Lifelong Dynamic Optimization for Self-Adaptive Systems: Fact or Fiction?

Tao Chen
Department of Computer Science
Loughborough University, Leicestershire, UK
t.t.chen@lboro.ac.uk

Abstract—When faced with changing environment, highly-configurable software systems need to dynamically search for promising adaptation plan that keeps the best possible performance, e.g., higher throughput or smaller latency — a typical planning problem for self-adaptive systems (SASs). However, given the rugged and complex search landscape with multiple local optima, such a SAS planning is challenging especially in dynamic environments. In this paper, we propose LiDOS, a lifelong dynamic optimization framework for SAS planning. What makes LiDOS unique is that to handle the “dynamic”, we formulate the SAS planning as a multi-modal optimization problem, aiming to preserve the useful information for better dealing with the local optima issue under dynamic environment changes. This differs from existing planners in that the “dynamic” is not explicitly handled during the search process in planning. As such, the search and planning in LiDOS run continuously over the lifetime of SAS, terminating only when it is taken offline or the search space has been covered under an environment.

Experimental results on three real-world SASs show that the concept of explicitly handling dynamic as part of the search in the SAS planning is effective, as LiDOS outperforms its stationary counterpart overall with up to 10× improvement. It also achieves better results in general over state-of-the-art planners and with 1.4× to 10× speedup on generating promising adaptation plans.

Index Terms—Self-adaptive systems, search-based software engineering, multi-objectivization, configuration tuning

I. INTRODUCTION

Many software systems are highly-configurable, thereby making them flexible to different needs. When operating under dynamic and uncertain environment, those systems are capable of changing their own configuration at runtime with an aim to achieve the best of their performance objective, e.g., higher throughput or smaller latency [1], [2] — a typical type of self-adaptive systems (SASs) that we consider in this work. For example, APACHE STORM, a stream processing framework, can change some adaptation options (e.g., num_counters and numSplitters) at runtime to react to the changing dynamic and uncertain environment, leading to a highly complex search landscape. For example, we see a rather rugged landscape from Figure 1 on the case of STORM under the WORDCOUNTER environment, where there are many local optima — the sub-optimal adaptation plans for which all the neighboring plans have worse performance — that can easily trap the planner. As a result, the produced plans may be far away from the global optimum, especially considering that the planner needs to produce a plan efficiently with little overhead. As we will show in Section III-A, the landscape can differ following dynamic environmental changes, which further complicates the SAS planning.

In this paper, we tackle the above challenge by proposing LiDOS, a framework that implements the concept of lifelong and truly dynamic optimization for SAS planning. By designing a mechanism that takes the characteristics of SAS planning into account, the “dynamic” we consider refers to the case that the changes are explicitly captured as the search in the planner proceeds, which is the exact definition of dynamic optimization coined by Nguyen et al. [26], hence the useful information
for planning after the change can be preserved to better overcome local optima. As such, the search and optimization in LiDOS runs continuously, terminating only when the SAS is taken offline or all search space has been covered under an environment — the true lifelong optimization. In this way, we hope to make the lifelong dynamic optimization more factual for SASs. This differs from previous SBSE work for SAS planning where dynamic optimization is more “fictional”; such that the planner is pseudo-dynamic because, although it runs continuously, the true “dynamic” is not explicitly considered in the search, but a vanilla search algorithm is directly adopted and hoping that the changes will eventually be coped with via the re-evaluated fitness [12], [13], [14], [15]; it can also be distinguished from the stationary planners, in which the planning is restarted from scratch upon an environmental change or based on a fixed frequency [9], [10], [11]. Both the pseudo-dynamic and stationary planners are not ideal given the rugged landscape and the need of timely planning for SAS.

The key novelty of this work is that we formulate the lifelong dynamic optimization for SAS as a multi-modal optimization problem. This was inspired by our observation that, for highly-configurable systems, the global optimum of performance in an environment is likely to be (or very close to) a local optimum in the other. Our aim is to continuously preserve as many local optima as possible without losing the tendency towards achieving the global optimum, hence the planning can better deal with the complex landscape even under environmental changes. The core of LiDOS for tackling the multi-modal optimization problem is a refined MMO — the meta multi-objectivization model proposed by Chen and Li at FSE’21 [27], because (1) it is simple; (2) it fits the needs of a multi-modal optimization problem; and (3) it has been shown to be promising in mitigating local optima traps for highly-configurable systems. In a nutshell, our contributions are:

1) We formulate the lifelong dynamic optimization for SAS planning as a multi-modal optimization problem motivated by characteristics of the highly-configurable systems.
2) To solve the multi-modal optimization problem, we refine the MMO with a new auxiliary objective that can be easily measured, and no normalization is required, thereby eliminating the need of tuning the weight, which has been shown as a highly sensitive parameter in MMO [27].
3) We design an architecture that uses a hierarchical feedback loop to realize the concept of lifelong dynamic optimization for SASs, where the search in planning runs continuously and the dynamic changes that occur during the run are explicitly handled.
4) Through experiments on three real-world SASs of different performance objectives, scales, complexity, and under six environments, we evaluate LiDOS against its stationary variant (for ablation analysis), as well as state-of-the-art pseudo-dynamic [12], [15] and stationary planners [9].

The results are encouraging: we observe that when comparing with its stationary variant, LiDOS produces considerably better adaptation plans in general (with $p < 0.05$ and high effect size) and up to $10 \times$ improvement. LiDOS also performs significantly better overall than the state-of-the-art planners with greater efficiency: it shows $1.4 \times$ to $5 \times$ speedup of planning over the pseudo-dynamic planners and $2.5 \times$ to $10 \times$ speedup on the stationary one. This proves that lifelong (truly) dynamic optimization for SAS planning is more of a fact than fiction.

To promote open science, the code and data in this work can be accessed at: [https://doi.org/10.5281/zenodo.5386103](https://doi.org/10.5281/zenodo.5386103)

In what follows, Section II introduces the background. Section III elaborates the motivation and design of our LiDOS framework. Section IV presents our experiment methodology, followed by a discussion of the results in Section V. The threats to validity are discussed in Section VI. Sections VII and VIII analyze the related work and conclude the paper, respectively.

II. PRELIMINARIES

Here, we describe the necessary background and context.

A. SELF-ADAPTATION FOR CONFIGURABLE SYSTEMS

While the purpose of self-adaptation can vary, in this work, we focus on improving the performance of highly-configurable systems such as smaller latency and higher throughput. We model this type of SAS with $n$ adaptation options such that the $i$th option is denoted as $x_i$, which can be either a binary or integer variable. The search space of all adaptation plans, $\mathcal{X}$, is the Cartesian product of the possible values for all the $x_i$. Formally, the ultimate goal is to achieve the following for every monitoring timestep despite the environment changes:

\[
\text{argmin } f(x), \ x \in \mathcal{X}
\]

where $x = (x_1, x_2, ..., x_n)$. However, recall the landscape shown in Figure 1 the planner may be easily trapped at some of the undesired local optima, making the problem non-trivial to address. The need for planning with less overhead (due to large search space and/or expensive measurements) makes it even more challenging. An additional complication with SAS is that, as the environment change, the search landscape can also change, which we will discuss in Section III-A.

As a concrete example in Figure 2 STORM can handle data streaming process under various environments on the incoming

\[\text{without loss of generality, we assume minimizing the performance.}\]
batch of jobs, e.g., RollingCount and WordCount. We see that (1) although the default plan leads to different results across the environments, it performs fairly poor overall — compared with the global optimum, it is 6.4× worse on RollingCount and 11.5× worse on WordCount; (2) the global optimum can differ for different environments. All above motivate the need of self-adaptation for Storm, where the aim is to achieve the best possible throughput by searching the right adaptation plan, e.g., settings for num_counters and numSplitters, over changing environments.

B. Dynamic Optimization

Given the increasingly growing field of SBSE, search algorithms have been widely applied to various software engineering tasks [28]. Indeed, any search algorithm is dynamic in nature, i.e., they dynamically determine what direction to explore or which solutions to keep depending on the fitness during the search. However, the active field of dynamic optimization means a different concept. According to Nguyen et al. [26], dynamic optimization refers to the case where the “algorithm needs to take into account changes during the optimization process as time goes by.” The changes here refer to the change in search landscape, including fitness function, global/local optimum and the optimization objectives, etc.

Despite there being a good match between the definition of dynamic optimization and the requirement for SASs, truly dynamic optimization has rarely been explored. Often, the planner with a search algorithm may either be stationary (restart the search whenever upon an environmental change or upon a fixed frequency) [9], [10], [11]; or pseudo-dynamic [12], [13], [14], i.e., there is no mechanism to handle the potential impact on the search landscape when environmental change occurs, but hoping the nature of the search algorithm can eventually adapt to it using the re-evaluated fitness. Both of the two ways can waste valuable information that would benefit the search after the change, especially considering the highly rugged landscape for SASs from Figure 1. However, the difficulties of truly dynamic optimization for SASs are:

1) What information from the planning before the environmental change is useful for planning thereafter?
2) How to identify and preserve such information?

This paper tackles the above difficulties in the next section.

III. DESIGNING LiDOS

In this section, we present the motivations and technical designs behind LiDOS.

A. Key Characteristics of SAS Changes

A clear characteristic for SAS planning from Figure 1 is:

- **Characteristic 1**: Highly-configurable systems often come with a multi-modal landscape, i.e., there can be many local optima, making the planner to be easily trapped at them. This is consistent with what has been reported from prior studies on highly-configurable systems [3], [4].

By investigating the performance under different environments of the SAS studied in this work using fitness landscape analysis [29], [30], we additionally observed some common patterns and found that (Figure 3 is an example of Storm):

- **Characteristic 2**: The search landscape does change across different environments, including the global optimum, but the overall multi-modal property is still preserved and there are shared local optima. This matches with the insight by Jamshidi et al. [25], which states that the performance values between diverse environments, although different, share similar distribution (an indicator of similar ruggedness/peakedness [29]), hence the model of the landscape can be linearly transferred.

- **Characteristic 3**: The global optimum in one environment is likely to become a local optimum (which is close to the new global optimum) under another environment. This has also been hinted by evidence from the literature [25].

The above suggests that the change of environments for SAS can indeed impact the search landscape, but there is certain information on the multi-modality that can be shared before and after the environmental change. This means that truly dynamic optimization that improves the plan continuously is vital for SAS, yet we need a specific mechanism to handle the dynamic changes as the search proceeds and achieve two goals:

- **Goal 1**: preserving as many local optima as possible, since they can provide useful information upon environmental change (Characteristic 2 and 3).
- **Goal 2**: but doing so without losing the tendency towards finding the global optimum (Characteristic 1).

B. Multi-objectivization with Multi-modality

**Goal 1** motivates us to formulate the SAS planning as a multi-modal optimization problem [21], for which there exist quite a few search algorithms. However, those algorithms prioritize exploring more local optima while may negatively affecting **Goal 2**, hence they are ill-suited to our problem. As a result, in LiDOS, we refine the meta multi-objectivization (MMO) model proposed by Chen and Li at FSE’21 [27] for achieving **Goal 1** and **Goal 2** simultaneously.

In a nutshell, MMO is a recently proposed way of optimization for configurable and self-adaptive systems. The idea is to...
“multi-objectivize” the problem using an additional auxiliary objective. Unlike existing work that focuses on the algorithm level, MMO works at the more generic level of optimization model which is compatible with different multi-objective search algorithms, e.g., NSGA-II. Formally, MMO is defined as

$$\begin{align*}
\text{minimize} & \quad \left\{ g_1(x) = f_1(x) + w \times f_a(x) \\
& \quad g_2(x) = f_2(x) - w \times f_a(x) \right\}
\end{align*}$$

whereby $g_1(x)$ and $g_2(x)$ are the newly transformed objectives to be optimized; $f_1(x)$ is the target performance objective that is of concerned, e.g., throughput; $f_a(x)$ is the auxiliary objective of an additional performance attribute such that, under the given context, no stakeholders care about its value, e.g., CPU load. $w$ is a weight parameter that balances the contributions. While MMO was not originally designed for multi-modal problem but for more efficiently mitigating the issues of local optima, the uniqueness lies in that it simultaneously possesses the following properties in the transformed space:

1) Because of the Pareto dominance relation on $g_1(x)$ and $g_2(x)$, MMO preserves adaptation plans that with very different values of $f_a(x)$ (despite being similar on $f_t(x)$) as they tend to be incomparable in the sense of Pareto dominance. This encourages the search to explore a wider range of the search space and hence more likely to find and preserve different local optima, i.e., fitting the multi-modal property in nature (thereby satisfying Goal 1).

2) The global optimum of the original $f_t(x)$ is still Pareto-optimal in the transformed space. In particular, if adaptation plan $x_1$ has a better target performance objective than $x_2$ (i.e., $f_t(x_1) < f_t(x_2)$), then whatever their auxiliary objective values are, $x_2$ will not be better than $x_1$ on both $g_1$ and $g_2$: in the best case for $x_2$, they are nondominated to each other (hence suitable for Goal 2). Note that naively optimize the raw $f_t(x)$ and $f_a(x)$ (instead of $g_1(x)$ and $g_2(x)$) is harmful to Goal 2, as the search would be forced to optimize $f_a(x)$, which is of no interest to us.

In this regard, MMO is a perfect fit for our problem of lifelong dynamic optimization for SAS planning. We refer interested readers to the work of Chen and Li for more detailed examples and proofs of the above.

However, we cannot directly apply MMO to our problem due primarily to two reasons:

1) The $f_a(x)$ in MMO was designed to be an additional performance objective. This is, however, not ideal for SAS planning, as it may introduce extra measurement overhead that can be harmful for runtime adaptation.

2) More importantly, since $f_t(x)$ and $f_a(x)$ often come with rather different scales for SASs, normalization is required. However, $f_a(x)$ cannot change without the context of the transformed space ($g_1(x)$ and $g_2(x)$) instead of the original space (line 22). Since it is impractical to influence the managed system during planning, the measurements of

---

**Algorithm 1: Multi-objectivization Planner for SASs**

```
Input: Adaptation plan space $\mathcal{V}$; the Cyber-Twin of managed system $\mathcal{F}$; adaptation interval $k$
Declare: $s_{best}$ the best adaptation plan on $f_t(x)$
1. Randomly initialize a population of $n$ adaptation plan $\mathcal{P}$
2. Measure($\mathcal{P}, \mathcal{F}$)
3. $t = t + n$
4. while The search space has not been covered in the current environment and the system still runs do
   5. $\mathcal{P}' = \emptyset$
   6. while $|\mathcal{P}'| < n$ do
      7. $\{a_x, b_y\} \leftarrow$ MATING($\mathcal{P}$)
      8. $\{a_x, b_y\} \leftarrow$ DO CROSSOVER AND MUTATION($\mathcal{V}, a_x, b_y$)
      9. /* if in the same environment, only newly explored plans need to be measured; re-emerged ones can reuse the previously measured objectives. */
      10. if $a_x$ is new in current environment then
         11. Measure($a_x, \mathcal{F}$)
         12. $t = t + 1$
      13. if $b_y$ is new in current environment then
         14. Measure($b_y, \mathcal{F}$)
         15. $t = t + 1$
      16. $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{a_x, b_y\}$
   17. $\mathcal{P}' \leftarrow \mathcal{P}' \cup \mathcal{P}'$
   18. for $\forall s' \in \mathcal{P}'$ do
      19. $N' \leftarrow$ GETCLOSESTNEIGHBORINGPLANS($s', \mathcal{U}'$)
      20. $a \leftarrow$ GETMOSTDISSIMILARPLANONTARGET($s', N'$)
      21. $f_a(x)$ of $s'$ \leftarrow $f_t(x)$ of $a$
      22. ComputeMMOMODEL($\mathcal{U}'$)
      23. $\mathcal{P} \leftarrow$ NONDOMINATEDSORTING($\mathcal{U}'$)
      24. $\mathcal{P} \leftarrow$ top $n$ adaptation plans from $\mathcal{U}$
      25. $s_{best}' \leftarrow$ BESTPLAN($\mathcal{P}$)
      26. if $s_{best}'$ is better than $s_{best}$ then
         27. $s_{best} \leftarrow s_{best}'$
      28. /* adapt the managed system with a better plan upon every $k$ new measurements. */
      29. if $t \geq k$ and $s_{best}$ has not been sent then
         30. $t = 0$
         31. SendForADAPTATION($s_{best}$)
```

---

C. Refined MMO based Planning

Refining MMO by finding an appropriate replacement of the auxiliary objective $f_a(x)$ is non-trivial, as we need to ensure that (1) $f_a(x)$ is an easy-to-measure objective; and (2) it should be within the same scale as that of $f_t(x)$ (hence the $w$ can be removed by setting $w = 1$) while (3) similarly performing adaptation plans on $f_t(x)$ can have very different values on $f_a(x)$, which is an important requirement for multi-objectivization with MMO as shown by Chen and Li.

Since MMO is often used with a population-based multi-objective search algorithm (we use NSGA-II in this work), to replace the $f_a(x)$ in MMO, we focus on using the neighboring adaptation plans in the current population (and generated offsprings) during the search. Algorithm 1 shows the pseudo-code for the refined MMO using NSGA-II as the underlying algorithm for SAS planning. As can be seen, to use MMO, the key is that selecting which adaptation plan to preserve will be done in the transformed space ($g_1(x)$ and $g_2(x)$) instead of the original space (line 22). Since it is impractical to influence the managed system during planning, the measurements of
unique adaptation plans are conducted on a Cyber-Twin of the system to be managed (line 2, 11 and 14).

Specifically, at each iteration/generation, we compute the $f_a(x_a)$ for an adaptation plan $x_a$ as the follows:

1) Find the adaptation plan(s) $N = \{x_1, x_2, ..., x_k\}$ with the closest distance (in terms of the variable space $x$) to $x_a$, in the current population/offsprings. We use Euclidean distance as it offers good discriminative power (line 19).

2) Identify the $x_a$ from $N$ such that its $f_i(x_a)$ has the largest difference to $f_i(x_a)$, and then set $f_a(x_a) = f_i(x_a)$. In this way, we reduce the chance of having $f_i(x_a) = f_a(x_a)$, which essentially invalidates multi-objectivization (line 20-21).

Clearly, the new auxiliary objective $f_a(x)$ is easy-to-measure and it will have the same scale as $f_i(x)$. While the neighboring plans for SASs may behave similarly on the performance, Nair et al. [3] have shown that for highly-configurable systems, those neighboring plans can also lead to radically different results. The is because, for example, when two plans only differ on whether to turn on the cache option, the performance results can differ dramatically even though these plans exhibit very similar representation in the search algorithm. Similar observations have also been registered by the others [4]. As a result, our design also ensures that adaptation plans can have similarly-performing $f_i(x)$ but very different $f_a(x)$.

Since LiDOS is a lifelong dynamic optimization framework, the multi-objectivization planner terminates when all the search space in an environment has been explored, or otherwise, it runs throughout the lifetime of the SASs. An adaptation to the managed system is triggered every $k$ new measurements and when a better plan has been found (line 29-31). Here, $k$ serves as a parameter that determines the extent of convergence and tolerated overhead for the planning in LiDOS.

D. Architecture

Figure 4 shows the architecture of LiDOS, which uses a hierarchical feedback loop. As can be seen, we adopt the external management style for SAS [33] where the managing system and the managed system are separated.

The multi-objectivization planner, which is the core of LiDOS, runs in the inner loop. To enable lifelong optimization, the planning process runs as soon as the managed system is deployed and it will not stop unless the search space in an environment has been covered or the managed system terminates. To achieve dynamic optimization, the planner searches for a better adaptation plan by measuring a Cyber-Twin of the managed system, hence providing accurate measurements of the adaptation plans without influencing the system at production. Such a Cyber-Twin serves as a replica of the managed system (may be deployed on a different machine) and runs under exactly the same environment as what has been experienced by the managed system. In most cases, the Cyber-Twin can simply be another running instance of the managed system.

In contrast, the outer loop follows the normal adaptation process where it notifies the Cyber-Twin in the inner loop whenever there is an environmental change, thereby the Cyber-Twin can be configured under the same environment (e.g., by feeding the same workload to it). The outer loop will also set $t = 0$ in the planner and trigger re-measurement of the plans in the current population. In this way, the planner would be aware of the change via measuring the Cyber-Twin. Adaptation to the managed system will occur whenever there are $k$ new measurements and the planner in the inner loop finds a better adaptation plan.

Note that the architecture of LiDOS can be easily adopted to other common patterns for SASs, e.g., the MAPE-K loop.

IV. Experiment Setup

In this work, we seek to understand the following research questions (RQs):

- **RQ1**: Is being “dynamic” more beneficial than being “stationary” for SASs?
- **RQ2**: How effective is the adaptation achieved by LiDOS?
- **RQ3**: How efficient for LiDOS to produce a promising adaptation plan?

We ask RQ1 to verify whether considering lifelong dynamic optimization for SASs is more necessary than its stationary counterpart where an optimization run is simply restarted in the planning. We study RQ2 to confirm the effectiveness of LiDOS against state-of-the-art planners. Yet, it would still be meaningless if LiDOS requires a large amount of planning overhead to become effective. Therefore, RQ3 seeks to understand how much speedup that LiDOS can achieve against the state-of-the-arts.

A. Subject SASs

We study three real-world SASs that are widely used in prior work of configurable and self-adaptive systems:

- **Apache Storm**: a distributed stream processing computation framework that executes a series of job batches. Adaptation means altering runtime options such as `message_size`, `num_counters`, and `num_splitters`. The changing environments are related to the workload and type of a given batch of jobs.
TABLE I: Real-world self-adaptive systems studied.

| SAS | | Environment | \(|\mathcal{P}|\) | \(|\mathcal{O}|\) | \(|\mathcal{S}|\) | Used By |
|-----|-----|-------------|---------|--------|--------|--------|
| STORM | Throughput | WORDCOUNT ROLLINGCOUNT | 12 | 1,914 | 8 | [8], [3], [35] |
| KERAS/DNN | AUC | SHAPESALL ADIAC | 12 | 16,384 | 56 | [8], [24] |
| x264 | Latency | 128/44 8/2 | 17 | 53,662 | 8 | [8], [37], [35] |

\(|\mathcal{P}|\), \(|\mathcal{O}|\) and \(|\mathcal{S}|\) denote performance objective, \# adaptation options and search space, respectively.

- **KERAS/DNN**: the deep neural network (DNN) from the well-known KERAS framework for classification tasks as part of a learning system. When deployed and trained, the learning system then classifies newly given samples. Adaptation refers to adjust options includes `batch_size`, `num_filters`, and `log_decay`, for (re-)training the network in the system. The environment can change depending on the given dataset for training.

- **x264**: A system for encoding videos. Adaptation means tuning runtime options includes `no_mbtree`, `bframes`, and `ip_ratio`. Environment is determined by the sizes and frames of different videos.

More detailed information about the SASs has been shown in Table I where we consider two environments that can change at runtime for each SAS and use the same setting as previous work. Note that since the measurement of an adaptation plan can be expensive (even with a Cyber-Twin) [27], it is important to find an effective plan using as few measurements as possible. To expedite the experiment, we use the readily available dataset of those systems [31], [4], [34], [35] from existing work to emulate the Cyber-Twin of the managed system.

**D. Statistical Validation**

To verify statistical significance, all experiments in this work are repeated 50 runs.

1) **Pair-wise Comparisons**: We use the following methods for comparing the target performance objective of two planners:

- **Non-parametric test**: To verify statistical significance, we leverage Wilcoxon rank-sum test [41] — a widely used non-parametric test for SBSE and has been recommended in software engineering research for its strong statistical power on pairwise comparisons [42]. The standard \(\alpha = 0.05\) is set as the significance level over 50 runs. If the \(p < 0.05\), we say the magnitude of differences in the comparisons are significant.

- **Effect size**: To ensure the resulted differences are not generated from a trivial effect, we use \(\hat{A}_{12}\) to verify the effect size over 50 runs. According to Vargha and Delaney [43], when comparing \(\text{LiDOS}\) and its counterpart in this work, \(\hat{A}_{12} > 0.5\) denotes that the \(\text{LiDOS}\) is better for more than 50% of the times. In particular, \(0.56 \leq \hat{A}_{12} < 0.64\) indicates a small effect size while \(0.64 \leq \hat{A}_{12} \leq 0.71\) and \(\hat{A}_{12} \geq 0.71\) mean a medium and a large effect size, respectively.

As such, we say a comparison is statistically significant only if it has \(\hat{A}_{12} \geq 0.56\) (or \(\hat{A}_{12} \leq 0.44\)) and \(p < 0.05\).

2) **Three or More Comparisons**: In case more than two planners need to be compared, we apply Scott-Knott test [44] on all comparisons on the target performance objective over 50 runs, as recommended by Mittas and Angelis [44]. In a nutshell, Scott-Knott sorts the list of treatments (the planners) by their median values of the target performance objective. Next, it splits the list into two sub-lists with the largest expected difference [45]. For example, suppose that we compare \(A, B,\) and \(C\), a possible split could be: \(\{A, B\}, \{C\}\), with the rank of 1 and 2, respectively. This means that, in the statistical sense, \(A\) and \(B\) perform similarly, but they are significantly better than \(C\). Formally, Scott-Knott test aims to find the best split by maximizing the difference \(\Delta\) in the expected mean before and after each split:

\[
\Delta = \frac{|l_1|}{|l_1 + l_2|} (\bar{l}_1 - \bar{l})^2 + \frac{|l_2|}{|l_1 + l_2|} (\bar{l}_2 - \bar{l})^2
\]

whereby \(|l_1|\) and \(|l_2|\) are the sizes of two sub-lists \((l_1, l_2)\) from list \(l\) with a size \(|l|\), \(\bar{l}_1, \bar{l}_2\), and \(\bar{l}\) denote their mean values of the target performance objective.

**C. Parameter Settings**

For the key parameters of the stochastic search algorithm in all the planners, we apply the binary tournament for mating selection, together with the boundary mutation and uniformed crossover, as used in prior work for SASs [9], [40], [14]. The mutation and crossover rates are set to 0.1 and 0.9, respectively, with a population size of 20, which is widely used [9], [27].

To provide fair statistics for RQ1 and RQ2 (especially for those stationary counterparts), we set the adaptation interval \(k\) as 150 measurements (of the Cyber-Twin), as it is the smallest number that there is no change on the best adaptation plan in the last 5 iterations (for all planners), implying a reasonable convergence. Clearer trajectories of the performance change over time will be demonstrated for RQ3.
During the splitting, we apply a statistical hypothesis test $H$ to check if $l_1$ and $l_2$ are significantly different. This is done by using bootstrapping and $A_{12}$ [43]. If that is the case, Scott-Knott recurses on the splits. In other words, we divide the planners into different sub-lists if both bootstrap sampling and effect size test suggest that a split is statistically significant (with a confidence level of 99%) and with a good effect $A_{12} \geq 0.6$. The sub-lists are then ranked based on their mean values of the target performance objective.

V. RESULTS

A. RQ1: Benefit of Being “Dynamic”

1) Method: To answer RQ1, we compare LiDOS and LiDOS_{sta} over changing environments under reasonable convergence of the planning. For each SAS, we run the system transits from one environment to another and set $k = 150$. That is, for all planners, the planning is allowed to execute for 150 measurements and trigger an (better) adaptation. After the task in the current environment has been completed, the environment would be changed to another and the planner runs for a further 150 measurements before triggering the next adaptation. We report the target performance objective achieved by the last measurement. Wilcoxon rank-sum test and $A_{12}$ are used for pair-wise comparisons over 50 runs.

2) Result: As we can see from Table II LiDOS wins all the 6 cases from which 4 of them have $\hat{A}_{12} > 0.5$ and at least medium effect size. The improvement in terms of median and IQR of the target performance objective has also been considerably high. Remarkably, it achieves almost 10× median improvement on the case of x264 when the environment changes from the video with 128/44 to one with 8/2.

Since LiDOS and LiDOS_{sta} share the same optimization model and search algorithm, the above are clear evidence on the benefits of being dynamic when planning for SASs, as the stationary counterpart, which restarts the search upon environmental change, will waste the accumulated information that can be useful after the change. Therefore, we say:

RQ1: Explicitly considering “dynamic” in the planning of SASs is generally more beneficial than its stationary counterpart with up to 10× improvement.

B. RQ2: Effectiveness of Adaptation

1) Method: For RQ2, we compare LiDOS with the state-of-the-art pseudo-dynamic planner (Ramirez et al. [12] & Kinneer et al. [15]) and the stationary planner (Chen et al. [9]) discussed in Section IV-B. We use the same setting for environmental changes as that for RQ1 and report the results by the last measurement. Since there are more than two planners, we use Scott-Knott test to rank the results over 50 runs.

2) Result: The results are reported in Table III. Clearly, LiDOS has been constantly ranked as the first with considerable improvement on the median performance over the others. Overall, the pseudo-dynamic planner by Ramirez et al. [12] & Kinneer et al. [15] tends to be better than the stationary one used by Chen et al. [9], but the improvement is not significant. This is not surprising, as the SOGA may be trapped at some undesired local optima, which tends to be not useful but misleading for the search after the environmental change.

To confirm the effectiveness of preserving different local optima for dynamic optimization in SAS planning, Figure 5 plots the performance of the adaptation plans found before and after the environmental change for one run of STORM. From Figure 5, we see that the plans preserved by LiDOS right before the change occur contain a wide range of local optima (In fact, $c_3$ is the global optimum) — 6 out of 7 are the best among all their neighboring plans. In contrast, the pseudo-dynamic planner by Ramirez et al. [12] & Kinneer et al. [15] is more tempting to converge to one adaptation plan,  

### Table II: Comparing LiDOS and the variant of LiDOS with a stationary setting (denotes as LiDOS_{sta}) over 50 runs. 

| Method     | Median | IQR    | $\hat{A}_{12}$ (p value) |
|------------|--------|--------|--------------------------|
| LiDOS_{sta} | 4803   | 206    | 0.56 (p = 0.347)         |
| LiDOS      | 4883   | 122    |                          |

Throughput (msgs/min) for STORM, WORDCOUNT → ROLLINGCOUNT

| Method     | Median | IQR    | $\hat{A}_{12}$ (p value) |
|------------|--------|--------|--------------------------|
| LiDOS_{sta} | 10119  | 36     | 0.75 (p < 0.001)         |
| LiDOS      | 10200  | 60     |                          |

### Table III: Comparing LiDOS and the variant of LiDOS with a stationary setting (LiDOS_{sta}) over 50 runs.

| Method     | Median | IQR    | $\hat{A}_{12}$ (p value) |
|------------|--------|--------|--------------------------|
| LiDOS_{sta} | 0.064  | 0.105  | 0.70 (p = 0.038)         |
| LiDOS      | 0.274  | 0.225  |                          |

### Table IV: Latency (s) and Throughput (msgs/min) for x264, 8/2 → 128/44

| Method     | Median | IQR    | $\hat{A}_{12}$ (p value) |
|------------|--------|--------|--------------------------|
| LiDOS_{sta} | 31.147 | 32.210 | 0.80 (p < 0.001)         |
| LiDOS      | 3.877  | 0.703  |                          |

Latency (s) for x264, 128/44 → 8/2

| Method     | Median | IQR    | $\hat{A}_{12}$ (p value) |
|------------|--------|--------|--------------------------|
| LiDOS_{sta} | 113.350 | 13.610 | 0.70 (p < 0.001)         |
| LiDOS      | 106.510 | 9.050  |                          |

Latency (s) for x264, 8/2 → 128/44

The task under an environment will not be processed until the first adaptation has been taken.

While the planner runs continuously, we observed no further new measurements when the environmental change took place for all cases.
TABLE III: Comparing LiDOS and the state-of-the-art pseudodynamic and stationary planners for SASs over 50 runs. In each case, the planners are sorted by Scott-Knott rank, median, and then IQR. The same formats as Table II applied.

| Method       | Rank | Median | IQR |
|--------------|------|--------|-----|
| LiDOS        | 1    | 4883   | 122 |
| Ramirez et al. & Kinneer et al. | 2 | 4761   | 0   |
| Chen et al.  | 2    | 4761   | 244 |

Table: Throughput (msgs/min) for STORM, WORDCOUNT → ROLLINGCOUNT

| Method       | Rank | Median | IQR |
|--------------|------|--------|-----|
| LiDOS        | 1    | 10200  | 60  |
| Ramirez et al. & Kinneer et al. | 2 | 10140  | 21  |
| Chen et al.  | 2    | 10119  | 116 |

Table: Throughput (msgs/min) for STORM, ROLLINGCOUNT → WORDCOUNT

| Method       | Rank | Median | IQR |
|--------------|------|--------|-----|
| LiDOS        | 1    | 0.274  | 0.225|
| Ramirez et al. & Kinneer et al. | 1 | 0.174  | 0.233|
| Chen et al.  | 2    | 0.059  | 0.113|

AUC for KERAS/DNN, SHAPEALL → ADAIC

| Method       | Rank | Median | IQR |
|--------------|------|--------|-----|
| LiDOS        | 1    | 0.292  | 0.292|
| Chen et al.  | 2    | 0.163  | 0.258|
| Ramirez et al. & Kinneer et al. | 2 | 0.112  | 0.245|

AUC for KERAS/DNN, ADAIC → SHAPEALL

| Method       | Rank | Median | IQR |
|--------------|------|--------|-----|
| LiDOS        | 1    | 3.877  | 0.703|
| Ramirez et al. & Kinneer et al. | 1 | 4.190  | 26.247|
| Chen et al.  | 2    | 6.663  | 32.307|

Latency (s) for X264, 128/44 → 8/2

| Method       | Rank | Median | IQR |
|--------------|------|--------|-----|
| LiDOS        | 1    | 106.510| 9.050|
| Ramirez et al. & Kinneer et al. | 2 | 113.310| 9.550|
| Chen et al.  | 3    | 120.660| 13.680|

Latency (s) for X264, 8/2 → 128/44

which can be a local optimum. When re-measuring the same adaptation plans after the change (Figure 5), we observe that:

1) LiDOS immediately obtain the best adaptation plans among those found by the state-of-the-art, i.e., Chen et al. under LiDOS, which is merely one of the local optima that it found before the change in Figure 5. This shows the usefulness of local optima for coping with the dynamic changes that occur in the planning of SASs.

2) Although the single plan found by Ramirez et al. is promising before (the 2nd best in Figure 5), it can be significantly worsened after the change (only the 7th best in Figure 5). This reveals the importance of exploring and preserving a wide range of local optima.

3) The random plans in the stationary planner (Chen et al.) are of generally worse performance than the other two.

As a result, we conclude that:

RQ2: LiDOS is effective as it is generally ranked better (in the statistical sense) than the state-of-the-art with considerable improvement because multiple local optima can be preserved upon environmental change during the lifelong dynamic optimization.
In summary, we state that:

**RQ3:** *LiDOS is more efficient than the state-of-the-art planners, achieving up to 5X and 10X speedup over the pseudo-dynamic and stationary one, respectively.*

VI. THREATS TO VALIDITY

Threats to internal validity can be related to the $k$, which determines the time allowed for planning. We set $k = 150$ in this work since we found that it reaches reasonable convergence for the SASs studied, i.e., for all planners, the best adaptation plan found does not change for the last 5 consecutive iterations. Further, in RQ3, we examine the trajectory of planning, which also indicates what would happen if a smaller $k$ is used. The other parameter settings follow what has been pragmatically used from the literature [9], [40], [14], [27]. However, we acknowledge that examining alternative parameters (and larger $k$) can be an interesting topic and we leave this for future work. To mitigate bias, we repeated 50 runs for each case.

When comparing the efficiency, we see that *LiDOS* achieves significant improvement over the others: $1.4 \times$ to $5 \times$ speedup over the pseudo-dynamic planner by Ramirez et al. [12] & Kinneer et al. [15]: $2.5 \times$ to $10 \times$ speedup on the stationary planner by Chen et al. [9].

In summary, we state that:

**RQ3:** *LiDOS is more efficient than the state-of-the-art planners, achieving up to 5X and 10X speedup over the pseudo-dynamic and stationary one, respectively.*
A. Stationary Optimization for SASs

In contrast to the dynamic (and pseudo-dynamic) optimization, stationary planners take a simpler strategy: the search is restarted from scratch whenever there is a new environmental change (or indeed a new search run under a fixed frequency) [9], [10], [11], [15], [47], [48], [49], [50]. Among other, Chen et al. [9] propose FEMOSAA, a framework that relies on stationary optimization for the SAS planning. While FEMOSAA is designed to make adaptation under a fixed frequency throughout the lifetime of SAS, the search and planning does not run continuously — every time the search restarts with some randomly generated adaptation plans. Elkhodary et al. [10] and Gerasimou et al. [11] also follow similar way with an exact/stochastic search, but the planning is conducted only when there is an environmental change or severe violation of performance requirement has been detected — yet again each time the search starts from scratch.

Clearly, stationary optimization ignores the information that could have been useful for the next planning, which makes the issues of local optima more difficult to address. With LiDOS, we seek to better exploit such information to improve the effectiveness and efficiency of SAS planning.

B. Dynamic Optimization for SASs

Dynamic optimization for SASs has been traditionally referred to the fact that a vanilla search algorithm runs continuously for planning [12], [13], [14], [15]. For example, Ramirez et al. [12] design PLATO, a framework that adopts SOGA for SAS planning. They have shown that without specific designs, SOGA can deal with environmental change by detecting the changes in fitness. As such, PLATO meets the concept of lifelong optimization. A similar notion that has been used for SASs is seeding — the planning and search are “seeded” by adaptation plans from previous runs of planning [13], [14]. Kinneer et al. [15] follows such a scheme using SOGA for searching in the SAS planner. Conceptually, seeding is similar to running the search and planning continuously and thus is referred to as dynamic optimization for SASs in the literature.

Although the search algorithm is dynamic in nature, the above differs from the true definition of dynamic optimization in the literature [26]. This is because no specific mechanism has been designed to handle the possible landscape change as the search proceeds, but leaving with the hopes that the algorithm would eventually cope with those changes in some ways. Therefore, the above work can be considered as pseudo-dynamic, and hence the information that can be leveraged after environment change is limited. In contrast, LiDOS contains the refined MMO, which is used and designed explicitly to address the formulated multi-modal optimization problem by taking the characteristics of SAS planning into account, aiming to better handle the dynamic changes as the planning runs. As we have shown, preserving multiple local optima (without harming the tendency towards the global optimum) can indeed provide more useful information than simply detecting the changing values of performance objective when the environment changes.

C. Control Theoretical Planning

Apart from the SBSE, control theoretical approaches have also been applied for SAS planning [17], [18], [19]. Among others, Maggio et al. [18] use Kalman filter to revise and update the state values of the controller model, where the core is a model predictive control scheme. Shevtsov and Weyns [19] also adopt a control theoretical approach, but they extend such with the simplex optimization method, which is essentially a kind of search algorithm.

We consider the contributions in this work as being complementary to the control theoretical planners rather than competitive, because fundamentally the purpose of search and optimization is to seek the global optimum. As a result, LiDOS can be integrated with a control theoretical planner such that it seeks to find the global optimum of the current system state modeled by the controller, similar to what has been done by Shevtsov and Weyns [19].

D. Performance Learning for SASs

Another relevant thread of research is performance learning for configurable systems. Various methods have been used, such as neural network [20], [21], linear regression [20], ensemble learning [23] and transfer learning [25]. However, the aim is to learn an accurate function that captures the correlation between adaptation options and the performance while we target planning, therefore our contributions are orthogonal to the above. For example, with an accurate performance model, the Cyber-Twin in LiDOS can be replaced by such a model, achieving faster measurements and hence boosting the planning.

VIII. Conclusion

This paper presents LiDOS, a lifelong and truly dynamic optimization framework with a hierarchical feedback loop architecture for SAS planning. We formulate the lifelong dynamic optimization for SAS as a multi-modal optimization problem, which is solved by the refined meta-objectivization (MMO) model. The aim is to preserve as many local optima as possible without harming the tendency towards the global optimum, hence providing useful information for planning after the environmental change.

Experiments on three diverse real-world SASs and different environmental changes show that compared with the stationary counterpart and other state-of-the-art planners, LiDOS is:

- more effective, as it achieves considerably better adaptation plan than the stationary counterparts and other state-of-the-art planners with up to 10× improvement;
- and more efficient, since it exhibits 1.4× to 10× speedup on producing promising plans.

This work is among the first attempt to explicitly handle the “dynamic” on landscape changes during the run of the search for SAS planning. We show that such a lifelong dynamic optimization is factual for planning in SAS, which can excite a few future research directions, such as considering information from all previous planning runs and memory mechanisms to more precisely track the details of landscape changes between different environments.
[27] T. Chen and M. Li, “Multi-objectivizing software configuration tuning,” in ESEC/FSE ’21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021, D. Spinellis, G. Gousios, M. Chechik, and M. D. Penta, Eds. ACM, 2021, pp. 453–465. [Online]. Available: https://doi.org/10.1145/3468264.3468555

[28] M. Harman, S. A. Mansouri, and Y. Zhang, “Search-based software engineering: Trends, techniques and applications,” ACM Computing Surveys (CSUR), vol. 45, no. 1, p. 11, 2012.

[29] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph, “Exploratory landscape analysis,” in 13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011, N. Krasnogor and P. L. Lanzi, Eds. ACM, 2011, pp. 829–836. [Online]. Available: https://doi.org/10.1145/2001576.2001690

[30] J. Tavares, F. B. Pereira, and E. Costa, “Multidimensional knapsack problem: A fitness landscape analysis,” IEEE Trans. Syst. Man Cybern. Part B, vol. 38, no. 3, pp. 604–616, 2008. [Online]. Available: https://doi.org/10.1109/TSMCB.2008.915539

[31] G. Singh and K. Deb, “Comparison of multi-modal optimization algorithms based on evolutionary algorithms,” in Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings, Seattle, Washington, USA, July 8-12, 2006, M. Cattolico, Ed. ACM, 2006, pp. 1305–1312. [Online]. Available: https://doi.org/10.1145/1143997.1144200

[32] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182–197, 2002.

[33] M. Salehie and L. Tahvildari, “Self-adaptive software: Landscape and research challenges,” ACM Trans. Auton. Adapt. Syst., vol. 4, no. 2, pp. 14:1–14:42, 2009. [Online]. Available: https://doi.org/10.1145/1516553.1516558

[34] P. Jamshidi, M. Velez, C. Kästner, and N. Siegmund, “Hyperparameter optimization for effort estimation,” CoRR, vol. abs/1805.00336, 2018. [Online]. Available: http://arxiv.org/abs/1805.00336

[35] Y. Gao, Y. Zhu, H. Zhang, H. Lin, and M. Yang, “Resource-guided configuration space reduction for deep learning models,” in 43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 23-30 May 2021. IEEE, 2021, pp. 175–187. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00028

[36] S. Kumar, R. Baboons, T. Chen, K. Li, and R. Buyya, “Multi-tenant cloud service composition using evolutionary optimization,” in 24th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2018, Singapore, December 11-13, 2018. IEEE, 2018, pp. 972–979. [Online]. Available: https://doi.org/10.1109/PADSW.2018.8644460

[37] K. Li, Z. Xiang, T. Chen, and K. C. Tan, “Bilo-cpd: Bi-level programming for automated model discovery in cross-project defect prediction,” in 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020. IEEE, 2020, pp. 573–584. [Online]. Available: https://doi.org/10.1109/TSE.2012.45

[38] T. Chen and R. Baboons, “Self-adaptive trade-off decision making for autoscaling cloud-based services,” IEEE Trans. Serv. Comput., vol. 10, no. 4, pp. 618–632, 2017. [Online]. Available: https://doi.org/10.1109/TSC.2015.2499770

[39] ——, “Symbiotic and sensitivity-aware architecture for globally-optimal benefit in self-adaptive cloud,” in 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings, Hyderabad, India, June 2-3, 2014, G. Engels and N. Bengmou, Eds. ACM, 2014, pp. 85–94. [Online]. Available: https://doi.org/10.1145/2593929.2593931