SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

August 2024, Vol. 67, Iss. 8, 182201:1–182201:18
https://doi.org/10.1007/s11432-023-4017-0

# ControlService: a containerized solution for control-algorithm-as-a-service in cloud control systems

Chenggang SHAN[1,2], Runze GAO[2], Zhen YANG[1], Wei ZHANG[1] & Yuanqing XIA[2*]

[1]*School of Artificial Intelligence, Zaozhuang University, Zaozhuang* 277100, *China;*
[2]*School of Automation, Beijing Institute of Technology, Beijing* 100081, *China*

Received 14 September 2023/Revised 25 January 2024/Accepted 13 February 2024/Published online 27 June 2024

**Abstract** As an extension of networked control systems, cloud control systems (CCSs) have emerged as a new control paradigm to improve the service quality of emerging control missions, such as data-driven modeling and automated vehicles. Existing studies have used the workflow-based restructured method to optimize the computation-intensive algorithms in the CCSs. However, the challenges here are how to define and submit these algorithms' workflows as cloud services and execute these algorithms' workflows in a containerized manner. Based on these challenges, we propose a containerized solution for the control-algorithm-as-a-service (C3aS) in the CCSs, namely ControlService. It offers the control algorithm as a cloud workflow service and uses a customized workflow engine to realize the containerized execution. First, we employ a cloud workflow representation method to define a control algorithm into an abstract cloud workflow form. Afterward, we provide a cloud service representation of the abstract cloud workflow. Next, we design a workflow engine and submit the cloud service to this workflow engine to implement containerized execution of this cloud service in the CCSs. In the experiment, we discuss the cloud service form and containerized implementation of the subspace identification method. Experimental results show that the proposed ControlService has significant performance advantages in computational time, reduction percentage, and speedup ratio compared with the baseline method.

**Keywords** cloud control systems, control-algorithm-as-a-service, containerization, workflow engine, cloud workflow service, subspace identification

## 1 Introduction

In recent years, the networked control systems (NCSs) [1] have played a key role in the field of the Internet of Things (IoTs) and have been widely used and studied in practice [2,3]. However, with the proliferation of IoT devices, the capacity, variety, and velocity properties of big data from IoT bring a huge network communication burden and computing burden to the control system, and the traditional NCSs have been unable to meet the requirements of high quality and real-time performance of the control system [4].

In this context, a new paradigm of NCSs named cloud control systems (CCSs) was proposed and further developed [5]. It uses the cloud's supercomputing power and scalability to do complex data collection, storage, and computation-intensive task processing to reduce the communication and computing burden encountered by traditional NCSs. Herein, all computing missions of the control system are aggregated to the cloud through the CCSs and processed as cloud services. This service form will make it easier, faster, and cheaper for users to deploy, maintain, and update the control behaviors of their control systems.

Control-as-a-service (CaaS), derived from [6], was later extended to automotive domain [7], robotics [8, 9], industrial control [10], etc. CaaS aims to take advantage of cloud computing capabilities to offer control algorithms or control behaviors as cloud services to achieve optimal control. The control algorithm is an integral part of CaaS, and control-algorithm-as-a-service (C3aS) is a key step in implementing CaaS. This paper focuses on C3aS, aiming to study the containerized solution of C3aS in CCSs.

* Corresponding author (email: xia_yuanqing@bit.edu.cn)

 

There are mainly the following two key challenges facing the implementation of C3aS. One challenge is how to describe the control algorithms and how to submit the control algorithms to CCSs as cloud services. Another challenge is how to implement control algorithms as cloud service instances in a containerized way running on the distributed CCSs infrastructure. With these challenges, we see great potential in cloud workflow representation methods for control algorithms [11,12]. The scalability of cloud computing resources and the parallelism of workflow topology provided by the cloud workflow representation method meet the requirements of complex and intensive computation of control algorithms. In this paper, we specify the workflow data structure of control algorithms and use JSON[1)] script to define the workflow task steps and then define Yaml[2)] submission file for control workflow service suitable for the Kubernetes[3)] container orchestration system. In addition, a Kubernetes-based control workflow engine is designed to implement control algorithms as service instances, responsible for the management, scheduling, and execution of task containers of control algorithm workflows.

Currently, some studies have focused on the practical applications of CCSs, such as unmanned surface vehicles (USVs) [13], robotics [14], and intelligent and connected vehicles (ICV) [15]. Nevertheless, these studies do not involve the concept of C3aS and the containerized solution for control service. There are also some studies that study execution efficiency for control algorithms in the form of workflows under the framework of CCSs, such as subspace identification (SID) [11], data-driven predictive control (DPC) [12,16]. However, these studies cannot provide an automated cloud workflow management engine suitable for the Kubernetes container orchestration system.

Based on our former studies [17, 18], this paper proposes a novel containerization solution for C3aS in CCSs, namely ControlService, which supports the control algorithm in the form of cloud workflow service and uses the customized control workflow engine to realize the containerized execution of the cloud workflow service. First, we employ a workflow-based method to reconstruct the control algorithm, specify the workflow topology of the control algorithm, decouple the control algorithm following its workflow topology, and present the mirrored forms of control workflow tasks. Second, we use a JSON-formatted script to define the workflow task steps and present the Yaml file for the submission of the control algorithm service. Next, we design and implement a control workflow engine that containerizes the control algorithm workflow as a service instance in a cloud environment. The above execution steps constitute our proposed containerization solution for C3aS. We use ControlService to conduct extensive experiments around computation-intensive algorithms like SID. Compared with the Baseline, experimental results show that our ControlService has significant performance advantages in computational time, reduction percentage, and speedup ratio. To our knowledge, this is the first practical exploration of C3aS in the CCSs field. The control workflow engine at the platform-as-a-service (PaaS) layer proposed in this paper will be an automated software platform for the practice of CCSs. The main contributions of this paper are summarized as follows:

(1) Present the overview of ControlService and elaborate on the CCS architecture based on the C3aS.

(2) Present a JSON-based definition method of control algorithm workflow and a submission method of cloud service instances of control algorithms.

(3) Implement a control workflow engine for cloud infrastructure and elaborate its architecture design, core components, and working principles.

(4) Present the cloud workflow service form for the SID algorithm based on the ControlService solution. Experimental results verify the effectiveness of ControlService after submitting the SID algorithm as a cloud workflow service.

The remainder of the paper is organized as follows. Section 2 introduces related work. Section 3 elaborates on the containerized solution for C3aS, while Section 4 describes workflow construction for the SID algorithm. Numerical experiments have been implemented to verify the effectiveness of our proposed ControlService solution in Section 5. Finally, Section 6 concludes this paper.

We have open-sourced the proposed solution. The source code is publicly available on GitHub[4)].

1) JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write and for machines to parse and generate.

2) Yaml is a data serialization language commonly used to write configuration files.

3) Production-grade container orchestration. https://kubernetes.io.

4) https://github.com/CloudControlSystems/ControlService.

## 2 Related work

With the development of CCSs, the concept of CaaS has appeared in various applications. Esen et al. [6] presented the CaaS concept and provided architectural design and proof-of-concept implementation for a cloud service-based throttle limitation scenario in the automotive domain. The proposed software architecture enables the cloud backend to control the concept car by CAN-to-WiFi gateway. Ref. [10] introduced a new architecture for industrial control systems to transfer and distribute control tasks from dedicated controllers to network switches. The architecture uses end-to-end deterministic technology and a joint task and traffic scheduling algorithm to combine control and transmission functions in network switches. The work reported in [7] proposed a multi-layer CaaS architecture to connect physical and logical objects and integrate basic control patterns to organize a consistent orchestration of control services. The integration of context managers and system environment analysis aims to monitor and orchestrate the deployment of context-aware control services in time to realize adaptive control systems. Bezerra et al. [19] proposed a fog computing-based control as a service architecture to dynamically process events in IoT and intelligent environments, meeting the control requirements with globality, adaptability, responsiveness, availability, and non-conflict. From this point of view, CaaS is a common form of the CCS to provide users with control capabilities via cloud services. However, these studies focus on service architecture, planning, and control optimization and do not involve a containerized solution of the service instance in the cloud. To keep pace with cloud native[5)], we consider new service forms for control algorithms and container-based practical solutions for control algorithm services.

Workflow is an abstraction and automation of a business process that defines tasks with interconnected dependencies and executes tasks by leveraging computing resources [20]. The workflow parallel structure is suitable for distributed cloud computing infrastructure architecture, which can support parallel running of control algorithms on the cloud. Gao et al. [11] designed a workflow establishment method of SID to match the distributed cloud environment and built a containerized cloud workflow processing system to run the logic- and data-dependent SID workflows. Ref. [12] presented a construction method for DPC workflow and proposed a novel cloud-edge collaborative containerized workflow-based DPC system with a disturbance observer to improve the computation efficiency and control effects. Dai et al. [16] proposed a novel computational data-enabled DPC algorithm in a cloud environment to solve a large-scale online optimization problem with high dimensional decision variables. Subsequently, Dai et al. [21] devised a cloud-based computational model predictive control (MPC) using a parallel multi-block alternating direction method of multipliers (ADMM) algorithm. The new parallel multi-block ADMM algorithm follows the workflow design idea and is specially designed for the non-convex computing problem with nonlinear constraints during the vehicle charging schedules. Herein, cloud-based control architecture and workflow-based control actions are designed to construct and deploy the proposed controllers as services in the cloud. The above studies adopt the cloud control architecture and workflow-based algorithm reconstructed method to improve computational efficiency and quality of control algorithms and also present the experimental system for cloud infrastructure. However, these reconstructed control algorithms based on workflow structure depend too much on customized experimental systems, in which the universality and agility restrict the platform transplantation and promotion of control services.

To summarize the related work, we can conclude that the concept of CaaS has integrated into cloud computing and control technology, and control algorithms for workflow structures suitable for distributed cloud environments have also emerged. However, the bottleneck here is the absence of a C3aS containerization solution for CCSs, which contains a set of definitions and submission methods for cloud services of control algorithms and an automated control workflow management engine. The latter can manage, schedule, and run containerized control algorithm workflow in the cloud infrastructure based on the Kubernetes platform. The aim of this paper is to provide a complete C3aS containerization solution for the cloud control field with automation, agility, and generality.

## 3 Containerized solution for C3aS

This section first presents the overview of ControlService for CCSs, followed by the cloud workflow service and an automated control workflow engine architecture for cloud scenarios.

5) Cloud native is a set of cloud technology product systems based on container, microservice, DevOps, and other technologies, and is the future development direction of cloud computing.

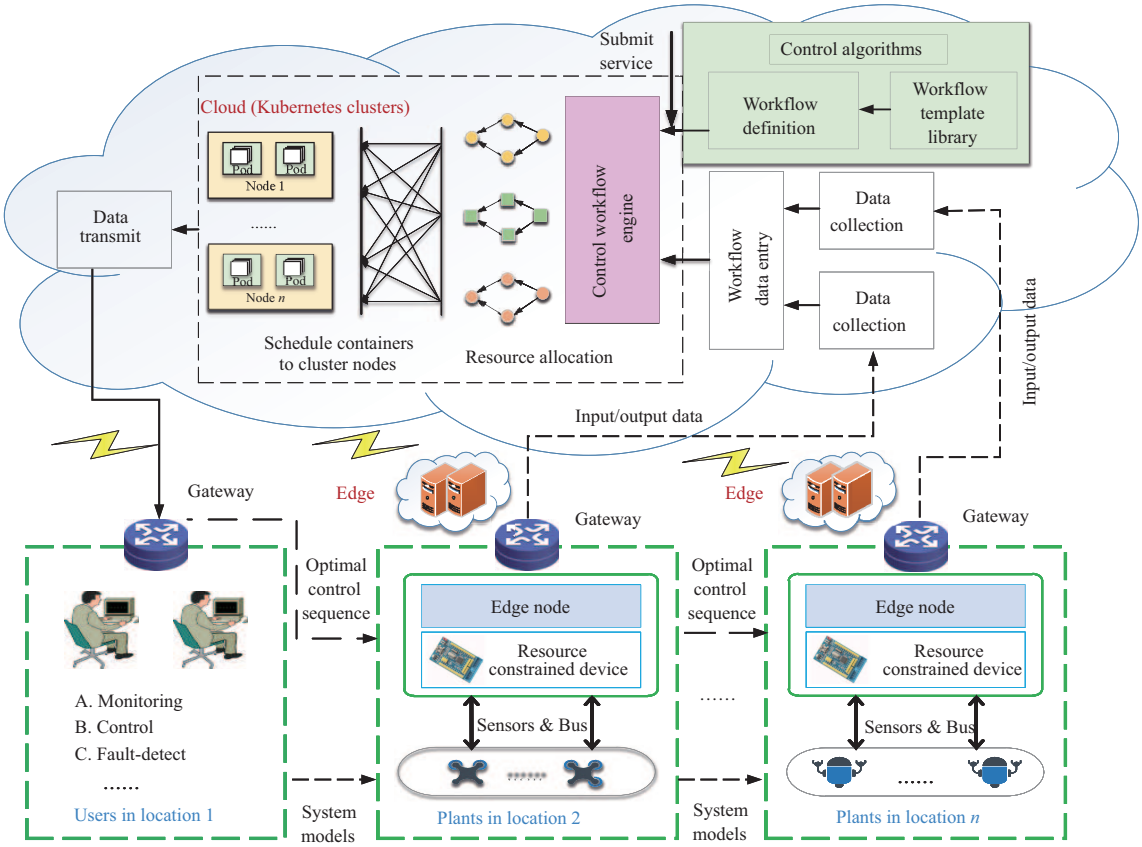


**Figure 1** (Color online) Overview of ControlService for CCSs.

### 3.1 Overview of ControlService

The ControlService for CCSs is illustrated in Figure 1. A complete containerization solution for C3aS consists of four parts: cloud service definition, cloud service submission, workflow scheduling, and workflow execution. Its core functionalities are as follows:

(1) Provide an interface to the public or private cloud in CCSs, allowing users to define and submit cloud services of related algorithms on demand.

(2) Implement the containerized execution of related algorithm workflows.

(3) Provide the ability of flexible deployment and uninstall capabilities for the proposed control workflow engine on the Kubernetes platform.

With the help of the Kubernetes platform, ControlService integrates cloud service definition, cloud service submission, containerized workflow scheduling, and workflow execution to form an automated containerized solution for C3aS.

The CCS architecture that ControlService relies on consists of four layers, including the cloud platform, edge, terminal, and user layers. Some core modules are described in Table 1. The cloud platform layer is located at the top of the overview. It takes charge of receiving the input and output data from the edge nodes, workflow definition, service submission, workflow containerization, and outputting system models or optimal control sequences to users or edge nodes. The data collection modules gather the collected data into the workflow data entry. At the same time, based on the workflow template library, the workflow definition module, data collection module, and control algorithm module together build the control workflow as a cloud service and submit this cloud service to the control workflow engine. Next, the control workflow engine employs workflow data, the received control workflow service, and the task images to implement workflow containerization. The cloud platform (elaborated in Subsection 3.3) provides the resource allocation and running environment for the control workflow engine. Finally, the system model or optimal control sequence is fed back to the user or edge node devices. With a small-scale computing capacity, the edge layer can handle local tasks with time-sensitive and private data, which better complements the cloud. The terminal layer is located at the bottom right of the overview and

**Table 1** Descriptions of core modules in Figure 1

| Module | Description |
|---|---|
| Cloud | An abstract concept for cluster resources comprising the public or private cloud |
| Edge nodes | Resource-limited computing devices in different edge scenarios |
| Workflow template library | A sharing warehouse for storing common control workflow task images and topologies |
| Workflow definition | Define control algorithm workflows via workflow templates |
| Control algorithms | Reconstructed control algorithms using the workflow method |
| Workflow data entry | A middle module to provide the input and output data for control algorithm workflows |
| Terminal layer | Multiple edge scenarios distributed in different geographic locations |
| User layer | An interactive interface of ControlService, responsible for monitoring and controlling the CCSs and cloud workflow service delivery |

consists of multiple edge nodes distributed in different geographical locations. Edge nodes are independent computing systems that manage end devices, while they are often limited by computing capacity, power consumption, and cost. Therefore, the control task is outsourced to the cloud when the edge computing capacity is insufficient. The user layer receives the optimized system model to meet various functional requirements, and the optimized control sequence is directly sent to the edge node.

### 3.2 Cloud workflow service

Cloud services refer to obtaining required services through the network in an on-demand and easily scalable way [22]. This service can be IT, software, internet-related technologies, etc. This paper mainly studies the implementation of the C3aS. Most control algorithms in CCSs are computation-intensive and require a lot of computing resources to ensure their performance. Through user interaction (command line or web-based), the control algorithm can run flexibly on the cloud infrastructure. Cloud services require control algorithms to be easy to define, deliver, and portable, which requires an abstract representation method independent of computing environments to describe control algorithms as services.

#### 3.2.1 *Cloud workflow representation*

New information technologies have expanded the coverage of workflow across various applications. The resource-independent workflow description method is suitable for porting control algorithms across cloud infrastructures and delivering them as cloud services. A workflow, often illustrated by a directed acyclic graph (DAG), defines the control of how tasks are coordinated and executed [23]. It consists of numerous tasks and precedence dependencies between these tasks. Herein, the computation tasks are represented as nodes, and the data- and control-flow dependencies between them are represented as edges [24]. These dependencies typically represent sharing files created by one task and consumed by its descendant tasks.

In the CCSs, based on numerical scientific calculation method and distributed optimization theory, the centralized control computing tasks are restructured to build a parallel workflow form matching the distributed cloud infrastructure, which can fully exploit the computing power of the cloud infrastructure, reduce the computing time of the control algorithm, and improve the overall control performance of the system [11]. This DAG-based declarative approach is prone to reconstructing the control algorithm into cloud workflow.

A DAG-based cloud workflow, with the topology of a single entry and single exit, is suitable for the graphical representation of sequential or parallel control algorithms. As shown in Figure 2, the mentioned cloud workflows are represented graphically by DAGs. In the cloud environment, workflow tasks would be scheduled to different processing nodes that encapsulate the required computing resources through container technology. The parallel structure of workflows makes them suitable for distributed cloud infrastructure.

#### 3.2.2 *Cloud workflow service definition*

The abstract JSON-formatted workflow description provides a resource-independent workflow description method. It captures all tasks that perform computation, the execution order of these tasks, task dependencies, the required inputs, the expected output, required resources, task images, and the required parameters. For the restructured control workflows, workflow tasks are packaged into containers through

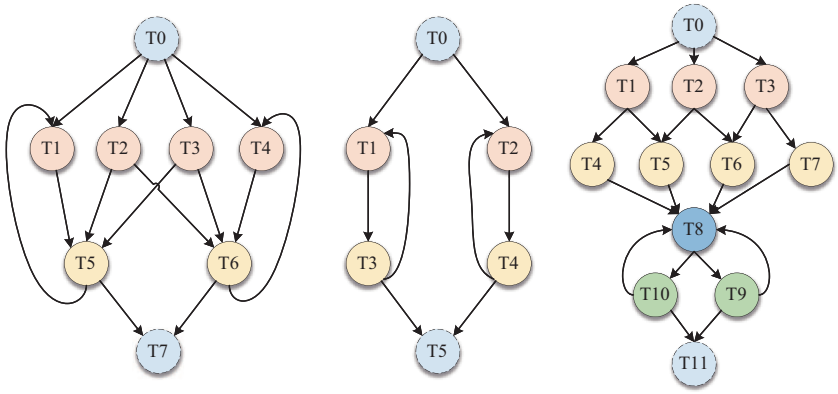


**Figure 2** (Color online) Cloud workflow example.

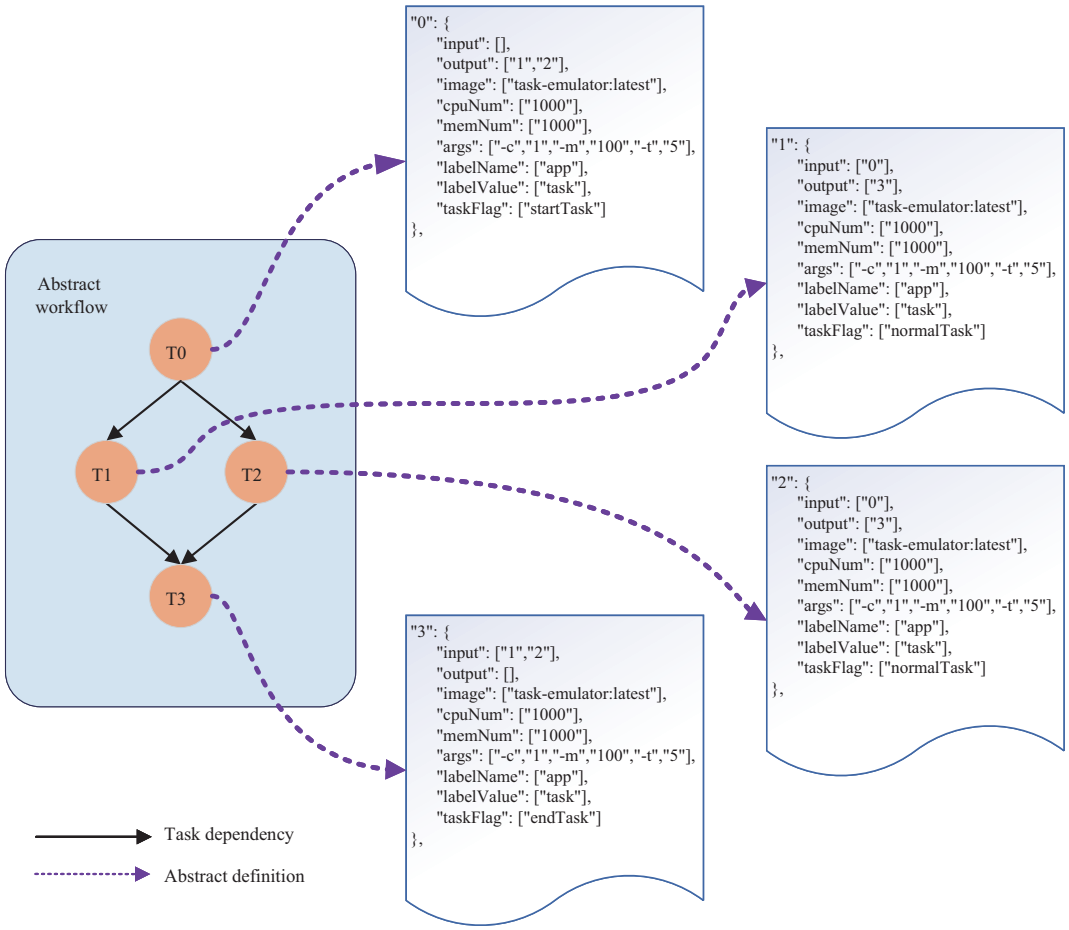


**Figure 3** (Color online) Abstract JSON-formatted description of a workflow with four task nodes.

the Docker[6)] technology and built in the form of an Image file stored in the local Harbor or remote Docker Hub repository.

Figure 3 shows the abstract JSON-formatted workflow definition with four task nodes. This JSON-formatted workflow definition follows the Key-value approach, with the task execution sequence number as the top-level Key and the task information as the top-level Value. Each task node works as a workflow step and contains `input`, `output`, `image`, `cpuNum`, `memNum`, `args`, `labelName`, `labelValue`, and `taskFlag`. The workflow runs in the order of tasks defined as DAG. These attributes of the task node are shown in

6) Docker is the most widely used open source container engine. It is an operating system level virtualization technology and a simple application packaging tool.

**Table 2** Description of task node attributes

| Attribute | Description |
|---|---|
| input | The input dependencies from the parent node |
| output | The output dependencies emitted to the descendant node |
| image | The task image address of the current task |
| cpuNum | The CPU resource requirements for this task container |
| memNum | The memory resource requirements for this task container |
| args | The required running parameters for this task container |
| labelName | The tag name of the pod that encapsulates this task container |
| labelValue | The tag value of the pod, combined with above tag name, forms a tag to match the specified node |
| taskFlag | Task identification, which specifies whether the current task is an entry, exit, or intermediate task |

Table 2.

As shown in Figure 3, the abstract JSON-formatted workflow description is readable and extremely simple. In addition, it can accommodate deeper workflows and provide better scalability. This flexible workflow definition method facilitates users to customize the control algorithm workflow and can work as an interface to connect with the front end.

### 3.2.3 *Cloud workflow service submission*

Figure 4 shows the process of workflow processing after submitting the cloud workflow service. We use the command line approach to submit cloud workflow service to control workflow engine running on the Kubernetes-based cloud infrastructure. The control workflow engine is our custom workflow management engine resided on a user-facing Kubernetes-based cloud infrastructure. The control workflow engine is to bridge the control algorithms and the CCSs infrastructure by mapping a user-defined high-level workflow description, an abstract workflow description named cloud workflow service, to an executable containerized workflow as shown in Figure 4.

Kubernetes employs its core components[7)] to manage cluster resources comprising one or more master nodes and several workload nodes. We populate the abstract JSON-formatted workflow definition into the ConfigMap[8)] file. Then the ConfigMap joins environment parameters to form a Yaml file that can be submitted as a cloud workflow service for deployment. A ConfigMap contains a complete abstract JSON-formatted workflow information and usually regards the editable JSON information as variable configurations. ConfigMap can decouple container image information from variable configurations to support the portability of workload pods[9)]. ConfigMap uses the pod's volume[10)] to mount the configuration file to a specified directory in the pod of the workflow injection module (Subsection 3.3). This process refers to the `Inject ConfigMap` in the leftmost part of Figure 4. The workflow injection module can parse the configuration file to obtain workflow information, equivalent to `Parse Json file` in the leftmost part of Figure 4. Note that the workflow injection module and the ConfigMap need to be in the same namespace.

As shown in Figure 4, we proceed to the workflow injection phase after submitting the workflow service. The parser of workflow injection module reads the ConfigMap file, parses it to get workflow information, and then packages the workflow for containerized workflow builder (Subsection 3.3) via gRPC[11)] communication.

### 3.2.4 *Workflow executing*

After the workflow reception is complete, the executable workflow is generated in a containerized way on cloud infrastructure. Workflow namespaces, persistent volume claim (PVC)[12)], and service[13)] need to be created one after another to prepare the workflow containerized execution. Herein, we only focus on the

7) Refer to the control plane's components, including Kube-apiserver, Etcd, Kube-scheduler, Kube-controller-manager, and cloud-controller-manager.

8) A ConfigMap is an API object to store non-confidential data in Key-value pairs. A ConfigMap allows you to decouple environment-specific configuration from your container images, making your application easy to port.

9) A Pod is an atomic scheduling unit of the Kubernetes cluster that can wrap one or more task containers.

10) Volume helps containers write data to local storage and enable file sharing between containers.

11) gRPC is a modern open-source high-performance remote procedure call (RPC) framework that can run in any environment.

12) A PVC is a request for storage by a user and consumes PV resources through a claim.

13) Service is an abstract way to expose an application running on a set of pods as a network service.

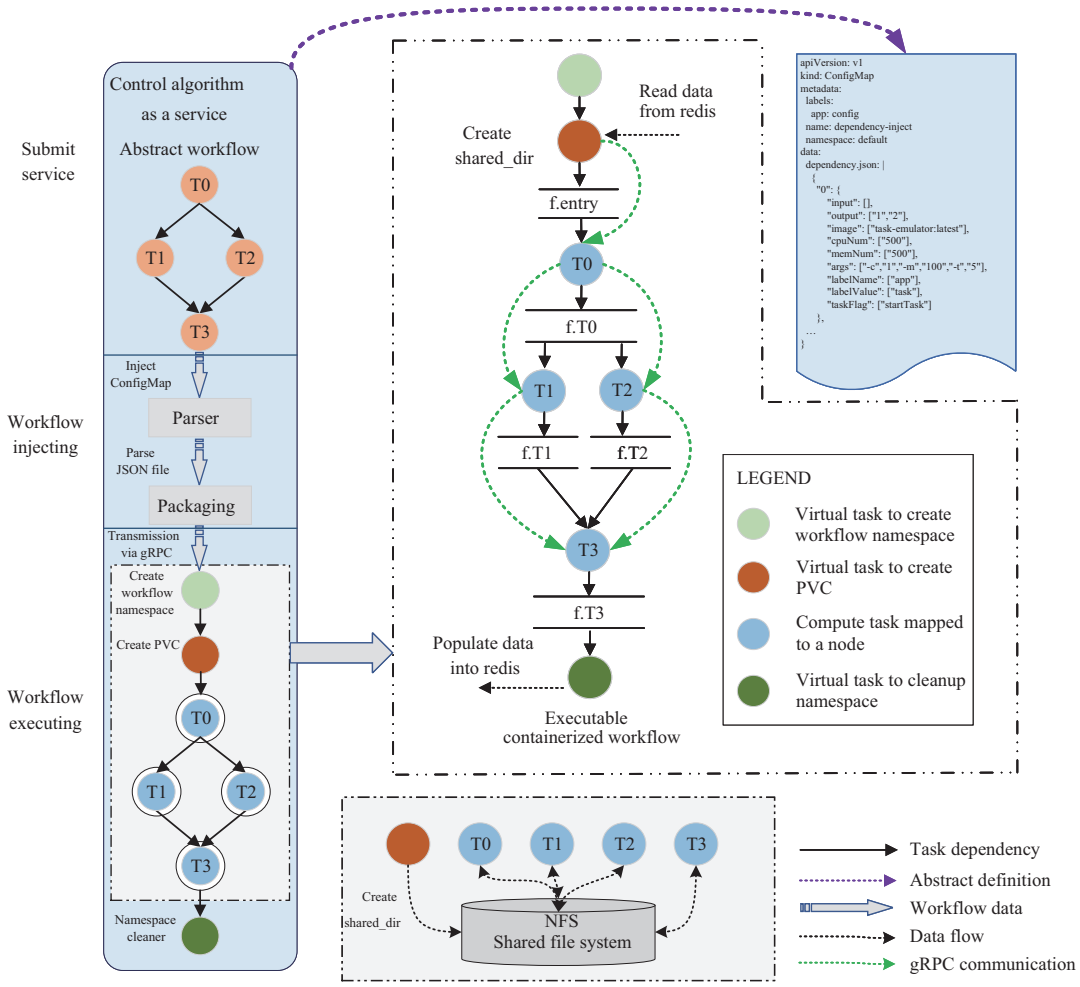


**Figure 4** (Color online) Data flow of a submitted abstract workflow service with an NFS sharing mechanism is running on the CCS infrastructure. The C3aS consists of three phases: service submission, workflow injection, and workflow execution. The control algorithm service is submitted via the Yaml file. The mapped abstract workflow is an executable containerized workflow, of which task pods are triggered between them by gRPC communication and an NFS sharing mechanism works to share intermediate data between tasks.

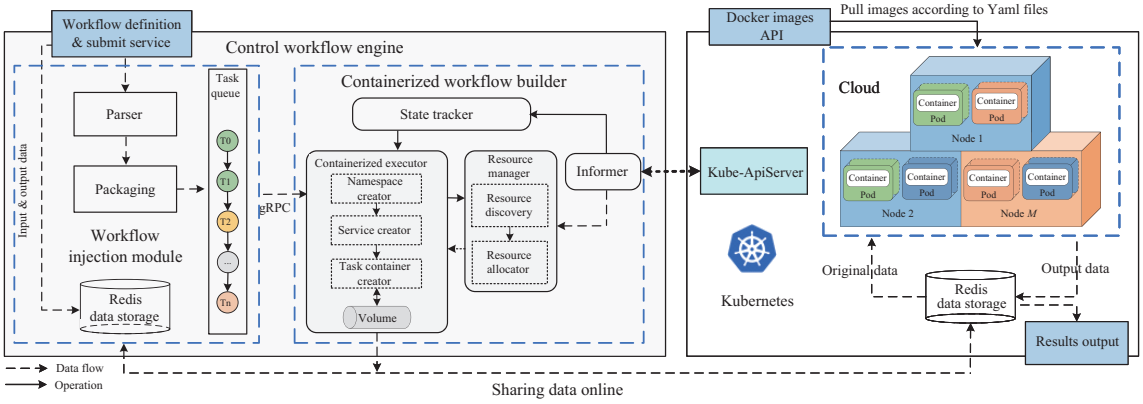


**Figure 5** (Color online) Architecture of control workflow engine for cloud scenario. The control workflow engine transforms the submitted abstract control algorithm workflow service into an executable containerized workflow that runs on the Kubernetes-based cloud infrastructure.

workflow execution process, data management between workflow tasks, and the service communication mechanism provided by Kubernetes. The other relevant modules are described in Figure 5.

**Data management during execution.** The Control workflow engine uses a network file system (NFS) on the Kubernetes cluster to store intermediate files. Workflow tasks wrapped into pods are scheduled to cluster nodes, and they exchange intermediate data between them in the form of files through NFS across the cluster. The NFS function implementation requires deploying the NFS business mounting pod and StorageClass service in advance. The PVC creation of a workflow requires StorageClass service to provide support. The created task pod binds its persistent volume (PV)[14] and PVC and implements data sharing via a mounted sharing directory. This directory is the `shared_dir`. As shown in the bottom part of Figure 4, multiple task pods in the same workflow namespace use the `shared_dir` in the NFS to exchange data.

For the executable containerized workflow in the middle part of Figure 4, the creations of namespaces and PVC are abstracted into two virtual entry tasks, accompanied by the generation of shared directory `shared_dir`. First, the virtual entry task accesses the Redis database to read input data and generates an output file identified by a logical filename `f.entry`. Then, the subsequent task $T_0$ takes an input file `f.entry` and generates a single output file identified by the logical name `f.T0`. Similarly, according to the dependencies between workflow tasks, we get the intermediate files such as `f.T1`, `f.T2`, and `f.T3` and the input-output relations between them, as shown in the middle part of Figure 4. Finally, the virtual exit task populates the output data into the Redis database and cleans up this workflow namespace.

**Service mechanism.** Service is created before each task pod. In a workflow namespace, each workflow task pod matches with a unique service, aiming to use gRPC communication to trigger task invocation via the ClusterIP[15] setting provided by the service. As shown in the middle part of Figure 4, the task invocations follow the task dependencies among workflow tasks.

### 3.3 Control workflow engine

As mentioned above, it is the control workflow engine's responsibility to not only translate abstract tasks to task pods to execute them but also to manage the executable containerized workflow, schedule task pods, and handle failures. As depicted in Figure 5, the control workflow engine consists of two main top-level entities, including a workflow injection module and a containerized workflow builder. Both entities serve as core modules to implement containerized execution of control workflows, run as pods, and talk to each other via gRPC. In addition, the initial data needed by the cloud workflow is stored in the Redis database. The workflow definition part reads the data to build the cloud workflow through the workflow injection module. The cloud environment on the right of Figure 5 is based on Kubernetes and provides a containerized running platform. Once deployed, the workflow injection module and containerized workflow builder work as core function pods and run on the Kubernetes cluster. Also, the NFS business mounting pod is deployed to provide NFS sharing service for all task pods within the same workflow.

**Workflow injection module.** This module is an independent module to inject the abstract cloud workflow service into containerized workflow builder. Its parser module takes care of reading variable configurations from the ConfigMap file in abstract workflow definition and parses the JSON-formatted workflow description. The packaging module is responsible for initializing, encapsulating workflow information, and sending workflow to the containerized workflow builder with the help of a queue mechanism.

**Containerized workflow builder.** This module is responsible for workflow reception, workflow containerization execution, resource monitoring, and resource allocation. It comprises four core subcomponents described below. Other low-level modules are described in Table 3.

Containerized executor. It comprises namespace creator, service creator, task container creator, and volume. It is in charge of receiving workflows and generating workflow namespaces, task services, and task pods. Task pods populate intermediate data into local storage via volume for NFS sharing. Each task pod within control workflows communicates with each other via gRPC based on the Kubernetes service.

Resource manager. It aims to obtain the remaining resources through resource discovery and offer resource provisioning for the task pods using resource allocator.

State tracker. It monitors the underlying resource states from the informer through the list-watch mechanism and responds to state queries of various resource objects to other modules at any time.

14) A PV is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using StorageClasses.

15) ClusterIP exposes the service on a cluster-internal IP and makes the service only reachable from within the cluster.

**Table 3** Descriptions of internal modules in Containerized Workflow Builder

| Module | Functionality |
|---|---|
| Namespace creator | Generate workflow namespace to realize resource isolation between different workflows |
| Service creator | Create task service corresponding to each task pod for gRPC communication between workflow tasks with dependencies |
| Task container creator | Create task pods under the current workflow namespace and bind PVC and PV to share with NFS shared directory between task pods |
| Volume | A local storage volume used to store intermediate data for hosted tasks pod |
| Resource discovery | Obtain the remaining resources across the Kubernetes cluster through Informer |
| Resource allocation | Offer the required resources for the newly created task pods with our customized resource allocation algorithm |

Informer. It synchronizes resource objects and events between Kubernetes core components and its local cache. It feeds back to resource discovery the remaining resources of the Kubernetes cluster and responds state tracker to monitor the state changes of the resource objects.

The containerized workflow builder uses the informer to update the resource objects in the Kubernetes cluster, monitors resource objects through the state tracker, provides resource provisioning with the help of the resource manager, and immediately creates workflow task pods via the containerized executor in response to workflow generating requests from the workflow injection module.

**Cloud environment.** The cloud environment for our control workflow engine is built based on the Kubernetes clusters, aiming to leverage rich cloud computing resources to meet the computational needs of complex control tasks. The Redis database is pre-deployed in a Kubernetes cluster. As a component of the Kubernetes control plane, the Kube-apiserver[16)] works to accommodate resource objects of the Kubernetes cluster to all kinds of APIs and provides the cluster's shared state to the front through which all other components interact.

## 4 Workflow construction for SID algorithm

To evaluate the performance of ControlService, we use the methods of distributed lightweight algebra operation to construct the cloud workflow for the SID algorithm.

### 4.1 Subspace identification method

The SID methods have been widely applied in many fields, such as power grids [25], chemical processing [26], data-driven control [27], and fault detection [28]. With SID methods, we can use the historical data to build the standard state-space models for analyses and designs. Singular value decomposition (SVD) is a reliable matrix operation that enables accurate identification of the state-space models based on input and output data under mild conditions. Herein, the high-dimension and low-rank Hankel matrices are derived to store the historical data and obtain the final system model. However, the large-scale data matrices in SID usually increase the computational amount of matrix operations, which makes the SID method impossible for large-scale or real-time identification tasks.

### 4.2 SID cloud workflow design

Recently, a workflow-based SID method was proposed in [11] to accelerate SID for large-scale or real-time identification tasks. Intuitively, the SID algorithm uses the parallel features of workflow topology to restructure its own structure so as to match the distributed cloud infrastructure and obtain superior computing performance. We take the classical SID method like N4SID (as shown in Algorithm 1) [29] as an example to study the workflow reconstruction process of the N4SID algorithm and obtain abstract workflow form suitable for our ControlService solution. The oblique projection and other new variables of Algorithm 1 are defined in Appendix A.

By borrowing the workflow restructured method from [11], we get the workflow structure with the degree of parallelism of the SVD stage being $n/\mathrm{col} = 10$ of Algorithm 1, as shown in Figure 6. The MPT in Figure 6 represents the maximum number of parallel tasks in a workflow.

16) The Kube-apiserver serves as the front end and is a component of the Kubernetes control plane that exposes the Kubernetes API.

**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:
$\mathcal{O}_i = Y_f/_{U_f}\boldsymbol{W}_p$,
$\mathcal{O}_{i-1} = Y_f^-/_{U_f^-}\boldsymbol{W}_p^+$;

2: Calculate the SVD of the weighted oblique projection: $W_1\mathcal{O}_i W_2 = USV^{\mathrm{T}}$;

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

$$USV^{\mathrm{T}} = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} S_1 & 0 \\ 0 & S_2 \end{bmatrix} \begin{bmatrix} V_1^{\mathrm{T}} \\ V_2^{\mathrm{T}} \end{bmatrix};$$

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

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

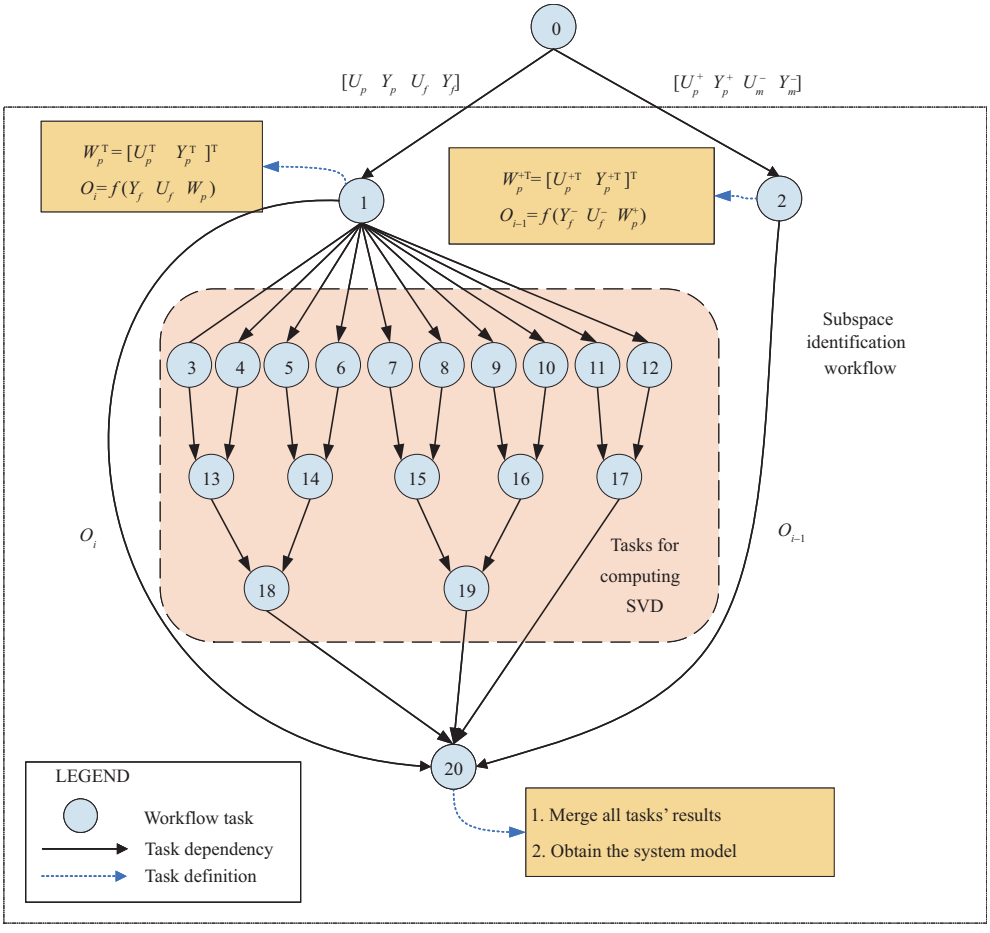


**Figure 6** (Color online) Workflow structure of the SID with MPT = 10.

Table 4 describes the functions of the tasks in the SID workflow and their relationships. There are five kinds of task images in Figure 6, except the initial task image $\mathrm{Image}_{\mathrm{ini}}$ concerning node 0 used in this workflow. For instance, the image $\mathrm{Image}_C$ can be reused ten times in the second level, of which the task indexes are $\mathrm{Task}_3$–$\mathrm{Task}_{12}$ but with different input data. The function and relationship of each task image in Figure 6 are described as follows.

(1) The initialization task $\mathrm{Task}_{\mathrm{ini}}$ concerning node 0 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 $[U_p, Y_p, U_f, Y_f]$, $\mathrm{Task}_1$ 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 oblique projection. The result $O_i$ is divided into ten slices along the column direction.

**Table 4** Relationships between tasks in workflow and functions

| Image | Index | Level | Function |
|---|---|---|---|
| $\text{Image}_{\text{ini}}$ | $\text{Task}_{\text{ini}}$ | – | Prepare Hankel matrices by creating history data |
| $\text{Image}_A$ | $\text{Task}_1$ | 1 | Generate $O_i$ for SVD tasks |
| $\text{Image}_B$ | $\text{Task}_2$ | 1 | Generate $O_{i-1}$ for calculating extended state matrix $X_{i+1}$ |
| $\text{Image}_C$ | $\text{Task}_3$–$\text{Task}_{12}$ | 2 | Compute the truncated SVD of each block of $O_i$ |
| $\text{Image}_D$ | $\text{Task}_{13}$–$\text{Task}_{19}$ | 3, 4 | (1) Merge the parent tasks' results; (2) compute new truncated SVD |
| $\text{Image}_E$ | $\text{Task}_{20}$ | 5 | (1) Merge all tasks's results; (2) obtain the final system model |

**Table 5** Experimental platform configuration

| Experimental mode | Scale | Infrastructure specification | Software | Hardware | |
|---|---|---|---|---|---|
| Baseline | One node | ecs.hfc6.xlarge | ubuntu 20.4 | CPU | 4 cores |
| | | | | Memory | 8 GB |
| | | | | Bandwith | 2 Gbps |
| Argo | Kubernetes cluster: | ecs.hfc6.2xlarge | ubuntu 20.4 | CPU | 8 cores |
| | One master | | Argo 3.3.8 | Memory | 16 GB |
| | Six nodes | | Kubernetes 1.19.6 | Bandwith | 2 Gbps |
| ControlService | Kubernetes cluster: | ecs.hfc6.2xlarge | ubuntu 20.4 | CPU | 8 cores |
| | One master | | Kubernetes 1.19.6 | Memory | 16 GB |
| | Six nodes | | Docker 18.09.6 | Bandwith | 2 Gbps |

(3) Similarly, $O_{i-1}$ is calculated by $\text{Task}_2$ based on $[U_p^+, Y_p^+, U_f^-, Y_f^-]$.

(4) These ten slices are delivered to $\text{Task}_3$–$\text{Task}_{12}$ as input data, respectively.

(5) $\text{Task}_{13}$–$\text{Task}_{19}$ carry out the MAT operations on the results of $\text{Task}_3$–$\text{Task}_{12}$, i.e., merge the parent tasks' results and compute the new truncated SVDs.

(6) Finally, the export task $\text{Task}_{20}$ collects all the results of $\text{Task}_1$, $\text{Task}_2$, $\text{Task}_{17}$, $\text{Task}_{18}$, and $\text{Task}_{19}$ and merges them and later estimates the system model through the least squares method.

### 4.3 Algorithm complexity analysis

According to the source code of workflow tasks[17)], workflow tasks communicate with each other by gRPC mechanism. The task corresponding to each node will traverse its input and output branches, and the maximum value of the latter determines the MPT in a workflow. The complexity of the SID workflow algorithm in the ControlService depends on the workflow's MPT, so the time complexity of the SID workflow algorithm is O(MPT).

## 5 Experimental evaluation

In the following, we evaluate ControlService and compare it with the state-of-the-art Argo workflow engine (Argo) and the single-machine running mode of the SID algorithm. Herein, we do not reconstruct the SID algorithm (N4ISD) using the workflow-based method in the single-machine running mode (Baseline). For the SID algorithm, we make comparisons and analyses under different execution situations of ControlService, Argo, and Baseline.

### 5.1 Experimental setup

The Baseline runs in the cloud server with a 4-core CPU and 8 GB memory. Its server specification is ecs.hfc6.xlarge in Alibaba Cloud. ControlService and Argo require a Kubernetes cluster to test SID cloud workflow. The Kubernetes cluster consists of one master and six nodes. Each node possesses an 8-core CPU and 16 GB memory, running Ubuntu 20.4, Kubernetes v1.19.6, and Docker version 18.09.6. Python 3.8, Numpy 1.23.5, and Scipy 1.9.3 work as scientific libraries in workflow task implementation. Table 5 lists the hardware and software parameters of the three experimental environments. We adopt the following cloud workflow submission methods to test ControlService, Baseline, and Argo solutions.

**Baseline.** The Baseline uses single machine operation mode to run Algorithm 1 without workflow structure.

17) https://github.com/CloudControlSystems/ControlService/tree/main/CotrolService-solution/sid-source.

**Argo.** As a general cloud-native workflow engine in the industry, Argo has a certain representative. We define the workflow task dependency relationship as DAG, described via the Argo workflow definition template in the Yaml file, and submit the Yaml file to the Argo workflow engine through the Argo binary tool. For Argo, we redefine the task code to exchange data between tasks via Redis or NFS.

**ControlService.** We employ the containerized method to deploy the control workflow engine within ControlService as mentioned in (Subsection 3.3).

### 5.2 Metrics

The following quantitative metrics are used to evaluate the performance of our ControlService against Argo and Baseline.

**Computation time.** Represent the elapsed time of Algorithm 1 for the Baseline. In Argo and ControlService, the computation time refers to the lifecycle of a workflow in which all workflow tasks are complete.

**Reduction percentage.** Represent the decline degree in the computation time parameter against the Baseline and reflect the improvement degree of computation efficiency.

**Speedup.** Stand for the ratio of the processing time on a single processor and on multi-processors, expressed as

$$S_p = T_s/T_p, \tag{1}$$

where $T_s$ is the processing time of sequential execution for the initial algorithm without workflow structure, and $T_p$ is that of parallel execution for reconstructed workflows.

### 5.3 Results and analysis

With external influences in mind, the Kubernetes cluster carries no other load and runs each evaluation three times at different times of one day. This subsection uses algorithm 1 and two SID workflow structures after algorithm reconstruction to test the execution performance of ControlService, Argo, and Baseline, evaluate the efficiency of SID workflows after algorithm reconstruction, and verify the feasibility of ControlService solution. The resource units C and G of Groups 1 and 2 in Table 6 represent the number of CPU core and memory capacity Gi bytes (i.e., $1024 \times 1024 \times 1024$ B) in the Kubernetes task container, respectively.

The first SID workflow structure is tested in Group 1 with an MPT of 2 and a total task number of 5, as shown in Figure 7. The mapping between workflow task nodes and task images refers to Table 4. Herein, nodes 3 and 4 share the $\text{Image}_c$ of the $\text{Task}_c$. The second SID workflow structure shown in Figure 6 possesses an MPT of 10 and a total task number of 20, which is depicted in the second group of Table 6. The identified dynamic models of the two groups of workflow tests are as follows:

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

We use middle- and large-scale input data to test the execution performance of ControlService, Argo, and Baseline, respectively. The middle scale includes two sets of input data ($N = 10, j = 1000$) and ($N = 20, j = 1000$). The large scale includes four sets of input data ($N = 10, j = 10000$), ($N = 20, j = 10000$), ($N = 50, j = 10000$), and ($N = 50, j = 20000$). Herein, $N$ and $j$ represent the number of rows and columns of the Hankel matrix $U_p$ and $U_f$, respectively. The mean values of experimental evaluation results are recorded in Table 6.

**Comparison and analysis of Argo and ControlService methods.** For SID workflows of different concurrency MPT, Table 6 shows the execution time of ControlService and Argo under six groups of middle-scale and large-scale input data, respectively. For the Argo workflow engine, the calculational time metrics of Groups 1 and 2 at middle- and large-scale are not advantageous because of Argo's unique workflow processing logic. Argo is responsible for workflow containerization execution through its core components, including the Argo server and workflow controller. The Argo server calls Kubernetes Api to operate resource objects, and the workflow controller responds to resource changes in CRD (custom resource definition) to realize workflow containerization[18]. The unique workflow processing logic has

18) Argo Workflows—Github, 2023. https://github.com/argoproj/argo.

**Table 6** Evaluation results in Monte Carlo experiments for SID workflows

| Parameter | | | | Middle-scale | | Large-scale | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | $N = 10$ $j = 1000$ | $N = 20$ $j = 1000$ | $N = 10$ $j = 10000$ | $N = 20$ $j = 10000$ | $N = 50$ $j = 10000$ | $N = 50$ $j = 20000$ |
| Baseline | | | Time (s) | 0.015 | 0.025 | 1.84 | 2.48 | 4.431 | 18.191 |
| Group 1: MPT = 2 Tasks = 5 | 4C 8G | Argo | Time (s) | 30.41 | 30.38 | 30.41 | 30.405 | 30.409 | 30.584 |
| | | | Reduction (%) | −202633 | −121420 | −1553 | −1126 | −586 | −68 |
| | 1C 2G | Control-Service | Time (s) | 0.178 | 0.299 | 1.746 | 2.863 | 6.568 | 22.389 |
| | | | Reduction (%) | −1086 | −1096 | 5.1 | −15 | −48 | −23 |
| | 4C 8G | Control-Service | Time (s) | 0.088 | 0.131 | 0.709 | 1.029 | 1.947 | 6.988 |
| | | | Reduction (%) | −487 | −424 | 61 | 59 | 56 | |
| Group 2: MPT = 10 Tasks = 20 | 1C 2G | Argo | Time (s) | 50.49 | 50.521 | 50.689 | 50.541 | 50.566 | 50.643 |
| | | | Reduction (%) | −336500 | −201984 | −2655 | −1938 | −1041 | −178 |
| | 1C 2G | Control-Service | Time (s) | 0.339 | 0.477 | 0.608 | 0.709 | 1.348 | 3.134 |
| | | | Reduction (%) | −2160 | −1808 | 67 | 71 | 70 | 83 |

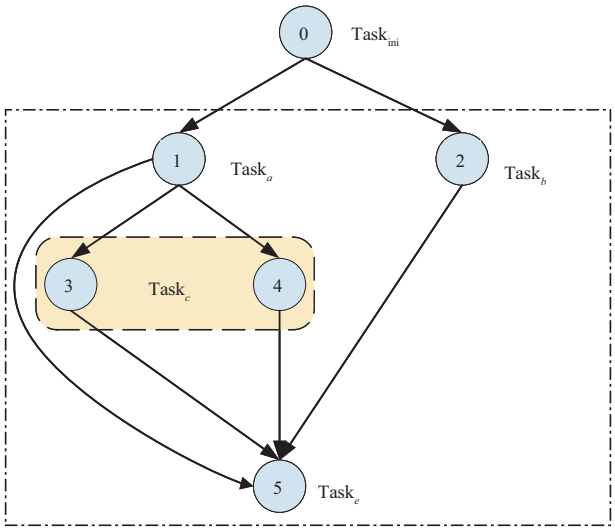


**Figure 7** (Color online) Workflow structure of the SID with MPT = 2.

no advantage over the ControlService solution in terms of execution time because the control workflow engine in ControlService is superior to Argo in terms of workflow processing time [18]. All resource events in the Argo workflow need to interact with Kube-apiserver. Too much reading and writing to the Etcd database causes Kubernetes to respond slowly and prolongs the execution time of the task container. In addition, the data interaction between SID workflow tasks in Argo uses the Redis database as a temporary storage, and this frequent access to Redis also consumes a significant amount of workflow execution time. After many experimental tests for the Argo method, we find that the high-dimensional matrix operation under middle- and large-scale data input accounts for a small proportion of the workflow computing time. Therefore, as the input data $(N, j)$ increases, the workflow computation time of the Argo method remains around 30.4 s with little change. For Group 2, as the number of SID tasks increases, the connection, reading, and writing to Redis becomes more frequent and consumes more time. The computation time of the workflow increases rapidly from about 30.4 s to about 50.5 s. Similar to Group 1, the variation of input data $(N, j)$ has little effect on the computation time of the workflow.

**Impact of different data scales on computation time.** Figure 8 depicts the variation in the computation time curve of workflow or Baseline algorithm for different data scales. The data scale spans from $(N = 10, j = 1000)$ to $(N = 50, j = 20000)$. As shown in Figure 8, the larger the data scale, the larger the computation brought by the high-dimensional matrix, and each method will consume more computation time. However, for the Argo method, due to the unique processing logic of Argo (as described in the previous paragraph), the workflow computation time of the Argo method of Groups 1 and 2 from Table 6 reaches about 30.4 and 50.5 s, respectively, with no growth trend and little change in computing time, and the results are not ideal. From Figure 8(a), the workflow computation time of

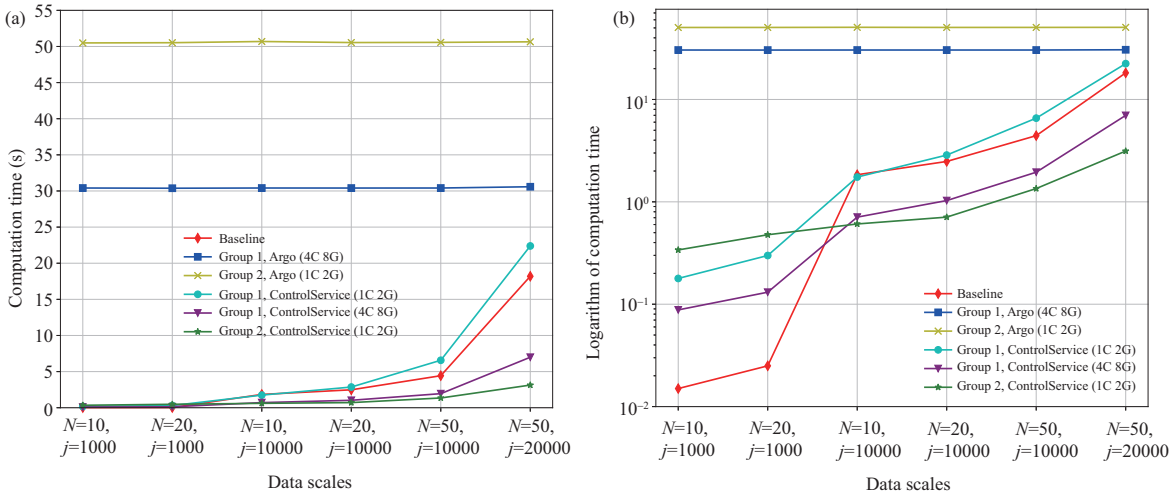


**Figure 8** (Color online) Variation of computation time for different data scales. (a) Computation time; (b) logarithm of computation time.

the Baseline, ControlService's Group 1, and ControlService's Group 2 gradually increases as the data scale increases. When the data scale ranges from $(N = 50, j = 10000)$ to $(N = 50, j = 20000)$, the computation time curves of all the methods except the Argo method experience a rapid rise. Only the time curve of the ControlService method of Group 2 rises gently, which is closely related to the workflow with high concurrency (MPT = 10) used in this method. The high concurrency topology of workflow accelerates the execution speed of workflow and significantly reduces the computation time of workflow.

Figure 8(b) uses a logarithm figure to plot computation time for different data scales and to zoom in on Figure 8(a). As shown in Figure 8(b), from middle-scale $(N = 20, j = 1000)$ to large-scale $(N = 10, j = 10000)$, excluding the Argo method, the calculation time curves of other methods all increase sharply, which is closely related to the sudden increase of data scale. At the same time, Group 1's ControlService (4C 8G) and Group 2's ControlService (1C 2G) methods quickly surpass the Baseline and keep the workflow computation time low. Among them, the ControlService (1C 2G) method of Group 2 shows outstanding advantages thanks to its high workflow concurrency. Even if the container resource configuration reduces to (1C 2G), the acceleration of workflow execution is not affected. For the ControlService (1C 2G) method of Group 1, although it has workflow concurrency with an MPT of 2, the workflow computation time is slightly higher than the Baseline due to the smaller container resource configuration (1C 2G). We can see that the smaller container resource configuration affects Group 1's ControlService (1C 2G) method, resulting in higher computation time.

**Reduction percentage.** Table 6 shows that except for the ControlService (1C 2G) method of Group 1, the performance advantage of workflow computation time in ControlService is significant in the case of large-scale input data. Compared with the Baseline algorithm, the workflow computation time of Group 1's ControlService (4C 8G) reduces by 56% to 62%, and the workflow computation time of Group reduces by 67% to 83%. Figure 9 also shows that under large-scale input data, the ControlService is more suitable for containerized execution of SID workflows in Groups 1 and 2. The workflow computation time is much less than that of Argo and the Baseline algorithm, which is very useful for large-capacity data and high-complexity model scenarios, such as power grids and multi-agent domains. As shown in Table 6 and Figure 8(b), the workflow computation time metric of the ControlService on the middle scale is higher than that of the Baseline algorithm. It is because the gRPC communication mechanism between tasks in the ControlService solution does not play its advantage in transferring task data in the middle-scale. If the ControlService method runs the SID workflow in a data-driven real-time dynamic control field with high-frequency requirements, the workflow computation time needs to be reduced to less than 20 ms. It requires further optimization of the gRPC communication mechanism between tasks in the ControlService scheme. It is also our future research direction.

**Speedup.** The speedup metric reflects the ratio of the execution time between the Baseline and the reconstructed workflow algorithm under the Argo and ControlService methods. As shown in Figure 10, taking the Baseline as a reference, the Argo and ControlService solutions do not have advantages under middle-scale input data $(N = 10, j = 1000)$ and $(N = 20, j = 1000)$. The reason is similar to the analysis

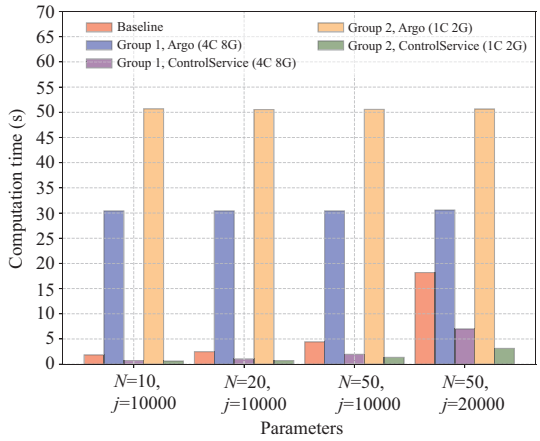


**Figure 9** (Color online) Comparison of computation time under large-scale input data.

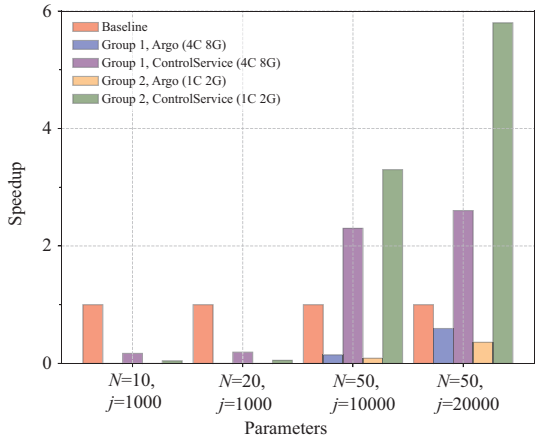


**Figure 10** (Color online) Relationship between workflow scale and speedup.

of the above subsections. For large-scale input data ($N = 50, j = 10000$) and ($N = 50, j = 20000$), the speedup ratio of computation time of the ControlService solution under Group 1 reaches 2.3 and 2.6, respectively. The speedup ratio of calculation time in Group 2 reaches 3.3 and 5.8, respectively. Because of the large amount of data processed by workflow tasks, Group 2 with high concurrency MPT shows a better performance advantage. Experimental results show that the workflow structure with reasonable concurrency can obtain a smaller computation time and a higher speedup ratio of computation time. The speedup ratio metric reflects that the SID algorithm with workflow reconstruction is significantly better than the Baseline in computing efficiency and also verifies the superior performance of the ControlService method.

**Container resource configurations.** Figure 11 depicts the computation time of the Baseline and ControlService methods for different container resource configurations. The Baseline runs on the standalone node of (4C 8G), and Group 1 divides the ControlService into two resource configurations: (1C 2G) and (4C 8G). In Group 1, for the SID workflow with an MPT of 2 and 5 tasks, the Kubernetes cluster is able to accommodate both resource configurations to ensure the successful execution of the workflow. Group 2 has only one container resource configuration (1C 2G) for the ControlService method. It is due to the fact that for the SID workflow with an MPT of 10 and 20 tasks in Group 2, considering the simultaneous start of the task containers in the ControlService, the Kubernetes cluster can only supply 20 task containers with resource configuration (1C 2G) rather than resource configuration (4C 8G). From Group 1 in Figure 11, as the container resource configuration increases from (1C 2G) to (4C 8G) at each data scale, the SID workflow computation time decreases, which is much better than the Baseline algorithm. From Group 1 to Group 2, the MPT of SID workflow increases from 2 to 10, and although the container resource configuration is (1C 2G), it still shows a low workflow computation time. We can see that the larger the MPT of the workflow is, the more it can accelerate the execution of the workflow. The larger the container resource configuration, the more it can further reduce the computation time of the workflow. Group 1's ControlService (1C 2G) method affects the computing performance of tasks due to the low container resource configuration and low workflow concurrency, so the performance of workflow computing time is not superior to the Baseline algorithm.

In addition, the standard deviation error line of the Baseline in Figure 11 is small because the Baseline algorithm runs on a single machine node without external influences. In the case of ignoring the effect of container resource configuration, the ControlService methods all show small standard deviation error lines, indicating that the workflow calculation time is relatively concentrated, which also reflects the stability and reliability of the control workflow engine in ControlService.

## 6 Conclusion and future work

In this paper, we have presented the ControlService solution for containerized implementation of the C3aS in CCSs. The ControlService solution uses a workflow-based method to reconstruct various control

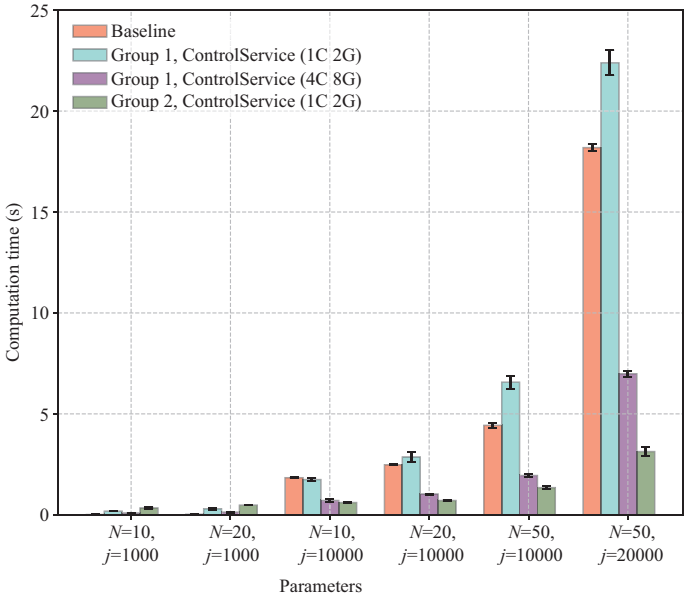


**Figure 11** (Color online) Relationship between computational time and container resource configurations.

algorithms in CCSs, presents the workflow definition of this algorithm in JSON format and the delivery method of the cloud service instance, realizes the customized control workflow engine, and verifies the feasibility of the solution by taking the SID algorithm as an example. Experimental results show that our ControlService, compared to the Baseline and Argo, has significant advantages in computational time, reduction percentage, and speedup ratio. In future work, we will further optimize the gRPC communication mechanism between tasks in the ControlService solution to adapt to the real-time requirements of workflow containerized execution under middle-scale input data and to be suitable for the data-driven real-time dynamic control field with high-frequency requirements.

**Acknowledgements** This work was supported by National Natural Science Foundation of China (Grant Nos. 61836001, 62173035).

**References**

1 Xia Y Q, Fu M Y, Liu G P. Analysis and Synthesis of Networked Control Systems. Berlin: Springer, 2011
2 Zhang Y, Xia Y, Zhai D H. Structural controllability of networked relative coupling systems. Automatica, 2021, 128: 109547
3 Peng Z, Wang J, Wang D, et al. An overview of recent advances in coordinated control of multiple autonomous surface vehicles. IEEE Trans Ind Inf, 2020, 17: 732–745
4 Xia Y Q, Qin Y M, Zhai D-H, et al. Further results on cloud control systems. Sci China Inf Sci, 2016, 59: 073201
5 Xia Y, Zhang Y, Dai L, et al. A brief survey on recent advances in cloud control systems. IEEE Trans Circuits Syst II, 2022, 69: 3108–3114
6 Esen H, Adachi M, Bernardini D, et al. Control as a service (CaaS) cloud-based software architecture for automotive control applications. In: Proceedings of the 2nd International Workshop on the Swarm at the Edge of the Cloud (SWEC), Seattle, 2015. 13–18
7 Lyu M, Benfenatki H, Biennier F, et al. Control as a service architecture to support context-aware control application development. IFAC-PapersOnLine, 2019, 52: 1085–1090
8 Tessaro V, Vick A, Krüger J. Universal identification and control of industrial manufacturing equipment as a service. In: Proceedings of IOP Conference Series: Materials Science and Engineering, 2021. 012023
9 Buerkle A, Eaton W, Al-Yacoub A, et al. Towards industrial robots as a service (IRaaS): flexibility, usability, safety and business models. Robot Comput-Int Manuf, 2023, 81: 102484
10 Yang Z, Zhao Y, Dang F, et al. CaaS: enabling control-as-a-service for time-sensitive networking. In: Proceedings of the IEEE Conference on Computer Communications (INFOCOM), New York, 2023
11 Gao R, Xia Y, Wang G, et al. Fast subspace identification method based on containerised cloud workflow processing system. IEEE Trans Automat Sci Eng, 2024. doi: 10.1109/TASE.2023.3316287
12 Gao R, Li Q, Dai L, et al. Workflow-based fast data-driven predictive control with disturbance observer in cloud-edge collaborative architecture. IEEE Trans Automat Sci Eng, 2024. doi: 10.1109/TASE.2023.3270203
13 Wang Z, Yang S, Xiang X, et al. Cloud-based mission control of USV fleet: architecture, implementation and experiments. Control Eng Pract, 2021, 106: 104657
14 Liu B, Wang L, Liu M, et al. Federated imitation learning: a novel framework for cloud robotic systems with heterogeneous sensor data. IEEE Robot Autom Lett, 2020, 5: 3509–3516
15 Chu W, Wuniri Q, Du X, et al. Cloud control system architectures, technologies and applications on intelligent and connected vehicles: a review. Chin J Mech Eng, 2021, 34: 139
16 Dai L, Huang T, Gao R, et al. Cloud-based computational data-enabled predictive control. IEEE Internet Things J, 2022, 9: 24949–24962
17 Shan C, Wang G, Xia Y, et al. Containerized workflow builder for Kubernetes. In: Proceedings of IEEE 23rd International Conference on High Performance Computing & Communications, the 7th International Conference on Data Science & Systems,

the 19th International Conference on Smart City, the 7th International Conference on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), Haikou, 2021. 685–692
18 Shan C, Xia Y, Zhan Y, et al. KubeAdaptor: a docking framework for workflow containerization on Kubernetes. Future Generation Comput Syst, 2023, 148: 584–599
19 Bezerra D D F, Medeiros V W C D, Gonçalves G E. Towards a control-as-a-service architecture for smart environments. Simul Model Pract Theor, 2021, 107: 102194
20 Coleman T, Casanova H, Pottier L, et al. WfCommons: a framework for enabling scientific workflow research and development. Future Gener Comput Syst, 2022, 128: 16–27
21 Dai L, Ma Y, Gao R, et al. Cloud-based computational model predictive control using a parallel multiblock ADMM approach. IEEE Internet Things J, 2023, 10: 10326–10343
22 Borangiu T, Trentesaux D, Thomas A, et al. Digital transformation of manufacturing through cloud services and resource virtualization. Comput Industry, 2019, 108: 150–162
23 Viriyasitavat W, Xu L D, Dhiman G, et al. Service workflow: state-of-the-art and future trends. IEEE Trans Serv Comput, 2021, 16: 757–772
24 Tang X. Reliability-aware cost-efficient scientific workflows scheduling strategy on multi-cloud systems. IEEE Trans Cloud Comput, 2022, 10: 2909–2919
25 Wu T, Venkatasubramanian V M, Pothen A. Fast parallel stochastic subspace algorithms for large-scale ambient oscillation monitoring. IEEE Trans Smart Grid, 2016, 8: 1494–1503
26 Ghosh D, Hermonat E, Mhaskar P, et al. Hybrid modeling approach integrating first-principles models with subspace identification. Ind Eng Chem Res, 2019, 58: 13533–13543
27 Salim M, Ahmed S, Khosrowjerdi M J. A data-driven sensor fault-tolerant control scheme based on subspace identification. Intl J Robust Nonlinear, 2021, 31: 6991–7006
28 Cheng C, Wang Q, Nikitin Y, et al. A data-driven distributed fault detection scheme based on subspace identification technique for dynamic systems. Intl J Robust Nonlinear, 2023, 33: 3107–3128
29 van Overschee P, de Moor B. N4SID: subspace algorithms for the identification of combined deterministic-stochastic systems. Automatica, 1994, 30: 75–93

## Appendix A Definitions in N4SID algorithm

The matrices $\mathcal{A} \in \mathbb{R}^{p\times j}$ and $\mathcal{B} \in \mathbb{R}^{q\times j}$ are performed as follows.

**Definition 1.** Orthogonal projection.

The orthogonal projection of the row space of $\mathcal{A}$ into the row space of $\mathcal{B}$ is defined as $\mathcal{A}/\mathcal{B}$:

$$\mathcal{A}/\mathcal{B} = \mathcal{A}\mathcal{B}^{\dagger}\mathcal{B}, \tag{A1}$$

where † means the Moore-Penrose pseudoinverse.

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

**Definition 2.** Oblique projection.

The oblique projection of the row space of $\mathcal{A}$ along the row space of $\mathcal{B}$ into the row space $\mathcal{C} \in \mathbb{R}^{r\times j}$ is defined as

$$\mathcal{A}/\mathcal{B}\mathcal{C} = \left(\mathcal{A}/\mathcal{B}^{\perp}\right)\left(\mathcal{C}/\mathcal{B}^{\perp}\right)^{\dagger}\mathcal{C}. \tag{A2}$$

$U_p^+$ and $U_f^-$ are defined as

$$U_p^+ = \begin{bmatrix} u(0) & u(1) & \cdots & u(j-1) \\ u(1) & u(2) & \cdots & u(j) \\ \vdots & \vdots & \ddots & \vdots \\ u(N) & u(N+1) & \cdots & u(N+j-1) \end{bmatrix}, \tag{A3}$$

$$U_f^- = \begin{bmatrix} u(N+1) & u(N+2) & \cdots & u(N+j) \\ u(N+2) & u(N+3) & \cdots & u(N+j+1) \\ \vdots & \vdots & \ddots & \vdots \\ u(2N-1) & u(2N) & \cdots & u(2N+j-2) \end{bmatrix}, \tag{A4}$$

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

Then, $\boldsymbol{W}_p^+$ is defined as

$$\boldsymbol{W}_p^+ = \begin{bmatrix} Y_p^+ \\ U_p^+ \end{bmatrix}. \tag{A5}$$

In step 5 of Algorithm 1, $Y_i$ and $U_i$ are defined as

$$Y_i = \begin{bmatrix} y(N) & y(N+1) & \cdots & y(N+j-1) \end{bmatrix}. \tag{A6}$$

$$U_i = \begin{bmatrix} u(N) & u(N+1) & \cdots & u(N+j-1) \end{bmatrix}. \tag{A7}$$