Fast Subspace Identification Method Based on Containerised Cloud Workflow Processing System

Runze Gao, Yuanqing Xia, Senior Member, IEEE, Guan Wang, Liwen Yang, and Yufeng Zhan

Abstract—Subspace identification (SID) has been widely used in system identification and control fields, since it can estimate system models while only relying on the input and output data using reliable numerical operations. However, the high-dimension Hankel matrices are involved to store these data and used to obtain the system models, which increases the computation amount of SID and makes SID unsuitable for the large-scale or real-time identification tasks. In this paper, a novel fast SID method based on cloud workflow processing approach and container technology is proposed to accelerate the traditional algorithm. First, a workflow establishment method of SID is designed to match the distributed cloud environment, based on the computational feature of each calculation stage. Second, a containerised cloud workflow processing system is established to execute the logic- and data- dependent SID workflow mission based on the Kubernetes system. Finally, the experiments show that the computation time is reduced by at most 91.6% for the large-scale SID mission and decreased to within 20 ms for the real-time mission parameter.

Note to Practitioners—Subspace identification has became a widely used method in various fields, including power grids, chemical processing, data-driven control, and fault detection. However, as systems become larger and more complex, the computational challenges increase. To address this issue, this paper proposes a workflow-based method for subspace identification that can be executed in a cloud environment to accelerate the process. This note outlines the steps that practitioners can take to apply this method. The first step is to design a workflow structure as proposed method in this paper. This structure should be customized to fit the specific needs of the practitioner’s application. The second step is to build a containerized cloud workflow processing system that can execute the workflow. This system should be based on the Kubernetes system and designed to handle the specific computational requirements of the workflow. Practitioners who work in fields where computational efficiency is crucial for system identification operations can benefit from the proposed method. By following the steps outlined above, practitioners can streamline the process of subspace identification and achieve improvements in computational efficiency.

Index Terms—Subspace identification, cloud computing, container technology, cloud workflow processing, directed acyclic graph.

I. INTRODUCTION

In the decades, subspace identification (SID) methods have attracted substantial interest in system identification and control fields. SID methods have been applied in a wide range of domains, such as power grids, chemical processing, data-driven control and fault detection [1], [2], [3]. By SID methods, standard state-space models could be built based on the historical data and used for further analyses and designs. Several important algorithms, including CVA, N4SID, MOSEP and data-driven predictive control, have been developed [1], [3], [4], [5]. SID methods enable accurate identification of state-space models using the reliable matrix operations, such as singular value decomposition (SVD), and the state-space models can be consistently identified from the input and output data under mild conditions. In SID methods, the high-dimensional and low-rank Hankel matrices are derived to store the historical data and obtain the final system model. Its major drawback, on the other hand, is that the large-scale data matrices increase the computation amount of matrix operations, which leads to SID methods being not suitable for the large-scale or real-time identification tasks.

Several methods have been proposed to reduce the computation time of system identification (SID) methods. The majority of these methods focus on improving the numerical algorithm design. For instance, a recursive method is employed to estimate the subspace spanned by the column vectors of the extended observability matrix, resulting in enhanced computation speed [6]. Based on this recursive method, computational complexity is further reduced by exploiting the data equation structure and employing array algorithms to solve linear problems [7]. For stochastic SID methods, the true system order is unknown, identifying results at different model orders requires comparison, leading to significant computational burdens. In addressing this issue, [8] proposes a one-calculation method by determining the most desired model order and provides a proof of equivalence. However, this work still remains the challenge of processing high-dimensional data. Therefore,
proposes a fast method to calculate the covariance matrices of parameters, which is an intermediate stage of [8]. Apart from improvements in numerical algorithms, there are efforts to enhance computational effectiveness by leveraging more powerful computing devices [2], [10]. For instance, [2] addresses a novel method to speed up large-scale ambient oscillation identification using a multi-core server. Similarly, [10] presents a fast SID method to identify hyperspectral information using FPGA. These methods offer advantages over numerical improvements by harnessing the computing power of multi-processor structures. However, [2] and [10] are still constrained to traditional local devices with fixed computing resources such as CPU and memory, lacking elasticity.

Nowadays, many scientific missions, such as deep learning, genetic calculation and earthquake wave analysis, have been processed in cloud server because of the computing ability of cloud computing [11], [12], [13], [14], [15], [16]. SID is also a kind of scientific mission, of which the calculation is highly time-consuming. However, the combined work of cloud computing and the SID mission does not exist. Furthermore, the computing mode of the SID mission is centralized, but the structure of the cloud environment is distributed. If the SID mission is deployed in a cloud server directly, the computation efficiency would not be improved since the computing ability of distributed structure is not utilized. Therefore, this paper proposes the workflow-based SID method, which restructures the SID mission into the cloud workflow form, for matching the distributed cloud environment. As shown in Fig. 1, the mentioned cloud workflows are represented graphically by directed acyclic graphs (DAGs). The nodes represent the computational tasks, and the directed edges between the nodes determine the interdependencies between the tasks [17], [18], [19]. In the cloud environment, the tasks in a workflow would be scheduled to different processing nodes, from which the required computing resources are encapsulated from a shared resource pool.

Therefore, a fast, stable and flexible scheme of encapsulating computing resources is required to execute workflow tasks. Cloud workflow processing based on a virtual machine (VM) is the present major approach [19], [20], [21], [22]. VM technology provides a virtual operating system to access computing resources and create isolated spaces for executing the tasks. However, the operation overheads of VM are quite high, including the creation, startup, configuring times and configuring complexity. For example, the creation and startup stages of VM cost several minutes and 30-40 s, respectively. To reduce the overheads, container technology is created and applied [23], [24], [25], [26]. This technology enables the smaller isolated processes for processing tasks to be created and started in an instant, and released automatically when the tasks are finished. Compared with VM technology, container also has potential benefits regarding the computing resource usage, overhead cost, migration speed across hosts and template sharable ability, as summarized in Table I [24].

Motivated by the above reasons, this work focuses on enhancing computation efficiency not through numerical algorithm design, as seen in [6], [7], [8], and [9], but rather from the perspective of computer-aided computational method. In comparison to works [2], [10], this work utilizes flexible cloud computing resources, overcoming performance limitations imposed by finite fixed resources. Consequently, the execution issue in cloud workflow processing is introduced and addressed in this work. Moreover, the integration of cloud computing introduces new possibilities for future applications and researches. In summary, to improve the computational efficiency of SID methods, by adopting cloud workflow processing approach and container technology, the main contributions of this paper on the novel method are summarized.

1) A workflow-based method is proposed to accelerate SID based on the cloud workflow processing approach and container technology. In particular, a workflow establishment method of SID is designed to match the distributed cloud environment, based on the computational feature of each calculation stage.

2) A containerised cloud workflow processing system is established to execute the logic- and data- dependent SID workflow missions based on the Kubernetes system.

3) Based on this system, the Monte Carlo experiments are carried out. The results show that the computation time is reduced by at most 91.6% for the large-scale SID mission and decreased to within 20 ms for the real-time mission parameter.

The rest of this paper is organized as follow. Section II presents the preliminary. Section III provides the overview of the cloud-based SID method. The computational features analyses and the establishment of the SID workflow are presented in Section IV. The design and establishment of the containerised cloud workflow processing system are provided in Section V. The experiment evaluation and the result

---

**TABLE I**

| Parameter                      | Container technology | VM technology |
|-------------------------------|----------------------|---------------|
| Kernel state                  | Booting needed      | Operational   |
| Creation time                 | < 1 s                | Minutes       |
| Start/stop time               | < 50 ms              | 30-40 and 5-10 s |
| Startup overhead              | Low                  | Relatively high |
| Configuring complexity        | Medium               | High          |
| CPU/memory usage              | Low                  | High          |
| Migration                     | Quick                | Slow          |
| Template sharing ability      | Strong               | Weak          |

---

Fig. 1. The examples of the cloud workflow structure.
Algorithm 1 N4SID: A Deterministic Subspace System Identification Method

Input: The scale parameters $N, j$ and the Hankel matrices of inputs and outputs $U_p, U_f, U_p^+, U_f^+, Y_p, Y_f, Y_p^+, Y_f^+$.

Output: The identified state-space matrices: $A, B, C$ and $D$.

1: Calculate the oblique projections:
   \[ O_i = Y_f / U_i W_p, \]
   \[ O_{i-1} = Y_f' / U_i W_p^+. \]

2: Calculate the SVD of the weighted oblique projection:
   \[ W_1 O_i W_2 = U S V^T. \]

3: Determine the order by inspecting the singular values of $S$ and partition the SVD accordingly to obtain $U_1, S_1$.

   \[ U S V^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} S_1 & 0 \\ 0 & S_2 \end{bmatrix} \begin{bmatrix} V_{1}^T \\ V_{2}^T \end{bmatrix}. \]

4: Determine the extended matrices $\Gamma_i^+$ and $\Gamma_{i-1}^+$ and estimated state sequences $X_i$ and $X_{i+1}$.

5: Solve the set of linear equations for $A, B, C$ and $D$:
   \[ \begin{bmatrix} X_{i+1} \\ Y_i \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} X_i \\ U_i \end{bmatrix}. \]

Analyses are provided in Section VI. The final section presents the conclusion and future work.

II. PRELIMINARY OF SUBSPACE IDENTIFICATION

In this preliminary, the standard SID algorithm is recalled. Consider the deterministic state-space model to be identified:

\[ x(k+1) = Ax(k) + Bu(k) \]
\[ y(k) = Cx(k) + Du(k) \]

where the system matrices $A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times m}, C \in \mathbb{R}^{l \times n}, D \in \mathbb{R}^{l \times m}$.

In non-sequential data processing, Hankel matrices are always used to store and deal with data. For example, the Hankel matrices of the input data series $\{u(0), u(1), \ldots, u(2N+j-2)\}$ are defined as

\[ U_p = \begin{bmatrix} u(0) & u(1) & \cdots & u(j-1) \\ u(1) & u(2) & \cdots & u(j) \\ \vdots & \vdots & \ddots & \vdots \\ u(N-1) & u(N) & \cdots & u(N+j-2) \end{bmatrix}, \]

\[ U_f = \begin{bmatrix} u(N) & u(N+1) & \cdots & u(N+j-1) \\ u(N+1) & u(N+2) & \cdots & u(N+j) \\ \vdots & \vdots & \ddots & \vdots \\ u(2N-1) & u(2N) & \cdots & u(2N+j-2) \end{bmatrix}. \]

where the subscript “$p$” stands for “past”, “$f$” means “future” and $N, j \in \mathbb{N}^+$. The Hankel matrices of \{y(0), y(1), \ldots, y(2N+j-2)\} are written likewise, denoted as $Y_p$ and $Y_f$. The matrix containing the $U_p$ and $Y_p$ is $W_p$:

\[ W_p = \begin{bmatrix} Y_p \\ U_p \end{bmatrix} \]

To keep the convergence of the identified results, the parameter $j$ is required to be much larger than $N$ [1]. Thus the Hankel matrices in system identification are always high-dimension and low-rank, which inspires the authors to further handle the matrices. As one of the classical SID methods, N4SID is summarized as Algorithm 1 [4], where the oblique projection and other new variables are defined in Appendix A, and the details of step 4 are described in Appendix B.

III. OVERVIEW OF CLOUD-BASED SUBSPACE IDENTIFICATION

In this section, the overview of the cloud-based SID method is provided, as shown in Fig. 2. This overview consists of three layers, which are the cloud platform, terminal and users layers. Some important modules are described in Table II. The cloud platform layer is the upper part of this overview. After receiving the input and output data from the edge nodes, the data collection modules transmit these data to the workflow data entry. Meanwhile, the SID workflow template is pulled to the workflow template entry from the template register. Then, the data and workflow structure as well as task images are received by the containerised cloud workflow processing system (with the star symbol in Fig. 2), which is designed in Section V. This system creates containers and schedules them to different computing nodes to keep load balance. Finally, the identified models are obtained and transmitted to the users node.

The terminal layer is in the lower right side of this overview. There would be multiple edge nodes distributed in different geographic positions. Each edge node is a relatively independent system close to the terminal plants, which generate original data. However, the computing ability is constrained in the edge nodes by the device’s cost, deployment space and energy supply. Thus, it is beneficial to bring the computation-intensive SID missions to the cloud platform. The users layer is in the upper left side of this overview. In the users node, the received models can be applied in various functions, such as system monitoring, data-driven control and fault-detecting, etc.

IV. FAST WORKFLOW-BASED SUBSPACE IDENTIFICATION METHOD

In this section, the computational features of all calculation stages in Algorithm 1 are analyzed in detail. The results show that SVD is the most time-consuming stage. Then, we carry out the parallelization of SID method based on the truncated SVD algorithm. Next, the workflow of SID method in DAG form is established. Finally, the computational complexity analysis is provided.
TABLE II
DESCRIPTIONS OF PART DEFINED MODULES IN FIG. 2

| Module                          | Description                                                                 |
|---------------------------------|-----------------------------------------------------------------------------|
| Cloud Resource Pool             | A abstract concept for describing all computing resource, including physical server, VM and container |
| Template Register               | A sharing warehouse for storing reusable images of pre-defined tasks in SID method |
| Workflow Template               | The template consisting of the task images and the topology structure between the tasks |
| Workflow Template Entry         | A middle module to provide the template of the SID workflow pulled from the Docker registry |
| Workflow Data Entry             | A middle module to provide the input and output data of the identified plant for the SID workflow |

A. Feature Analysis of Each Calculation Stage

To discuss the computational features, Algorithm 1 is divided into three stages, which are
- Oblique projection: compute the oblique projections $O_i, O_{i-1}$.
- SVD: compute the SVD and obtain the system order $n$ and decomposed results $U_1, S_1$.
- Estimation: compute the extended matrices $\Gamma_i^t, \Gamma_{i-1}^t$, the estimated state sequences $X_i, X_{i+1}$ and estimate the system matrices $A, B, C, D$.

Further, to analyse the averaged calculation time of each stage, a group of Monte Carlo experiments with 100 times repetition of a two-order system are carried out in a local computer with i7-3770 CPU and 8 GB memory. In Table III, the statistical results are listed by dividing into three stages, i.e., the small-, middle- and large-scale stages. In the small-scale stage, oblique projection is the main time-consuming process. But the total calculation time is relatively less, so it is unnecessary to construct workflow for this stage.

With the data scale increasing, the cost time of SID method grows and becomes hard to ignore. In the middle-scale stage, of which the parameters are always used in SID-based data-driven predictive control, SVD occupies the major stage as the percentage is over 65%. In the large-scale stage, the percentage of SVD is over 98%. That is to say, when the data amount reaches the middle- and large-scale stages, SVD is the main time-consuming stage and the total calculation times are relatively high. Thus, the speedup of SID method is indeed desired.

B. Parallelization Based on the Truncated SVD Algorithm

To accelerate the computation of SID method, we consider it in a distributed parallelization framework. Based on the analysis in the last subsection, the SVD stage would be reconstructed into a DAG, as the main part of the whole SID workflow. The truncated SVD algorithm can achieve the distributed decomposition in a lightweight way [27], which inspires us to establish the DAG of SVD.

Assume an $m \times n$ matrix $A$ is split column-wise into submatrices $A_1$ and $A_2$ with the sizes $m \times n_1$ and $m \times n_2$, respectively, and $n_1 + n_2 = n$. Conduct SVD on $A_1, A_2$ and obtain the results $A_1 = U_1 \Sigma_1 V_1^T, A_2 = U_2 \Sigma_2 V_2^T$. Then, the SVD of $A = [A_1 A_2]$ can be written as

$$
\begin{bmatrix}
A_1 & A_2
\end{bmatrix} =
\begin{bmatrix}
U_1 & U_2 \Sigma_2
\end{bmatrix}
\begin{bmatrix}
V_1^T & 0 \\
0 & V_2^T
\end{bmatrix}
= E
\begin{bmatrix}
V_1^T & 0 \\
0 & V_2^T
\end{bmatrix}
\tag{6}
$$

where $E = \begin{bmatrix} U_1 \Sigma_1 & U_2 \Sigma_2 \end{bmatrix}$. Conduct SVD on $E$ and obtain

$$
\begin{bmatrix}
A_1 & A_2
\end{bmatrix} = U \Sigma \tilde{V}^T
\begin{bmatrix}
V_1^T & 0 \\
0 & V_2^T
\end{bmatrix} = U \Sigma V^T
\tag{7}
$$

where $V^T = \tilde{V}^T \begin{bmatrix} V_1^T & 0 \\
0 & V_2^T \end{bmatrix}$ is the product of two orthogonal matrices and hence is also an orthogonal matrix.

In SID method, the matrices to be decomposed by singular values are high-dimensional since these matrices are derived...
and combined by the original Hankel matrices. But since \( j \) is required to be much larger than \( N \), the row lengths of these matrices are much less than the column widths. That is to say, these matrices are low-rank matrices in the meantime. Therefore, the low-rank approximation method can be used here to discard the low-value data in each level of decomposition, which is beneficial to reduce the computation.

The low-rank approximations would be conducted after the individual SVDs are finished, for which the criteria is that the order of magnitude difference of these matrices are much less than the column widths. That is to say, these matrices are low-rank matrices in the meantime. Therefore, the low-rank approximation method can be used here to discard the low-value data in each level of decomposition, which is beneficial to reduce the computation.

The result of truncated SVD is truncated to a rank-\( r \) approximation as follows.

\[
[A_1 \ A_2] \approx \begin{bmatrix} U_1 \Sigma_1 V_1^T \\ U_2 \Sigma_2 V_2^T \end{bmatrix} = U_r \Sigma_r V_r^T.
\]  

(8)

Note that in this design, we have not discarded the singular values within the rank. Since the order of magnitude difference between the singular values before and after the rank-number is very large, the accuracy decrease for calculation brought by the truncated singular values could be ignored, compared with the accurate result. If users want to further improve computational efficiency by truncating more singular values, the uncertainty introduced by the over-truncated data should be considered. The related results on the uncertainty expressions can be found in [28].

This operation is therefore named as merge-and-truncate (MAT) operation rather than a simple merging. When there are several partitions, the MAT operation can be conducted pairwise using a DAG-based strategy, as created in Fig. 3.

The above steps are summarized in Algorithm 2. The matrix \( A_{mxn} \) is first partitioned column-wise, and the SVD of each partition is computed in the function \textsc{ParallelSVDByCols}. The \( U, \Sigma, V \)s of all partitions are merged using the function \textsc{MergeBlocks}, which invokes the function \textsc{BlockMerge} as routine. The function \textsc{DoTruncate} carries out the low-rank approximation for \( U, \Sigma, V \) by truncating the low-value data in the matrices. The result of truncated SVD is returned by the function \textsc{ParallelSVDByCols}.



### TABLE III

Averaged Calculation Time Statistics for Topical Parameters in Monte Carlo Experiments

| Data scale | Parameters | Oblique projection | SVD | Estimation | Total (s) | Major stage |
|------------|------------|--------------------|-----|------------|-----------|-------------|
|            |            | Time (s) | Per. | Time (s) | Per. | Time (s) | Per. |          |           |
| Small-scale |            | 0.0090 | 67.7% | 0.0040 | 30.1% | 0.0003 | 2.3% | 0.0133 | Oblique projection |
|            |            | 0.0092 | 22.5% | 0.0002 | 0.5%  | 0.0409 |
| Middle-scale |            | 0.0116 | 20.5% | 0.0408 | 78.8% | 0.0004 | 0.8%  | 0.0518 | SVD |
|            |            | 0.0348 | 33.9% | 0.0676 | 65.8% | 0.0004 | 0.0%  | 0.1028 |
| Large-scale |            | 0.2566 | 2.0%  | 13.1051| 98.0% | 0.0024 | 0.0%  | 13.3741|
|            |            | 0.4375 | 0.7%  | 66.2683| 99.3% | 0.0531 | 0.0%  | 66.7589|

Fig. 3. The column-based distributed SVD method by MAT operation.

### TABLE IV

Relationships Between Tasks in Workflow and Functions

| Image | Index | Level | Function |
|-------|-------|-------|----------|
| \( Image_{ini} \) | \( Task_{ini} \) | – | Prepare Hankel matrices by creating history data |
| \( Image_A \) | \( Task_1 \) | 1 | Generate \( O_i \) for SVD tasks |
| \( Image_B \) | \( Task_2 \) | 1 | Generate \( O_{i-1} \) for calculating extended state matrix \( X_{i-1} \) |
| \( Image_C \) | \( Task_{3- Task_{12}} \) | 2 | Compute the truncated SVD of each block of \( O_i \) |
| \( Image_D \) | \( Task_{13- Task_{19}} \) | 3, 4 | 1. Merge the parent tasks’ results 2. Compute the new truncated SVD |
| \( Image_g \) | \( Task_{20} \) | 5 | 1. Merge all tasks’ results 2. Obtain the final system model |

C. DAG Establishment of Subspace Identification

Based on the above analyses, the SID workflow is established in this subsection. Fig. 4 provides the workflow structure of SID with the degree of parallelism of the SVD stage being \( n/col = 10 \). Table IV describes the functions of the tasks in the SID workflow and their relationships. There are five kinds of task images, except the initial task image \( Image_{ini} \) used in this workflow. For instance, the image \( Image_A \) can be reused ten times in the second level, of which the task indexes are \( Task_3 - Task_{12} \) but with different input data. In Fig. 4, the function and relationship of each kind of the task image can be described as follows.

1) The initialization task \( Task_{ini} \) is responsible for creating history data of the identified system and preparing the Hankel matrices for the following SID workflow.

2) Based on the Hankel matrices \( [U_p, Y_p, U_f, Y_f] \), \( Task_1 \) conducts the course of \( W_p = [U_p Y_p] \), \( O_i = f(Y_f, U_f, W_p) \) where the function \( f \) means calculating...
Fig. 4. The workflow structure of the subspace identification method (MPT = 10).

In the workflow processing, the MPT is used to represent the maximal parallel tasks in a workflow, which means that at most, MPT tasks are running in parallel during the workflow execution. In this case, the MPT is 10, and the tasks number is 20. This subsection only provides one case with specific granularity. Using the same method, other SID workflow structures could be established with different parameters for different kinds and scales of problems, and different results would be obtained.

Remark 1: In fact, there are multiple methods to construct SVD-based workflows, and the one we have chosen is a more straightforward and effective approach. This low-rank approximation method is convenient for analysis in SID algorithm, which is a special data-driven modeling method. In various scenarios, different parallelization and acceleration techniques for SVD can be selected. There are still various other types of workflow construction methods and data-driven modeling problems that remain to be explored. For instance, a Python library is developed to accelerate SVD based on QR decomposition [29], which is proven effective for complex data-driven modeling problems like geophysical features analysis [30].

D. Computational Complexity Analysis

To study the improvement of this cloud workflow method, the computational complexity of Algorithm 2 is analyzed. Without loss of generality, we assume all the blocks are divided with the same column width. Let \( m \times n \) matrix \( A \) is partitioned column-wise into \( N \) blocks and the size of each block is \( m \times s \) where \( s = n/N \). Refer to [27], the floating point operations (flops) number of a complete SVD is approximated as \( 6mn^2 + 16n^3 \).

In Algorithm 2, the first step is to execute Image\(_C\). Thus, the flops number of the SVD of each block is \( 6ms^2 + 16s^3 \). By the low-rank approximation, assume that each SVD is truncated to a \( k \)-rank matrix. Then, at each level of the DAG, the tasks with Image\(_D\) are executed. The flops number of Image\(_D\) consists of two functions, which are the cost of merging \( 2mk^2 \) and the cost of new truncated SVD \( 4mk^2 + 176k^3 \), of which the latter is made up by the cost of SVD \( 6 \cdot (2k) \cdot (2k)^2 + 16 \cdot (2k)^3 = 176k^3 \), and the cost of updating \( V \) is \( 4mk^2 \). This task image is required to repeat \( N - 1 \) times. Thus, the total flops number is

\[
T(m, n, N) = N(6ms^2 + 16s^3) + (N - 1)(6mk^2 + 176k^3).
\]  

Since \( k \leq s \), the total number of flops is

\[
T(m, n, N) < \frac{12m^2 n}{N} + \frac{192n^3}{N^2}.
\]  

Thus, when the degree of parallelism \( N \) grows, the proposed method can accelerate the SID effectively. Since most of the tasks can be executed in parallel, the total time cost would be reduced further.
Remark 2: In the practical establishment of the SID workflow, we place the last two tasks with ImageD into the export task since the time costs of the two tasks are relatively less. Thus in Fig. 4, there are only seven tasks with ImageD.

V. CONTAINERISED CLOUD WORKFLOW PROCESSING SYSTEM

This system is the core of the cloud platform layer, of which the structure is provided in Fig. 5. Through the entry modules, the SID workflow template and data are loaded to this system, which comprises the pretreatment, task manager and monitoring parts. Then, the resource requests are submitted to the Kubernetes system, and the containers are created and distributed into the nodes in the cloud resource pool. In this section, the above three parts and the Kubernetes system are described, respectively.

A. Pretreatment Part

This part is in charge of processing the received workflow template and data and submitting the results to the subsequent parts. The workflow analyzer parses the workflow template and releases the below information for each task, which would be written in YAML files by the task manager.

- The task index and the level this task belongs to.
- The pre- and post-dependence relationships of this task.
- The image that would be pulled from the template regist-

Meanwhile, the input and output data of the identified plant are stored in a remote dictionary server (redis) [31] data storage system, which could share data online among different hosts via a network. The stored data would be loaded to the entrance container when the workflow begins.

B. Task Manager Part

The task manager takes charge of releasing new tasks to the cloud resource pool. The task release controller is the kernel module of this part in which the release strategy is imported. This module generates the release command based on the resource usage of the node cluster sensed by the resource state tracker. Then, since the Kubernetes system requires the command files in YAML data format, a result decoder is designed to transcode the resource request.

Besides, uncertainty events are unavoidable and would bring negative influence in cloud workflow processing [32]. Thus, the executor of the redis server plays a role in keeping the time consistency of the container initialization. When the containers in the workflow are created, the initialization time costs of obtaining necessary information, such as the DAG map and task index are different. Thus, when one container is ready, it would send a ‘ready’ signal to the executor in which the redis server runs. When all the ‘ready’ signals are collected and acknowledged, this executor would send the ‘start’ signal to each container at the same time. In addition, the CPU dispatching mode of the container should be set as specific cores but not the proportion of CPU ownership to reduce the uncertainty caused by resource competition.

Algorithm 2 Distributed Truncated SVD Algorithm

Input: A matrix to be decomposed $A_{m	imes n}$, block width col.
Output: Decomposed results $U$, $S$ and $V$.

1: function PARALLEL_SVD_BY_COLS($A_{m	imes n}$, col):
2:   Calculate the number of the truncated blocks
3:   $N_t = \text{round}(n/\text{col} + 0.45)$
4:   where the round command means taking the nearest integer.
5:   Build a series of lists to store the truncated blocks and decomposed results $l_0 \equiv \text{list}(0), l_1 \equiv \text{list}(1), l_2 \equiv \text{list}(2), l_3 \equiv \text{list}(3)$ where the list($t$) command means creating an empty list.
6:   Fill the list $l_t$ by the $N_t$ column blocks of $A$.
7: for each column block $A_{\text{block}}$ in the list $l_t$ do
8:   Do SVD on $A_{\text{block}}$ and obtain $\hat{U}_{\text{block}}, \hat{S}_{\text{block}}, V_{\text{block}}$
9:   $A_{\text{block}} = U_{\text{block}} \Sigma_{\text{block}} V_{\text{block}}^T$
10: end for
11: function DoMergeOfBlocks($U_1$, $U_2$, $U_3$):
12:   Calculate the degree of parallelism $N_l = \text{len}(l_t)$ where the len command means obtaining the length of a list.
13:   Calculate the workflow depth of SVD stage $\text{level} = \text{ceil}(\log_2 N_l)$
14: for $i \leftarrow 1$ to level do
15:   Create a series of copies of $U_{l_0}$, $U_{l_1}$, and $U_{l_2}$
16:   $l_{U_1} \leftarrow l_{U_0}$; $l_{U_2} \leftarrow l_{U_1}$; $l_{U_3} \leftarrow l_{U_2}$
17: for $j \leftarrow 1$ to $N_l$ with the step length being 2 do
18:   Take two adjacent groups of elements of $U_{l_0}$, $U_{l_1}$, $U_{l_2}$:
19:   $l_{U_1}(j)$, $l_{U_1}(j)$, $l_{U_2}(j)$, $l_{U_2}(j)$, $l_{U_3}(j)$, $l_{U_3}(j)$
20:   Conduct BlockMerge on the above variables and return the merge results $U_{l_j}$, $\Sigma_j$, $V_j$
21: end for
22: if $N_l$ is odd then
23:   Add the last elements of $l_{U_1}$, $l_{U_2}$ and $l_{U_3}$ into $l_{U_1}$, $l_{U_2}$, and $l_{U_3}$, respectively.
24: end if
25: end for
26: Return the final merged results which are defined as $\hat{U}$, $\hat{\Sigma}$, $\hat{V}$.
27: end function
28: function BlockMerge($U_1$, $\Sigma_1$, $V_1$, $U_2$, $\Sigma_2$, $V_2$):
29:   Conduct DoTruncate on $U_1$, $\Sigma_1$, $V_1$ and return $U_{l_1}$, $\Sigma_{l_1}$, $V_{l_1}$.
30:   Conduct DoTruncate on $U_2$, $\Sigma_2$, $V_2$ and return $U_{l_2}$, $\Sigma_{l_2}$, $V_{l_2}$.
31:   Do SVD on $[U_{l_1}, \Sigma_{l_1}, V_{l_1}]$ and obtain $U_1$, $\Sigma_1$, $V_1$
32:   Calculate $V_i = \hat{V} * \text{blkdiag}(V_{l_1}, V_{l_2})$.
33:   Return $U_i$, $\Sigma_i$, $V_i$.
34: end function
35: function DoTruncate($U$, $\Sigma$, $V$):
36:   Calculate the rank of $\Sigma$, which is $r = \text{rank}(\Sigma)$.
37:   Truncate the decomposed matrices as the rank:
38:   $U_k = U(:, 1:k)$; $\Sigma_k = \Sigma(1:k, 1:k)$; $V_k = V(1:k, 1:k)$.
39: end function
C. Monitoring Part

The monitoring part consists of the resource state tracker and task state tracker, which receive the required states by the list-watch scheme [33] from the Kubernetes system.

- The resource state tracker monitors the resource usage of the node cluster to be provided for the task manager.
- The task state tracker monitors the states of containers, such as running, completed and failed, and informs the task manager if the state of a container changes.

D. Kubernetes System

In this system, the resource allocator or master node takes charge of collecting computing resources and creating new containers in the node cluster as the requirement of task manager. Then, a practical workflow would be created to referring to the abstract workflow template, as shown in Fig. 6. Each task in the abstract workflow would be translated to an executing container scheduled into a node. The inter-dependencies between the tasks would be translated into the across-nodes communications to transmit the intermediate data. In addition, the original data required by the entrance task are provided by the redis data storage system, and the identified results are also stored in this redis system, which can be accessed externally.

Remark 3: From a technical perspective, even if we trust the honesty of cloud service providers, they may still be curious about the value of data and have the capability to access it. This is known as being ‘semi-honest’. Therefore, the issue of privacy protection for cloud-based environment should be considered. To address this, we can employ two methods: differential privacy and homomorphic encryption, to ensure that user privacy remains secure. Relevant researches on this topic can be found in [34], [35], [36], and [37].

VI. PERFORMANCE EVALUATION

In this section, a simple static algorithm is applied to evaluate the effectiveness of the proposed method and system. Then, a series of comparative experiments are carried out, and the results are provided. The discussions on the acceleration efficiency, three factors consisting of data scale, resource amount and container configuration, and the comparison with the recursive methods are also provided.

A. Cloud Resource Scheduling Algorithm

To keep the load balance of the node cluster, Algorithm 3 is applied in the proposed system. First, push the prepared tasks into the empty Task_list with their resource requests. Then, detect the current load utilization of $N$ computing nodes $U_1, U_2, \ldots, U_N$. Next, schedule the task to the node with the lowest load utilization and update it. Finally, create the containers in the target nodes as the scheduling solution.

Remark 4: In practice, the container can be created and started within one second. However, this process still will cost a little time, which may influence the SID missions with strong real-time requirements, such as SID-based data-driven control. Thus, we choose the static algorithm to reduce the influence by creating all containers before the mission’s start.

B. Experiments Setup

1) Identified Dynamical Model: The proposed method is applied to analyze the measurement data from a numerical system, of which the dynamical model is given as follows:

$$A = \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0.00014 & 0.00014 \end{bmatrix}, \quad D = 0$$

(11)
The results, including the computation times and reduction percentage, are presented in Table V. From Table V, the experiment results can be divided into two parts as parameter ranges: the middle- (first two rows) and large- (last four rows) scales. For the two kinds of parameters, different acceleration efficiencies are achieved.

1) For the middle-scale parameters, the computation times are reduced to within 20 ms, which is usually the period of real-time dynamical control [38]. That is to say, the proposed fast SID method can be applied to the data-driven control with high-frequency requirements, such as vehicle control, with no need for delay compensation.

2) For the large-scale parameters, the computation times are reduced by at most 91.6%, which would be very useful for the identification of high-volume data and high-complexity model, such as power grid and multi-agents.

D. Detailed Results and Discussions Under Different Conditions

In this subsection, the discussion on the acceleration efficiency under different conditions, including data scale, resource amount and container configuration, is provided.

1) Granularity of Workflow Structure: In the parallel system, the speedup is defined as the ratio of the data processing time on a single processor and on multi-processors, expressed as

\[ S_p = \frac{T_s}{T_p} \]  

(12)

where \( T_s \) is the processing time of sequential execution, and \( T_p \) is that of parallel execution. The relationship between the speedup and workflow granularity is presented in Fig. 8. For the middle-scale parameters, the first workflow structure with the MPT = 2 achieved a better performance, while the second

Algorithm 3 Cloud Resource Scheduling Algorithm For Fast Subspace Identification

**Input:** Empty list \( Task\_list \) to be scheduled.

**Output:** Scheduling solution of the \( Task\_list \).

1: Put \( task_i \) with the resource request \( r_i \) into \( Task\_list \).
2: Detect the load utilization of \( N \) nodes \( U_1, U_2, \ldots, U_N \).
3: for \( i = 1 \) to \( num(Task\_list) \) do
4: Schedule \( task_i \) to the node \( j \) with the lowest load utilization.
5: Update the load utilization of node \( j, U_j = U_j + r_i \).
6: Remove the task from \( Task\_list \).
7: end for
8: Create the containers in the target nodes.

Fig. 7. The SID workflow structure tested in the first group (MPT = 2).

Fig. 8. The relationship between speedup and workflow granularity.
TABLE V
RECORDED RESULTS STATISTICS FOR TOPICAL PARAMETERS IN MONTE CARLO EXPERIMENTS

| Parameters | Middle-scale | Large-scale |
|------------|--------------|-------------|
|            | \( N = 10 \) | \( N = 20 \) | \( N = 10 \) | \( N = 20 \) | \( N = 50 \) | \( N = 50 \) |
| \( j = 1,000 \) | \( j = 1,000 \) | \( j = 10,000 \) | \( j = 10,000 \) | \( j = 10,000 \) | \( j = 20,000 \) |
| Baseline | Time (s) | 0.0114 | 0.0164 | 0.4867 | 0.7527 | 3.1082 | 13.8369 |
| Recursive Method 1: R4SID | Time (s) | 0.1233 | 0.1765 | 1.9114 | 1.4675 | 5.8254 | 11.3162 |
| Recursive Method 2: RPBSSID | Time (s) | 0.1102 | 0.1278 | 1.0965 | 1.2308 | 4.0482 | 7.9165 |

| Group 1: | MPT = 2 | Tasks = 5 |
|-----------|---------|----------|
| 4 CPU | 8 GB | 0.0114 | 0.164 |
| 8 CPU | 8 GB | 0.0122 | 0.0163 |
| Group 2: | MPT = 5 | Tasks = 20 |
| 4 CPU | 8 GB | 0.0394 | 0.1724 |
| 8 CPU | 8 GB | 0.1019 | 0.1233 |
| Group 3: | MPT = 5 | Tasks = 20 |
| 4 CPU | 8 GB | 0.0533 | 0.0663 |
| 8 CPU | 8 GB | 0.0823 | 0.0747 |

TABLE VI
RESOURCES AND NODE CLUSTER INFORMATION

| Experiment Group | Nodes | Total Resource Pool | Node Type |
|------------------|-------|---------------------|-----------|
| Baseline         | 1     | 4 CPU, 8 GB         | ecc.hfc6.slarge |
| Recursive        | 1     | 4 CPU, 8 GB         | ecc.hfc6.slarge |
| Group 1          | 2     | 32 CPU, 64 GB      | ecc.hfc6.4slarge |
| Group 2          | 2     | 32 CPU, 64 GB      | ecc.hfc6.4slarge |
| Group 3          | 4     | 64 CPU, 128 GB     | ecc.hfc6.4slarge |

Fig. 9. The relationship between computation time and resource amount.

Fig. 10. The relationship between speedup and container configuration.

structure with the MPT = 10 significantly decreased the computation speed. This is because the data amount processed in this scale is relatively small, and more time is costed in communication when applying the second structure with much more channels. For the large-scale parameters, the data amount processed in each container is larger. Thus, the high-MPT structure brought significant improvement. This case states that the computation efficiency could be improved if applying the workflow structure with proper granularity.

2) Resource Amount: The relationship between the computation time and total resource amount is shown in Fig. 9. It is clear that all the computation times are reduced with the total resource amount rising. For example, the baseline is 18,1912 s when \( N = 50, j = 20,000 \). The computation time is reduced to 2.7692 s, and the reduction percentage is 84.8% when the cloud resource pool has 32 CPU and 64 GB memory. The computation time is further reduced to 1.7625 s, and the reduction percentage is 89.6% when the total resources are 64 CPU and 128 GB memory. This case states that a high computation speed could be achieved if enough computing resources are provided.

3) Container Configuration: The comparison experiments are conducted with different container configurations for the three groups. Since the SVD is the main time-consuming stage, the containers processing the SVD tasks are set as 8 CPU and 8 GB memory. However, the speedup improvement is limited, as shown in Fig. 10. In the first two bar charts, the speedups improved from 3.7810 to 4.2119 and from 3.2952 to 3.7908, respectively. Since there are only two SVD tasks in the first group, the resources are relatively unstrained. Therefore, the upgrade of container configuration could lead to some performance improvement. But in the latter two bar charts, the speedups remained nearly unchanged. This is because there were 17 SVD tasks in the third group, and the resources became strained. When the containers are created and run in a cloud resource pool at the same time, the competition for resources occurs, and the containers affect each other. This case states that the improvement of upgrading container configuration depends on whether the current resources are sufficient.
4) Relationship Between Data Scales and Computation Time: The relationship between data scale and cost time is presented in Fig. 11 and Fig. 12. We analyzed the features of the baseline and three groups, in which each container has 4 CPU and 8 GB memory. The data scale ranges from $N = 10$, $j = 1000$ to $N = 50$, $j = 20000$, where the former two belong to the middle scale and the latter four fall into the large scale. From Fig. 11, we find the computation time increases with the growth of the data scale. That holds true for all the groups.

In more detail, when the data scale reaches $N = 50$, $j = 10000$, the computation time experiences a sharp rise for the baseline and group 1 with MPT= 2. In comparison, the increases in computation times for group 2 and group 3 are slower, as they adopted larger workflow with MPT= 10. From Fig. 12, i.e., the relationship between the data scale and the logarithm of computation, it is clear that in the middle-scale stage, the workflow with a smaller MPT accelerates the SID mission, while group 2 and group 3 require more times. In summary, it indicates that the computation time is primarily dependent on the data scale. As the data scale increases, the time cost also rises. To achieve higher computation efficiency, users can choose the workflow with an appropriate MPT, as stated in Section VI-D1.

E. Comparison With the Recursive Methods

The recursive SID methods are the typical accelerated SID methods based on the improvement of the numerical algorithm design. Thus the recursive methods are used to compare the computational efficiency of the traditional numerical improvement and cloud-based method. Provided the parameters of middle- and large-scales, the recursive methods are also conducted in the same environment with the baseline.

The results of the Monte Carlo experiments are recorded in Table. V. In the middle-scale parameters, the recursive methods decrease the computation speed of the SID missions, of which the computation times are about several times that of the baseline. This is because the data scale is relatively small, and the iteration numbers of the recursive method are high, which lead to the waste of the computing resources. As the comparison, the cloud-based SID method with the workflow structure of Fig. 7 obtains better results, which can satisfy the real-time requirements. This case states that the cloud-based method could achieve faster computation speed for the real-time mission by designing the workflow structure with proper granularity.

In the large-scale parameters, the recursive methods reduce the computation time of the SID mission in most cases. The reduction percentages are from 35% to 41% and 40% to 57%. But the computation speed is sensitive to the length-width ratio of the matrix. For the R4SID, when $N = 50$, $j = 10,000$, i.e., the length-width ratio becomes larger, the computation time increases 31.5%. For the RPBSID$_{pm}$, the reduction percentage of the computation time is only 8.6%. As the comparison, the computation times are reduced by at most 91.6% by the cloud-based SID method. This case states that the proposed method could significantly improve the upper bound of the computation speed by cloud computing.

VII. Conclusion

This paper has presented a cloud-based method of fast SID combining the workflow processing approach and container technology, which could reduce the computation time by at most 91.6% for the large-scale SID missions and satisfy the real-time requirement for the parameter of SID-based data-driven predictive control. The proposed method can well deal with the large-scale and real-time identification missions, which are the main difficulties of the current SID methods.

In the future, we think there are three research directions that hold potential value. Firstly, since the cloud service providers are considered as ‘semi-honest’, the privacy protection issue become imperative for high-value missions, and the study of differential-privacy-based and homomorphic-encryption-based SID methods should be pursued. Secondly, in this work, container configuration is selected in discrete fixed set, which might be the optimal strategy. The co-design of cloud resource utilization cost and computation efficiency should be studied by means of container configuration auto-scaling. Thirdly, control and system identification are closed related areas. In control applications, there are also some advanced algorithms such as model predictive control and data-driven predictive control, which involve computationally intensive tasks. Therefore, we will also explore real-time control methods based on cloud workflow.

APPENDIX A
DEFINITIONS IN N4SID ALGORITHM

In what follows, the matrices $A \in \mathbb{R}^{p \times j}$ and $B \in \mathbb{R}^{q \times j}$.
**Definition 1: Orthogonal projection.**
The orthogonal projection of the row space of $A$ into the row space of $B$ is denoted by $A/B$ and defined as:

$$A/B = AB^\dagger B$$  \hspace{1cm} (13)$$
where $\dagger$ means the Moore-Penrose pseudo-inverse.

$A/B^\perp$ is the projection of the row space of $A$ into $B^\perp$ where $B^\perp$ represents the orthogonal complement of the row space of $B$, for which we have $A/B^\perp = A - A/B$.

**Definition 2: Oblique projection.**
The oblique projection of the row space of $A$ along the row space of $B$ into the row space $C \in \mathbb{R}^{r \times j}$ is denoted as:

$$A/B C = (A/B^\perp)(C/B^\perp) C.$$  \hspace{1cm} (14)$$

$U_p$ and $U_f$ are defined as:

$$U_p^+ = \begin{bmatrix} u(0) & u(1) & \ldots & u(j-1) \\ u(1) & u(2) & \ldots & u(j) \\ \vdots & \vdots & \ddots & \vdots \\ u(N) & u(N+1) & \ldots & u(N+j-1) \end{bmatrix},$$  \hspace{1cm} (15)$$

$$U_f^+ = \begin{bmatrix} u(N+1) & u(N+2) & \ldots & u(N+j) \\ u(N+2) & u(N+3) & \ldots & u(N+j+1) \\ \vdots & \vdots & \ddots & \vdots \\ u(2N-1) & u(2N) & \ldots & u(2N+j-2) \end{bmatrix}.$$  \hspace{1cm} (16)$$

where $U_p^+$ has one more vector row than $U_f$ and $U_f^+$ has one less vector row than $U_f$. $Y_p^+$ and $Y_f^+$ are denoted similarly.

Then, $W_p^+$ is defined as:

$$W_p^+ = \begin{bmatrix} Y_p^+ \\ U_p^+ \end{bmatrix}.$$  \hspace{1cm} (17)$$

In step 5, $Y_i$ and $U_i$ are defined as:

$$Y_i = \begin{bmatrix} y(N) & y(N+1) & \ldots & y(N+j-1) \end{bmatrix},$$  \hspace{1cm} (18)$$

$$U_i = \begin{bmatrix} u(N) & u(N+1) & \ldots & u(N+j-1) \end{bmatrix}.$$  \hspace{1cm} (19)$$

**APPENDIX B**

**STEP 4 IN N4SID ALGORITHM**

The extended matrices $\Gamma_i^+$ and $\Gamma_{i-1}^+$ and estimated state sequences $X_i$ and $X_{i+1}$ are determined as:

$$\Gamma_i^+ = W_i^{-1}U_iS_i^{1/2}, \Gamma_{i-1}^+ = \Gamma_i^0, X_i^0 = \Gamma_i^0C_i, X_{i+1} = \Gamma_{i-1}C_{i-1},$$

where $\Gamma_i^0$ means $\Gamma_i$ without the last block row.

**REFERENCES**

[1] P. Van Overschee and B. De Moor, *Subspace Identification for Linear Systems: Theory, Implementation, Applications*. Norwell, MA, USA: Kluwer, 1996.

[2] T. Wu, V. M. Venkatasubramanian, and A. Pothen, “Fast parallel stochastic subspace algorithms for large-scale ambient oscillation monitoring,” *IEEE Trans. Smart Grid*, vol. 8, no. 3, pp. 1944–1953, May 2017.

[3] Y. Xia, W. Xie, B. Liu, and X. Wang, “Data-driven predictive control for networked control systems,” *Inf. Sci.*, vol. 235, pp. 45–54, Jun. 2013.

[4] P. Van Overschee and B. De Moor, “N4SID: Subspace algorithms for the identification of combined deterministic-stochastic systems,” *Automatica*, vol. 30, no. 1, pp. 75–93, Jan. 1994.

[5] C. Yu, J. Chen, and M. Verhaegen, “Subspace identification of individual systems in a large-scale heterogeneous network,” *Automatica*, vol. 109, Nov. 2019, Art. no. 108517.

[6] H. Oku and H. Kimura, “Recursive 4SID algorithms using gradient type subspace tracking,” *Automatica*, vol. 38, no. 6, pp. 1035–1043, Jun. 2002.

[7] I. Houtzager, J.-W. van Wingerden, and M. Verhaegen, “Recursive predictor-based subspace identification with application to the real-time closed-loop tracking of flutter,” *IEEE Trans. Control Syst. Technol.*, vol. 20, no. 4, pp. 934–949, Jul. 2012.

[8] E. Dahler and L. Mevel, “Fast and efficient state and order subspace identification,” *IFAC Proc. Volumes*, vol. 44, no. 1, pp. 6523–6528, Jan. 2011.

[9] M. Dahler and L. Mevel, “Efficient multi-order uncertainty computation for stochastic subspace identification,” *Mech. Syst. Signal Process.*, vol. 38, no. 2, pp. 346–366, Jul. 2013.

[10] G. León, C. González, R. Mayo, D. Mozos, and E. S. Quintana-Ortí, “Noise estimation for hyperspectral subspace identification on FPGAs,” *J. Supercomput.*, vol. 75, pp. 1323–1335, May 2018.

[11] P. K. Senyo, E. Addae, and R. Bouteng, “Cloud computing research: A review of research themes, frameworks, methods and future research directions,” *Int. J. Inf. Manag.*, vol. 38, no. 1, pp. 128–139, Feb. 2018.

[12] J. Chen et al., “A parallel random forest algorithm for big data in a spark cloud computing environment,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 919–933, Apr. 2017.

[13] G. Xiao, J. Li, Y. Chen, and K. Li, “MaFCIS: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks,” *J. Parallel Distrib. Comput.*, vol. 141, pp. 49–58, Jul. 2020.

[14] Y. Xia, R. Gao, M. Lin, Y. Ren, and C. Yan, “Green energy complementary based on intelligent power plant cloud control system,” *Acta Automatica Sinica*, vol. 46, no. 9, pp. 1844–1868, Sep. 2020.

[15] P. Zhang and M. Zhou, “Dynamic cloud task scheduling based on a two-stage strategy,” *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 772–783, Apr. 2018.

[16] H. Yuan and M. Zhou, “Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems,” *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 3, pp. 1277–1287, Jul. 2021.

[17] F. Wu, Q. Wu, and Y. Tan, “Workflow scheduling in cloud: A survey,” *J. Supercomput.*, vol. 71, no. 9, pp. 3373–3418, May 2015.

[18] L. Zhang, K. Li, C. Li, and K. Li, “Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems,” *Inf. Sci.*, vol. 379, pp. 241–256, Feb. 2017.

[19] J. Zhu, X. Li, R. Ruiz, and X. Xu, “Scheduling stochastic multi-stage jobs to elastic hybrid cloud resources,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1401–1415, Jun. 2018.

[20] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, “Learning scheduling algorithms for data processing clusters,” in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 270–288.

[21] L. Ye, Y. Xia, S. Tao, C. Yan, R. Gao, and Y. Zhan, “Reliability-aware and energy-efficient workflow scheduling in IaaS clouds,” *IEEE Trans. Autom. Sci. Eng.*, vol. 20, no. 3, pp. 2156–2169, Jul. 2023.

[22] L. Yang, Y. Xia, X. Zhang, L. Ye, and Y. Zhan, “Classification-based differentiable workflows scheduling in clouds,” *IEEE Trans. Autom. Sci. Eng.*, early access, Nov. 1, 2022, doi: 10.1109/TASE.2022.3217666.

[23] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, “Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers,” *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 48–56, Jun. 2017.

[24] R. Ranjan, I. S. Thakur, G. S. Aujla, N. Kumar, and A. Y. Zomaya, “Energy-efficient workflow scheduling using container-based virtualization in software-defined data centers,” *IEEE Trans. Ind. Informat.*, vol. 16, no. 12, pp. 7646–7657, Dec. 2020.

[25] T. Goldschmidt, S. Hauck-Stattelmann, S. Malakuti, and S. Grüner, “Container-based architecture for flexible industrial control applications,” *Proc. IEEE*, vol. 136, no. 6, pp. 28–36, Mar. 2018.

[26] J. Mellado and F. Núñez, “A container-based IoT-oriented programmable logical controller,” in *Proc. IEEE Conf. Ind. Cyberphysical Syst.*, vol. 1, Jun. 2020, pp. 55–61.

[27] Á. Björck, *Numerical Methods in Matrix Computations*. New York, NY, USA: Springer, 2015.

[28] S. Gres, K. E. Tatis, V. Dertimanis, and E. Chatzi, “Low-rank approximation of Hankel matrices in denoising applications for statistical damage diagnosis of wind turbine blades,” *Mech. Syst. Signal Process.*, vol. 197, Aug. 2023, Art. no. 110391.
[29] R. Maulik and G. Mengaldo, “PyParSVD: A streaming, distributed and randomized singular-value-decomposition library,” in Proc. 7th Int. Workshop Data Anal. Reduction Big Sci. Data (DRBSD-7), Nov. 2021, pp. 19–25.

[30] A. Lario, R. Maulik, O. T. Schmidt, G. Rozza, and G. Mengaldo, “Neural-network learning of SPOD latent dynamics,” J. Comput. Phys., vol. 468, Nov. 2022, Art. no. 111475.

[31] J. Nelson, Mastering Redis, Birmingham, U.K.: Packt, 2016.

[32] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, “Uncertainty-aware online scheduling for real-time workflows in cloud service environment,” IEEE Trans. Services Comput., vol. 14, no. 4, pp. 1167–1178, Jul. 2021.

[33] G. Sayfan, Mastering Kubernetes, Birmingham, U.K.: Packt, 2017.

[34] Y. Mo and R. M. Murray, “Privacy preserving average consensus,” IEEE Trans. Auton. Control, vol. 62, no. 2, pp. 753–765, Feb. 2017.

[35] T. Tanaka, M. Skoglund, H. Sandberg, and K. H. Johansson, “Directed information and privacy loss in cloud-based control,” in Proc. Amer. Control Conf., May 2017, pp. 1666–1672.

[36] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Research on homomorphic encryption for arithmetic of approximate numbers,” in Proc. Int. Conf. Intell. Syst. Commun., IoT Secur. (ICISC015), Cham, Switzerland, Feb. 2023, pp. 409–437.

[37] A. B. Alexandru, K. Gatsis, Y. Shoukry, S. A. Seshia, P. Tabuada, and G. J. Pappas, “Cloud-based quadratic optimization with partially homomorphic encryption,” IEEE Trans. Auton. Control, vol. 66, no. 5, pp. 2357–2364, May 2021.

[38] J. Nilsson, “Real-time control systems with delays,” Ph.D. thesis, Dept. Autom. Control, Lund Inst. Technol., Lund, Sweden, 1998.

Runze Gao received the B.S. degree from the Beijing Institute of Technology, China, in 2017, where he is currently pursuing the Ph.D. degree in control science and engineering. His research interests include cloud control systems, model predictive control, and data-driven predictive control.

Yuanqing Xia (Senior Member, IEEE) received the M.S. degree in fundamental mathematics from Anhui University, Hefei, China, in 1998, and the Ph.D. degree in control theory and control engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001. From 2002 to 2003, he was a Post-Doctoral Research Associate with the Institute of Systems Science, Academy of Mathematics and System Sciences, Chinese Academy of Sciences, Beijing. From 2004 to 2006, he was a Research Fellow with the University of Glamorgan, Pontypridd, U.K. From 2007 to 2008, he was a Guest Professor with the Medical University of Innsbruck, Innsbruck, Austria. He is currently a Professor with the School of Automation, Beijing Institute of Technology, Beijing. His research interests include cloud control systems, networked control systems, robust control and signal processing, active disturbance rejection control, unmanned system control, and flight control.

Guan Wang was born in 1986. He received the M.S. degree in CTS from the University of Jinan, China, in 2014. He currently pursuing the Ph.D. degree with the School of Automation, Beijing Institute of Technology, China. He was a Lecturer with the School of Information Science and Engineering, University of Zhaozhuang, China, in 2019. His research interests include networking systems, cloud computing, gene expression data, and machine learning.

Liwen Yang received the Ph.D. degree from the Beijing Institute of Technology, Beijing, China, in 2018. Prior to join BIT, he was a Post-Doctoral Fellow with the Department of Computing, The Hong Kong Polytechnic University. He is currently an Assistant Professor with the School of Automation, BIT. His research interests include networking systems, game theory, and machine learning.

Yufeng Zhan received the Ph.D. degree from the Beijing Institute of Technology (BIT), Beijing, China, in 2018. Prior to join BIT, he was a Post-Doctoral Fellow with the Department of Computing, The Hong Kong Polytechnic University. He is currently an Assistant Professor with the School of Automation, BIT. His research interests include networking systems, game theory, and machine learning.