at SASO-DSS-2017 [54].

7. Related Work

Different research communities, such as artificial intelligence (AI) and self-adaptive software systems, have made attempts to deal with the trade-off between quality and timeliness of planning; in particular, the AI community has proposed a wide range of solutions to address this problem. This section provides an overview of existing approaches that aim at finding a balance between quality and timeliness of decision-making.

7.1. Automated Planning

The AI community has been working towards finding better algorithms and heuristics to deal with the issue of planning delay. Generally speaking, these approaches reduce the planning time by selectively exploring the decision search space, thereby trading off quality against timeliness. For instance, many algorithms/heuristics have been developed to reduce the planning time for deterministic domains [8, 9, 24, 32]. For non-deterministic and probabilistic domains, the state-of-the-art planning approaches based on Markov decision processes (MDP) [47], and partially observable Markov decision processes (POMDP) [35], are comparatively slow concerning the planning
Various algorithms have been suggested to improve the planning time for MDP [47] and POMDP [56, 62] planning; however, planning delay in non-deterministic and probabilistic domains is still an ongoing challenge.

Researchers have also suggested a broad category of planning algorithms, based on the idea of incremental planning, known as “anytime” planning: the planning process can be interrupted at any time to get a sub-optimal plan; however, more planning time leads to better plans [71]. This approach is a special case of hybrid planning since anytime algorithms utilize the execution time of a low-quality plan to devise an improved plan; once ready, the improved plan takes over the execution from the lower-quality plan. However, compared to anytime planning, the idea of hybrid planning is broader since it allows us to combine plans from different planning approaches.

The AI community has also explored hybrid planning directly. Hayes-Roth combined multiple planners to complete different sections of an abstract plan skeleton. The combined planners operate in a hierarchical fashion to solve different sub-problems [30], which is unlike the hybrid planning approach. Similar to Hayes-Roth, Dean proposed decomposing a planning problem into independent sub-problems, which are solved by different planners [17]. Beetz et al. [4] proposed an approach that projects the effects of contingencies on the plan (generated by the primary planner) under execution and, if required, revises the plan using a more deliberative planner. This is close to the proposed approach; however, unlike our approach, they invoke reactive planning by default and later improve the plan using deliberative planning; moreover, their solution is restricted to a particular plan specification language. Mausam et al. instantiated a hybrid planner specifically using GPT [10] and MBP [7] planners that act as a deliberative and a reactive planner respectively [46]. In contrast, our work adopts a more general approach by formalizing the hybrid planning problem and proposing a solution approach that is flexible – not restricted to a specific combination of reactive and deliberative planner.

7.2. Frameworks to Support Automated Planning

From the self-adaptive systems community, Kramer et al. [40] proposed a layered architecture inspired by Gat [21], which deals with the problem of planning delay through hierarchical decomposition of the planning domains. Tajali et al. extended the layered architecture by suggesting two types of planning: application planning and adaptation planning [66]. Since hierarchical decomposition of a planning domain reduces the planning state-space, the layered architecture helps to reduce the planning time. In a sense, the layered architecture is a kind of hybrid planning since the layers represent varying levels of deliberation that helps to balance quality and timeliness of plans. However, unlike the layered architecture, the proposed approach is not limited to a hierarchical decomposition of the planning domains. In fact, the proposed idea is more generic than the layered architecture since hybrid planning could be deployed within a layer to deal with the planning delay in that layer.

To find a balance between quality and timeliness of planning, the AI community has proposed various execution frameworks that, in general, also rely on the hierarchical decomposition of the planning domains [39]. Quite different from these layered architectures, Musliner et al. proposed a framework that ensures the execution of tasks in a specified deadline [51]. However, unlike hybrid planning, this framework requires hard deadlines to be specified in the planning specification.
7.3. Hyper-Heuristics

The research field of hyper-heuristics focuses on developing search methods or learning mechanisms for selecting or generating heuristics to solve computational search problems [11]. 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. The field of hyper-heuristics is motivated primarily by two foundational frameworks. The first (mathematical) framework, formulated by Wolpert, suggests that it is impossible to devise a silver-bullet algorithm since all optimization algorithms yield equivalent performance on average [70]. The second framework, suggested by Rice, helps in selecting an appropriate algorithm for a given problem [58].

Since planning, in general, fits into the category of search/optimization problems, researchers have applied ideas from the field of hyper-heuristics to select a planner (from a set of planners) to solve a planning problem. For instance, Gratch et al. [27, 28] proposed a system that uses hill-climbing search in the space of possible control strategies (i.e., planners) to solve a scheduling (i.e., planning) problem. However, their work was based on assumption that control strategies can be structured to facilitate a specific search method. Moreover, unlike Gratch et al., our proposed approach is more general since it is not limited to any specific domain.

7.4. Planning in Self-Adaptive Systems

Existing work by the self-adaptive systems community either ignores the timing concerns or focuses on domains where timing is not a first-class concern [45, 59]. As a consequence, the prior approaches broadly fall into either the category of reactive or deliberative planning.

The self-adaptive community has proposed several approaches that can be considered as reactive planning approaches. These approaches determine adaptation plans in a negligible time since plans are not generated at run time, but rather selected from an existing set of plans. Some of these approaches are – genetic algorithms [16], reinforcement learning [2, 37, 57], fuzzy-logic [33], case-based reasoning [63] and using predefined condition-action rules [14]. In addition, Zoghi et al. [72] suggested a variation of hill-climbing search, which generates an adaptation plan at run time but in a small search space since adaptation actions are already ranked (offline) for a given adaptation scenario.

In contrast, the community has also explored more deliberative planning approaches that dynamically generate plans. Such approaches are likely to provide higher quality plans since factors such as uncertainty in the environment, the current state of a system, and tactic latency are considered when generating a plan; however, since plans are generated at run time, the planning time is higher compared to reactive planning approaches. Examples of such approaches are – MDP-based planning [49], stochastic multi-player games (SMG) based planning [13], and model-checking [65]. These approaches are typically applied to domains where the planning time is not a concern or a plan search space is small.

Compared to the existing work for self-adaptive systems, the proposed hybrid planning approach explicitly focuses on the trade-off between quality and timeliness.
of planning. A key advantage of hybrid planning is the use of existing reactive and deliberative planning approaches that can be combined to find a balance between quality and timeliness of planning. Even though Ghahremani et al. [25] propose combining a reactive and a deliberative planning approach to find this balance, their approach invokes reactive planning only based on predefined conditions.

8. Research Plan

This section discusses the research plan, risks and planned timeline for this thesis.

8.1. Validation

Referring to the claims discussed in Section 2, validation needs to demonstrate that hybrid planning approach is:

- *effective* in terms of improving utility of self-adaptive systems;
- *general* enough to be applied to self-adaptive systems operating in different domains;
- *flexible* to combine different instantiations of reactive and deliberative planning.

To this end, we plan to conduct two case-studies from different domains. Case-studies from different domains will help to validate the claim about generality of the approach. Moreover, for the two case-studies, we plan to combine a different set of reactive and deliberative planning approaches to support the claim for flexibility of hybrid planning. For instance, in one case-study, we might combine deterministic (i.e., reactive) and MDP planning (i.e., deliberative), whereas in the other we might combine a reactive, rule-based adaptation approach with MDP planning. To validate effectiveness, in both the case-studies, we will demonstrate that hybrid planning improves utility of a self-adaptive system compared to using either alone.

As the first case-study, we will extend the simulation setup for the cloud-based system (discussed in Section 3) used in our previous work [52]. For this case-study, we plan to use realistic workload traces, both in offline and online phase, to demonstrate effectiveness of the proposed approach. However, if realistic traces are not accessible, we might use synthetic traces imitating realistic workload patterns as has been used by others for similar validation studies [20].

We have not decided on the second case-study yet. However, we will ensure that this case-study is from a different domain, such as cyber-physical systems. Moreover, the adaptation goals and utility function reflecting these goals will be different from the first case-study. Unmanned aerial aircraft is one potential case-study that we might investigate [31] (UAVs have been explored by other researchers (in the self-adaptive community) to validate their ideas [23, 50]).

8.2. Risks

In the context of self-adaptive systems, even though our preliminary work demonstrates the potential of hybrid planning and instance-based learning approach, there are some
risks, primarily, linked to the practicality of IBL. This section lists the major risks and proposes strategies to mitigate these risks.

- **Identifying sample problems:** For a good approximation of the performance model of a hybrid planner, it needs to be evaluated against a set of planning problems that comprehensively covers the planning problem space. However, as already discussed in Section 5.2.1.1, building a finite set of sample problems that (reasonably) cover an infinite sample problem space is difficult. In Section 5.2.1.1, we already explained an intuitive approach to identify sample problems by abstracting the state space based on quality attributes and other high-level (relevant) features of a domain.

- **High computation cost:** The key risk applying IBL in a realistic context is that the computation time to find the similar problem(s) can be high since nearly all the computation is delayed till the online phase [1]. This could be problematic since an additional run-time delay would negatively impact the effectiveness and practicality of hybrid planning. Here are the possible strategies to reduce the computation cost when finding similar problems:

  - **Optimize the set of sample problem:** We need to optimize the size of a sample set such that such that a space of planning problems (expected at run time) is represented using a minimum number of problems. A large set of planning problems could delay the similarity matching process; the time complexity would be \(O(n)\), where \(n\) is the number of sample problems. To this end, as already discussed in Section 5.2.1.1, we plan to explore instance (i.e., sample problems) reduction techniques developed by the machine learning community [67].

  - **Efficient memory indexing of sample problems:** To quicken the process of identifying the similar problem(s) at run time, an efficient indexing of sample problems is needed. To this purpose, we plan to investigate indexing techniques used by machine learning practitioners. For instance, when using k-Nearest Neighbor algorithm (for IBL) [48], \(kd\)-tree [5] is a widely used data structure for efficiently indexing the sample instances; the average time complexity for searching in a \(kd\)-tree is \(O(lg \ n)\).

- **Defining a similarity metric:** For a similarity metric that determines similar planning problems with a high precision, we need to define the metric using relevant attributes. To find relevant attributes and their relative weights in a similarity metric, we plan to explore cross-validation methods suggested by Moore and Lee [41].

- **Use of model-checking to profile sample problems:** In this thesis, we propose using a probabilistic model-checker to evaluate a hybrid planner against a set of sample problems; a probabilistic model-checker such as PRISM helps to capture uncertainty in the operating domain. As a proof-of-concept, our preliminary investigation in a limited context indicated the potential of model-checking [54]. If model-checking fails to provide correct results in a more general setup, then we plan to evaluate a hybrid planner by simulating hybrid plans on the actual system.
In addition to indicating the potential of model-checking, our initial experiments hinted that simulating hybrid plans on the actual system could be another approach to evaluate a hybrid planner.

8.3. Timeline

Based on the expected contributions discussed in Section 2, this section lists the main task categories that will make up this thesis work, along with the current status of tasks and rough estimates of the time it will take to complete them (refer Table 1).

- **Formalization of the hybrid planning problem**: We have already developed a formal framework explaining the problem of hybrid planning.

- **Hybrid planning in the context of self-healing systems**: As already discussed, we evaluated hybrid planning for a cloud-based system where reactive planning was invoked to handle faults such as constraint violations. Our experiments provided evidence for the potential of hybrid planning.

- **Implementation of a profiling infrastructure**: To support hybrid planning for self-optimizing systems, we proposed IBL approach that requires profiling a hybrid planner against a set of sample problems. We intend to automate the process by building a profiling infrastructure. This infrastructure will be used in both the case-studies.

- **Execution of the two case-studies**: To evaluate hybrid planning for self-optimizing systems, we will perform two case-studies as discussed earlier. In the first case-study, we will extend the cloud-based system used in our earlier work. The second case-study is yet to be decided.

- **Implement hybrid planning in the context of Rainbow framework**: We will implement hybrid planning in Rainbow framework. This task will help us realize engineering challenges when implementing hybrid planning in existing frameworks.

- **Develop a taxonomy and guidelines for planner selection**: The AI community has developed taxonomies for planning approaches and guidelines to choose among planning approaches depending on characteristics of an operating domain [26]. We will extend the existing work, specifically, in the context of self-adaptive systems.

In addition to these tasks, if time permits, I would like to extend the thesis work in the following directions:

- **Relax assumption Reactive-Deliberative-only**: We might instantiate hybrid planning using more than two planning approaches. For instance, use two reactive and one deliberative planning approach to instantiate hybrid planning.

- **Different combinations of reactive-deliberative planning**: Besides the two instantiations of hybrid planning corresponding to the two case-studies, we could investigate other possible instantiations of hybrid planning that can be useful in the context of self-adaptive systems.
<table>
<thead>
<tr>
<th>Task</th>
<th>Status</th>
<th>Est. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formalization of the hybrid planning problem</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>Validation in the context of self-healing systems</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>Implementation of a profiling infrastructure</td>
<td>Pending</td>
<td>1-2 months</td>
</tr>
<tr>
<td>Execution of two case-studies</td>
<td>Pending</td>
<td>7-12 months</td>
</tr>
<tr>
<td>Implement hybrid planning in the context of Rainbow framework</td>
<td>Pending</td>
<td>1-2 months</td>
</tr>
<tr>
<td>Develop a taxonomy and guidelines for planner selection</td>
<td>Pending</td>
<td>1-2 months</td>
</tr>
</tbody>
</table>

**Table 1. Research Plan**

- **Support for active learning**: Currently, as discussed earlier, the offline phase is used to build the performance model of a hybrid planner by profiling it against a set of planning problems that a system expects to observe at run time. However, over a period of time, a system might face new planning problems that are not represented in the set of sample problems. This might happen due to various reasons such as a change in work load patterns in the context of a cloud-based system. Therefore, the system needs to identify such new planning problems to actively improve the performance model of the hybrid planner. To this end, we might apply a machine learning technique known as active learning in which a learning algorithm is able to interactively query the user (or some other information source) to obtain the desired output for a new data point (e.g., a planning problem) [60]. In the context of hybrid planning, we will use model-checking as a source of information to decide whether reactive planning needs to be invoked for a new problem. However, a key challenge is to decide which problem should be considered as a new one.

- **Apply hybrid planning in the context of human-in-the-loop**: Further ahead, we might apply hybrid planning in the context of human-in-the-loop by treating human (i.e., a system administrator) as a reactive planner that is used in combination with some deliberative planner such as an MDP planner. Self-adaptive systems such as smart-grid often require intervention by system administrators, particularly, for emergency situations; an administrator could provide a quick decision based on his past troubleshooting experience. To apply hybrid planning in such a context, an interesting challenge will be to build a model of human decision-making that will be used to decide when it is safe for a human to make a decision. To this purpose, one might explore the framework proposed by Eskins et al. [19] to model human behavior.
References


