Open Composer: A Web-Based Application for Generating and Managing Batch Jobs on HPC Clusters

Masahiro Nakao
RIKEN Center for Computational Science (R-CCS)
Kobe, Japan
masahiro.nakao@riken.jp

Keiji Yamamoto
RIKEN Center for Computational Science (R-CCS)
Kobe, Japan
keiji.yamamoto@riken.jp

Abstract

Submitting batch jobs on HPC clusters usually requires familiarity with Linux commands and job schedulers, posing a significant learning barrier for beginners. To address this issue, we have developed Open Composer, a web-based application that helps users generate and manage batch jobs on HPC clusters. Open Composer automatically generates shell scripts using web forms defined for each application while also providing a real-time preview of the generated shell scripts and allowing direct editing. This feature helps reduce both the learning curve and the risk of syntax errors while maintaining flexibility in script writing. Open Composer provides a unified interface for job submission and status monitoring, and supports reusable job parameters, dynamic form widgets, and preprocessing steps. By enhancing usability and accessibility, Open Composer aims to make HPC resources more approachable for both novice and experienced users. Built on top of Open OnDemand[3], a widely adopted web portal framework for HPC environments, Open Composer ensures portability across various HPC clusters and seamless integration with institutional infrastructure, including authentication mechanisms.

CCS Concepts

• **Software and its engineering** → **Integrated and visual development environments**; *Organizing principles for web applications*; • **Human-centered computing** → Web-based interaction.

Keywords

high performance computing, web platform, user experience

ACM Reference Format:
Masahiro Nakao and Keiji Yamamoto. 2025. Open Composer: A Web-Based Application for Generating and Managing Batch Jobs on HPC Clusters. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3731599.3767428

1 Introduction

HPC clusters are widely used across various scientific and engineering domains, but their use often involves a steep learning curve. Specifically, users must complete several preparatory steps, such as installing an SSH client, generating a key pair, and registering the public key. In addition, familiarity with Linux commands and job schedulers through a command line interface (CLI) is required. When using graphical user interface (GUI) applications such as remote desktop environments or JupyterLab interactively, additional configurations like SSH port forwarding are also necessary. These configuration procedures and knowledge requirements vary across HPC clusters, making it necessary for users to relearn them each time they access a new cluster.

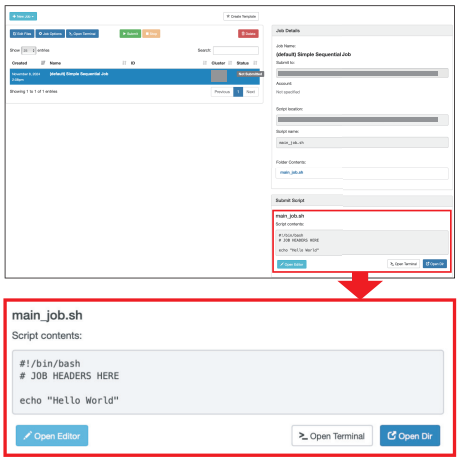

Figure 1: Job Composer on Open OnDemand

Open OnDemand[3], an open source web portal for accessing and operating HPC clusters, has been developed to address this issue and is widely used in HPC environments. Open OnDemand enables users to perform tasks such as file management, data transfer, job submission and monitoring, terminal access, and interactive use of GUI applications, all through a familiar web browser interface. By using Open OnDemand, users can reduce the learning burden, avoid complex setup processes, and access multiple HPC clusters through a unified interface. Moreover, Open OnDemand serves as an application platform that includes several pre-installed applications and also supports the installation of custom applications.

The primary use case of HPC clusters is to execute real-world applications as batch jobs. In this context, a batch job refers to a non-interactive computing task executed by a job scheduler. Batch jobs are usually written as shell scripts, where required computing resources, such as the number of nodes, are specified using scheduler directives (e.g., `#SBATCH -N 4`). Once submitted, the job scheduler determines appropriate compute nodes based on the current availability of the HPC cluster and then executes the batch job.

This work is licensed under a Creative Commons Attribution 4.0 International License.
SC Workshops '25, St Louis, MO, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1871-7/25/11
https://doi.org/10.1145/3731599.3767428

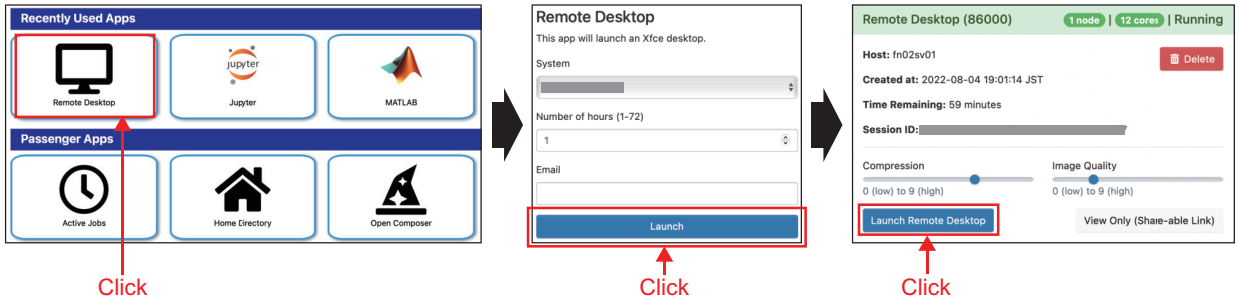

Figure 2: How to start a GUI application on Open OnDemand

To submit batch jobs via Open OnDemand, users can use Job Composer, a pre-installed application of Open OnDemand, as shown in Fig. 1. Job Composer allows users to generate, submit, and manage jobs through a web browser. However, as illustrated in the lower part of Fig. 1, job scripts still need to be written manually. This requires users to specify application execution commands and include scheduler directives, which imposes a significant learning burden and increases the likelihood of syntax errors.

To address this issue, we developed Open Composer, a web application that generates job scripts using parameters entered through application-specific web forms. This approach reduces the need to learn scheduler-specific syntax and helps minimize errors in script writing. In addition, Open Composer provides functionality for job submission and basic job management, and offers support for reusable job parameters, dynamic form widgets, and optional preprocessing steps, thereby reducing the burden on users. Open Composer is implemented as an application running on Open OnDemand and is integrated with its useful features, allowing seamless deployment in existing HPC environments and taking advantage of familiar web-based tools.

The structure of the remainder of this paper is as follows. Section 2 provides an overview of related work on automatic job script generation. Sections 3 and 4 cover the development and configuration of Open Composer. Finally, Section 5 summarizes the paper.

2 Related work

2.1 Web Form on Open OnDemand

The procedure for launching GUI applications in Open OnDemand is illustrated in Fig. 2. From the Open OnDemand dashboard, shown on the left in Fig. 2, clicking the icon of the desired GUI application opens the corresponding web form, shown in the center. After entering the required job parameters and clicking the "Launch" button, a job script is automatically generated within Open OnDemand and submitted to the job scheduler of the HPC cluster. When the GUI application starts running on a compute node, the user can click the "Launch `(application name)`" button shown on the right in Fig. 2 to display the application in their web browser via Open OnDemand's reverse proxy functionality.

As an alternative to Job Composer described in Section 1, efforts have been made to manage batch jobs using Open OnDemand's web forms for interactive jobs, where the submission procedure involves only the steps shown on the left and center of Fig. 2[5].

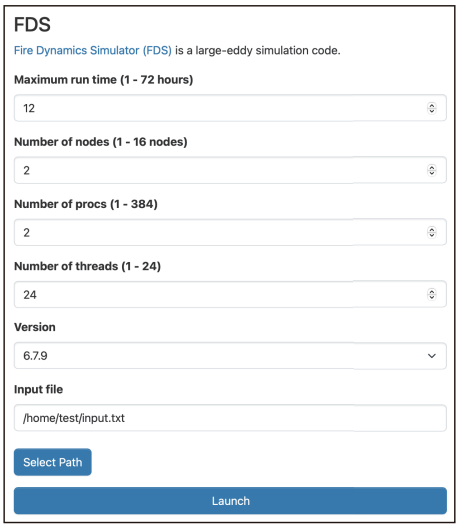

Figure 3: Example of a web form on Open OnDemand

```
1  script:
2    native:
3      - -t <%= hours %>:00:00
4      - -N <%= nodes %>
5      - -n <%= procs %>
6
7  batch_connect:
8    template: "basic"
9    script_wrapper: |
10     #!/usr/bin/env bash
11     export OMP_NUM_THREADS=<%= threads %>
12     spack load fds@<%= version %>
13     cd $(dirname <%= input_file %>)
14     mpiexec fds_mpi <%= input_file %>
```

Figure 4: Template of a job script

An example of a web form used for batch job submission is shown in Fig. 3, based on which a job script is generated using the user input and the job script template shown in Fig. 4. The web form and template are both defined in YAML with ERB (Embedded Ruby) syntax. Sections enclosed in `<%= %>` in Fig. 4 are replaced with values entered in the web form. Lines 1 to 5 define parameters

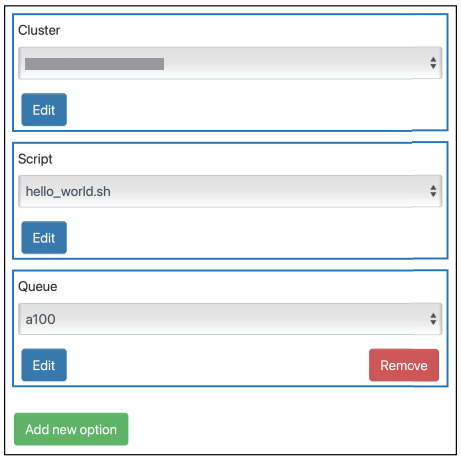

Figure 5: Project Manager on Open OnDemand

passed to the job scheduler, while lines 7 to 14 define the job script itself.

This batch job submission procedure completely hides both the application execution commands and the scheduler directives from users, and thus can be considered a solution to the issues associated with Job Composer. However, as a trade-off, the limited flexibility in editing the generated job script has emerged as a new issue. Since the job script is generated from predefined web forms and job script templates, users cannot add arbitrary application execution commands to the job script.

2.2 Application on Open OnDemand

As of July 2025, Open OnDemand includes a web application called Project Manager that is pre-installed as a beta feature in its latest version 4.0.6. Fig. 5 shows the Project Manager interface. Project Manager provides a feature that allows users to pass options to the job scheduler's submission command via a web form, offering functionality equivalent to specifying scheduler directives. Although Project Manager also includes functionality to create projects and register job scripts for each project, users are still required to manually describe the application execution commands. Since Project Manager does not provide script editing functionality, users need to either edit job scripts using other Open OnDemand features or upload existing job scripts.

Drona Workflow Engine[7, 9, 10] can generate job scripts containing both scheduler directives and application execution commands on Open OnDemand, as shown in Fig. 6. It should be noted that Drona Workflow Engine excels as a workflow engine for interactively creating complex job scripts, incorporating user-facing features such as warnings. This emphasis differs somewhat from the focus of this article, which is centered on generating relatively simple job scripts. In the upper section of Fig. 6, users can fill out a web form and click the "Preview" button to display the generated job script in the lower section. There, they can review and modify the job script before clicking "Submit Job" to launch the job. The "driver.sh" tab allows users to define preprocessing steps that are executed prior to job submission. Additionally, the "Live Output"

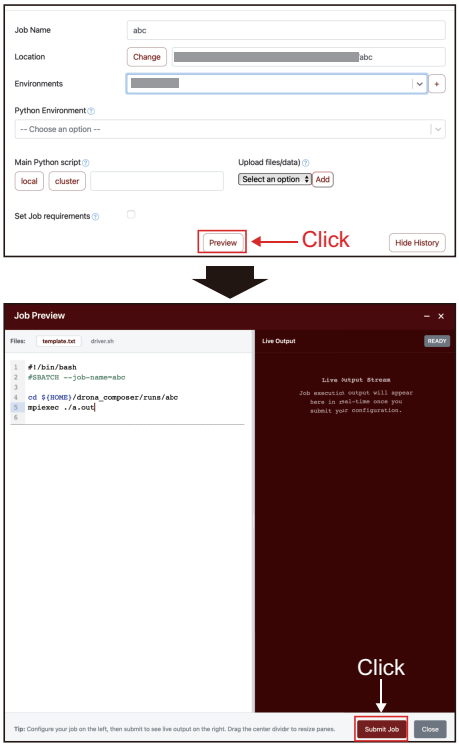

Figure 6: Drona Workflow Engine on Open OnDemand

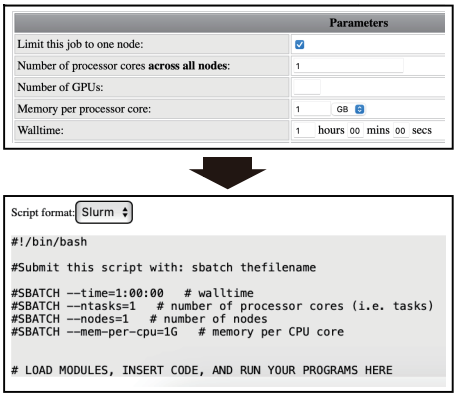

Figure 7: BYU Job Script Generator

section on the right provides real-time feedback from the submitted job. As of July 2025, based on the latest version of Drona Workflow Engine tested by the authors, the system supports both job submission and script reuse. However, features such as job status monitoring and canceling running jobs could not be confirmed.

2.3 Standalone Web Application

The previous discussion focused on web applications that run within Open OnDemand. This section describes those that operate independently. BYU Job Script Generator[2] is a web application designed to generate job scheduler directives for both Slurm and

Table 1: Comparison of batch job script generation applications

	Job Composer	Web form on Open OnDemand	Project Manager	Drona Workflow Engine	BYU Job Script Generator	**Open Composer**
Script Auto-Generation	No	Yes	(Partial)	Yes	(Partial)	**Yes**
Editable Script	Yes	No	No	Yes	No	**Yes**
Job Management	Yes	Yes	Yes	(Not confirmed)	No	**Yes**

PBS, as shown in Fig. 7. A notable feature of BYU Job Script Generator is its ability to display a real-time, automatically updated preview of the job script as users enter parameters into the web form, making it easy to review the script content. Although many similar web applications exist[4, 6, 11, 12], they are limited to generating only scheduler directives and do not support the generation of application execution commands. Moreover, these tools lack integration with job schedulers and therefore cannot be used to submit or manage jobs.

```
1 class Scheduler
2   def submit(...)
3     raise NotImplementedError
4   end
5 end
```

(a) Superclass

```
1 class Slurm < Scheduler
2   def submit(...)
3     :
```

(b) Subclass for Slurm

Figure 8: Adapter layer of a job scheduler

3 Development of Open Composer

3.1 Requirements

Open Composer aims to reduce the learning cost associated with generating batch job scripts and to provide an environment where users can easily submit the scripts to HPC clusters. By doing so, it enables a wide range of users, from beginners to experienced researchers, to use HPC resources more effectively. To achieve these goals, the following core requirements have been identified:

Script Auto-Generation Job scripts for various applications must be automatically generated based on user input through a web interface. This includes error checking to prevent submission failures due to incorrect parameters.

Editable Script Users should be able to directly modify the generated job scripts to suit specific needs.

Job Management The system should support job submission, as well as management tasks such as monitoring job status, canceling jobs, and reusing previously submitted job scripts with modified parameters. Moreover, the interface must unify access to different HPC clusters, hiding scheduler differences.

Table 1 summarizes how the applications discussed in the previous sections, along with Open Composer, address each of the identified requirements.

3.2 Design

Like many of the applications listed in Table 1, Open Composer is implemented as a web application running on Open OnDemand. Applications on Open OnDemand are launched with user privileges after users are authenticated by Open OnDemand. Therefore, Open Composer does not require its own user authentication mechanism. Since Open OnDemand already provides features such as file transfer, as described in Section 1, Open Composer does not need to implement them separately. This helps reduce development effort.

To fulfill the **Script Auto-Generation** requirement, Open Composer provides the following key features: (1) Automatic generation of job scripts from predefined web forms for specific applications, (2) Error checking for the input parameters provided in the web forms, and (3) A live preview of the job script that updates dynamically as users complete the form. These features are similar to those found in Drona Workflow Engine and BYU Job Script Generator. For the **Editable Script** requirement, Open Composer allows users to make direct modifications to the job script before submission. To address the **Job Management** requirement, Open Composer provides functions for submitting and canceling jobs, monitoring their status, and reusing previously submitted job scripts. It also offers a unified interface that abstracts differences among various job schedulers, ensuring a consistent user experience across HPC clusters. These capabilities enable users to manage the entire job execution process within a single interface, thereby enhancing both usability and operational efficiency.

In addition, Open Composer enhances integration with Open OnDemand by providing one-click access to frequently used tools. For example, users can launch a visualization application to analyze job output or open a terminal session or file browser directly from the job view.

3.3 Implementation

Since Open OnDemand uses the Ruby-based web framework Sinatra[1] as one of its backend components, Open Composer also uses Sinatra for its backend implementation. On the frontend, it employs Bootstrap for responsive layout and HTML together with JavaScript to support real-time job script preview and updates. To deploy Open Composer, an administrator simply places its source code in the Open OnDemand application directory, as Open Composer relies solely on software dependencies that are already installed as part of the packaged installation of Open OnDemand. By leveraging the features of Open OnDemand, Open Composer can also be deployed in a user's home directory, allowing only that user to access it.

To ensure compatibility across different job schedulers, Open Composer introduces an adapter layer that abstracts the differences among them. This is achieved by defining a generic superclass with standard functions such as submit(), cancel(), and query(), and implementing specific subclasses for each supported scheduler. At present, adapters are available for Slurm, PBS Pro, Sun Grid Engine, and Fujitsu TCS. A simplified example is shown in Fig. 8, where

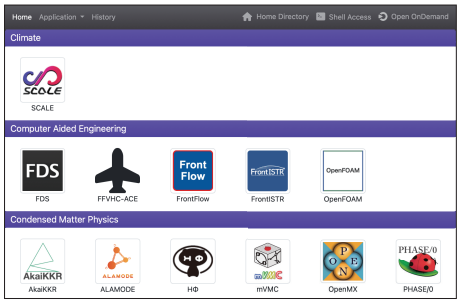

Figure 9: Home page on Open Composer

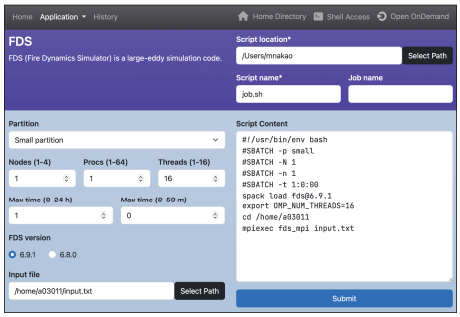

Figure 10: Application page on Open Composer

Fig. 8a defines the superclass and Fig. 8b shows an implementation for Slurm. This approach is also employed in Open OnDemand.

Open Composer has been released as open-source software, available at [8]. In the latest version of Open Composer (v1.5.0), released in May 2025, the codebase consists of 1,871 lines of Ruby, 1,036 lines of JavaScript, and 363 lines of HTML, including ERB templates. Of the Ruby code, the superclass shown in the left part of Fig. 8 consists of 20 lines, and the average number of lines for the subclasses corresponding to the four types of schedulers is 104.

3.4 Interface

Open Composer consists of three main pages: the home page, the application page, and the history page. This section provides an overview of each page.

3.4.1 Home Page. Fig. 9 shows the home page, where application icons are organized by category. Clicking an icon takes the user to the corresponding application page.

The navigation bar at the top of each page is shared across all pages. Links on the left provide access to each page, with the application page accessible via a dropdown menu. Links on the right include Open OnDemand Applications: "Home Directory" (for uploading and managing files), "Shell Access" (a web-based shell terminal), and a link to Open OnDemand itself.

3.4.2 Application Page. Fig. 10 shows the application page, where users can automatically generate a job script and submit it to the job scheduler.

The page is divided into four regions: top-left, top-right, bottom-left, and bottom-right. The top-left region displays the application name and its description. The top-right region includes fields for specifying the script location, script name, and job name. The bottom-left region contains input fields for application parameters. When parameters are entered, a job script is generated in real time and displayed in the text area in the bottom-right region. The job script can also be edited manually before submission. Clicking the "Submit" button at the bottom-right submits the job script to the job scheduler.

3.4.3 History Page. Fig. 11 shows the history page, where users can view the history of jobs submitted through Open Composer and check their status.

The red buttons located at the top-left are used to cancel running jobs or delete job entries from the table below. Each row in the table at the bottom of the page represents a submitted job. To cancel or delete jobs, users must select the corresponding checkboxes in the leftmost column. The text area and radio buttons at the top-right of the page are used to filter the jobs displayed in the table. The text area allows users to search for jobs by keyword, while the radio buttons filter jobs by status: "All", "Running", "Queued", or "Completed".

Clicking the link in the Job ID column displays profiling information retrieved from the job scheduler ("A" in the figure). Clicking the Application link opens the corresponding application page in Open Composer ("B"). The icon next to this link represents a visualization application registered in Open OnDemand; clicking it launches the web form for starting that application ("C"). Clicking the link in the Script Location column or the adjacent terminal icon opens the Open OnDemand Home Directory ("D") or Shell Access ("E") applications, respectively, starting at that file location. Clicking the Script Name link opens a window that displays the contents of the submitted job script ("F"), and clicking the "Load parameters" button in that window opens the corresponding application page with the parameters preloaded.

4 Configuration of Open Composer

4.1 Overview

This section explains how to configure Open Composer. To configure an application, a separate directory should be created for each one, containing two YAML configuration files: `manifest.yml` and `form.yml`. If needed, each file can also be written using the ERB format. Throughout this section, we refer to these files as the manifest file and the form file, respectively.

The manifest file contains a description of the application. Fig. 12 shows an example manifest file. The name field specifies the application name, the category field defines the application category, and the icon field specifies the path to the icon image, and the description field provides a description of the application. These fields are used on both the home page and the application page, as shown in Fig. 9 and Fig. 10.

The form file consists of the form, script, header, check, and submit sections. These correspond to the main form widgets, job script, header widgets, check script, and preprocessing steps executed before job submission, respectively. Among these, the form

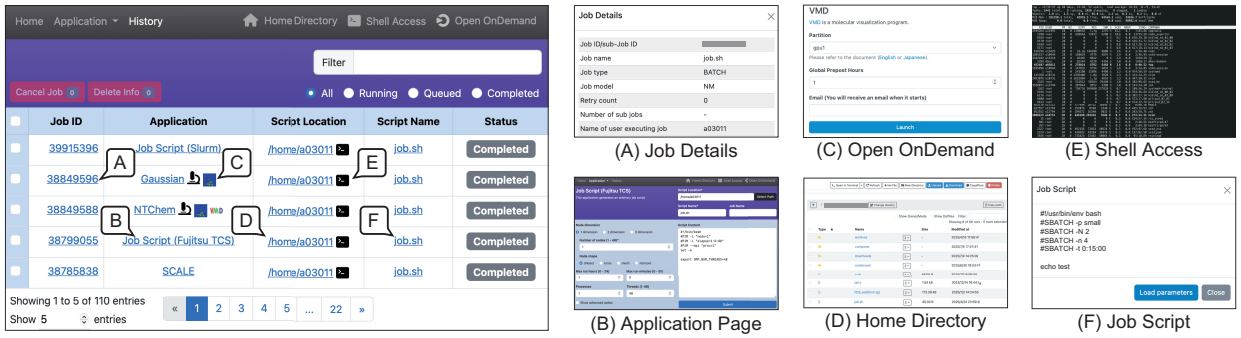

Figure 11: History page on Open Composer

```
1  name: FDS
2  category: Computer Aided Engineering
3  icon: fds.png
4  description: FDS (Fire Dynamics Simulator) is a large-
       eddy simulation code.
```

Figure 12: Sample of `manifest.yml`

Table 2: Real-world applications in our system

Category	Application
Climate	SCALE
Computer Aided Engineering	FDS, FFVHC-ACE, FrontFlow (blue/X), FrontISTR, OpenFOAM (Foundation/OpenCFD)
Condensed Matter Physics	ALAMODE, AkaiKKR, $\mathcal{H}\Phi$, mVMC, OpenMX, PHASE/0, Quantum Espresso, SALMON
Experimental Data Processing	KIERTÄÄ
Molecular Dynamics	GENESIS, GROMACS, LAMMPS, MODYLAS PIMD
Quantum Chemistry	ABINIT-MP, Gaussian, NTChem, SMASH
Quantum Simulation	braket

and script sections are required fields that define the bottom-left and bottom-right areas of Fig. 10, respectively. The header section defines the top-right area of Fig. 10, and a default header section is used if it is omitted.

The HPC cluster we manage has real-world applications installed, as shown in Table 2. In collaboration with the developers and users of each application, we developed the widgets required for the Open Composer web form, as listed below. These widgets can be used in both the header and form sections.

number Enter a numeric value
text Enter a text string
email Enter an email address
select Select a single item from a select box
multi_select Select multiple items from a select box
radio Select a single item from radio buttons
checkbox Select multiple items from checkboxes
path Select a file or directory path on the Open Composer server

For detailed information about the configurable options available for each widget, please refer to the manual available at [8].

4.2 Example of Configuration

Fig. 13 shows the form file that generates the application page in Fig. 10. Lines 1 to 35 define a form section, where keys such as `partition`, `npt`, `time`, `version`, and `input_file` correspond to individual input fields in the web form.

Lines 2 to 11 define a select widget. The `label` key specifies the label displayed in the web form, and the `options` key lists the selectable menu items. The first argument (e.g., "Small partition") is the display name, and the second argument (e.g., "small") is used to replace `#{partition}` in line 39 of the script section. The third and later arguments are explained in Section 4.3.

Lines 12 to 18 define a number widget that contains three numeric input fields. The `size` key specifies the number of input fields, and the `value`, `min` and `max` keys specify the initial, minimum and maximum values for each field, respectively, using array format. This is equivalent to writing HTML such as `<input type="number" value="1" min="1" max="4">` for the first element of the array. When the "Submit" button located below Fig. 10 is clicked, the input values are validated. When the value of `size` is 2 or greater, variables to be replaced in the script section should be expressed by appending an underscore and an index to the key, such as `#{KEY_n}`. The same rule applies to lines 19 to 25.

Lines 26 to 32 define a radio button widget. This widget is functionally similar to the `select` widget defined in lines 2 to 11, but it includes an optional `direction` key to specify the layout orientation of the radio buttons. By default, the buttons are arranged vertically; however, setting the value to `horizontal` changes the layout to a horizontal arrangement.

Lines 33 to 35 define a `path` widget. Clicking the `Select Path` button for this widget in Fig. 10 opens the modal window shown in Fig. 14, where users can select a directory or file path.

Lines 37 to 46 define a script section, which provides the functions `dirname(FILE_PATH)` and `basename(FILE_PATH)`, used on lines 45 and 46, respectively. Given a path that includes both a directory and a file name, `dirname(FILE_PATH)` returns the directory portion, while `basename(FILE_PATH)` returns only the file name.

Lines 48 to 51 define a check section, which performs detailed error checks that cannot be handled in the form section. For example, lines 19 to 25 create a widget for entering time values. If

```yaml
form:
  partition:
    widget: select
    label: Partition
    options:
      - [Small partition, small]
      - [Large partition, large,
         set-label-npt_1: Nodes (5-32), set-value-npt_1:\
         5, set-min-npt_1: 5, set-max-npt_1: 32, set-\
         label-npt_2: Procs (5-512), set-value-npt_2: 5,
         set-min-npt_2: 5, set-max-npt_2: 512]
  npt:
    widget: number
    size: 3
    label: [Nodes (1-4), Procs (1-64), Threads (1-16)]
    value: [1, 1, 16]
    min: [1, 1, 1]
    max: [4, 64, 16]
  time:
    widget: number
    size: 2
    label: [Max time (0-24 h), Max time (0-59 m)]
    value: [ 1, 0]
    min: [ 0, 0]
    max: [24, 59]
  version:
    widget: radio
    direction: horizontal
    label: FDS version
    options:
      - ["6.9.1"]
      - ["6.8.0"]
  input_file:
    widget: path
    label: Input file

script: |
  #!/usr/bin/env bash
  #SBATCH -p #{partition}
  #SBATCH -N #{npt_1}
  #SBATCH -n #{npt_2}
  #SBATCH -t #{time_1}:#{time_2}:00
  spack load fds@#{version}
  export OMP_NUM_THREADS=#{npt_3}
  cd #{dirname(input_file)}
  mpiexec fds_mpi #{basename(input_file)}

check: |
  if @time_1.to_i == 24 && @time_2.to_i != 0
    halt 500, "Exceeded Time"
  end
```

Figure 13: `form.yml` for Fig. 10

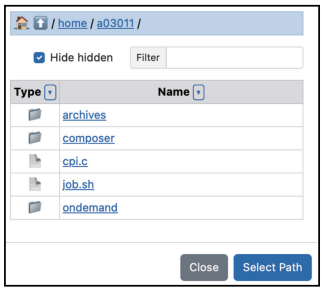

Figure 14: Modal of `path` widget

the scheduler allows a maximum of 24 hours, then entering 24

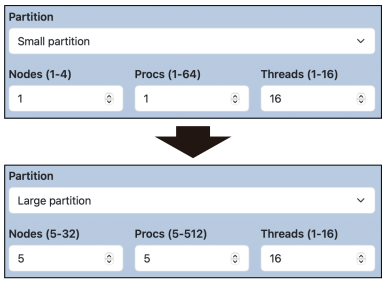

Figure 15: Update values in widget

```yaml
form:
  partition:
    widget: radio
    label: Partition
    options:
      - [GPU partition, gpu]
      - [Normal partition, small, disable-cuda]
  cuda:
    widget: number
    label: CUDA version (10, 11, 12)
    value: 12
    min: 10
    max: 12
```

Figure 16: Sample code of disabling widget

in the first (hours) field means that only 0 can be accepted in the second (minutes) field. However, the widget can only define the maximum value for each individual field and cannot enforce such interdependent constraints. Therefore, the check section uses Ruby language to implement more sophisticated validations. When the "Submit" button shown in Fig. 10 is clicked, the Ruby script defined in the check section is executed before the job is submitted to the scheduler.

4.3 Dynamic Form Widget

Open Composer provides a feature called the "Dynamic Form Widget", inspired by a feature of the same name in Open OnDemand. To enable the creation of more flexible and adaptive web forms, this feature allows certain actions such as changing values, disabling, or hiding other widgets when a specific option is selected in a widget with selectable options. It can be used with the select, radio, and checkbox widgets.

First, we explain how to change values, as shown in lines 8 to 11 of Fig. 13. Starting from the third argument in the `options` key, commands such as the following can be specified:

`set-(label|value|min|max)-(KEY):(VALUE)`

Following `set-`, specify the item and the key to update, followed by the `VALUE` that defines the new content. Fig. 15 shows an example. When "Large partition" is selected from the select box, the labels, initial values, and the minimum and maximum values of the numeric input fields defined in lines 15 to 18 are updated accordingly.

Next, we explain how to disable widgets using Fig. 16 and Fig. 17. In Fig. 16, lines 2 to 7 define a radio widget, and lines 8 to 13 define a number widget. The third element in line 7 indicates that the

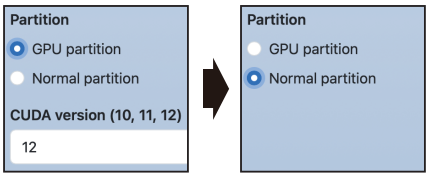

Figure 17: Disabled widget

```
1  script: |
2    pix_size = #{pix_size}
3    center = #{center}
4    filter = #{filter}
5
6  submit: |
7    #!/usr/bin/env bash
8    cd #{OC_SCRIPT_LOCATION}
9    mv #{OC_SCRIPT_NAME} ct.conf
10   genjs_ct ct.conf > #{OC_SCRIPT_NAME}
```

Figure 18: A part of `form.yml` for KIERTÄÄ

widget named cuda will be disabled. As shown in Fig. 17, when the `Normal` partition is selected in the radio widget, the cuda widget is disabled.

This paper omits the explanation of the widget-hiding feature, but its usage is the same as that of the disabling feature. Note that the difference between disabling and hiding a widget lies in how its value is treated in the script section. When a widget is disabled, its value does not appear in the script section. In contrast, when a widget is hidden, its value still appears in the script section. This behavior makes the widget-hiding feature particularly useful for forms with many input fields, where advanced or optional parameters can be hidden while still retaining their values for internal processing.

4.4 Preprocessing function

This section describes the preprocessing steps specified in the submit section. The steps are executed after the check section and before the job is submitted. One intended use case for this preprocessing is to generate a job script from a parameter file, where the parameter file itself is created based on the content defined in the script section.

We explain this feature using the example shown in Fig. 18, which is a part of the form file used in the real-world application KIERTÄÄ. In this example, the script section does not contain a job script, but rather a parameter file used by KIERTÄÄ. In the submit section, `OC_SCRIPT_LOCATION` and `OC_SCRIPT_NAME` indicate the script's location and filename, respectively. In line 9, the parameter file defined in lines 1 to 4 is renamed to `ct.conf`. Then, line 10 shows that the conversion script `genjs_ct` provided by KIERTÄÄ generates a job script from `ct.conf`. Since Open Composer submits the file specified by `OC_SCRIPT_NAME` to the job scheduler after preprocessing, the generated job script should be renamed accordingly.

5 Conclusion

This paper presented the development of Open Composer, a web application that enables users to generate and submit job scripts. Open Composer aims to reduce the learning curve associated with job schedulers while maintaining flexibility in script creation by combining structured web forms with an editable text area. In addition to script generation, it provides job management capabilities, including a history page for reusing previously submitted scripts and features such as job cancellation and status monitoring. Since Open Composer runs on Open OnDemand, it is easy to deploy in existing HPC environments and provides convenient one-click access to various Open OnDemand features. Through these capabilities, Open Composer simplifies the job submission process in HPC clusters and enhances overall user productivity.

To further enhance its usability and effectiveness in HPC clusters, we plan to extend Open Composer with several advanced features. One potential direction is the implementation of a project management feature, similar to the Project Manager described in Section 2.2. This would allow users to organize and manage related job scripts, parameter files, and results within dedicated project workspaces. In addition, we aim to support collaborative features such as sharing job templates or project settings among users or research groups. This could promote reproducibility and improve the efficiency of group workflows. Another promising enhancement is the integration of AI-based assistance for automatic parameter suggestion. This would help users generate job scripts more efficiently, especially for complex applications with many configurable parameters. Finally, integrating workflow tools into Open Composer would streamline the execution of complex, multi-step processes.

References

[1] Blake Mizerany. 2025. Sinatra. https://sinatrarb.com.
[2] Brigham Young University. 2025. BYU Job Script Generator. https://byuhpc.github.io/BYUJobScriptGenerator/.
[3] Dave Hudak et al. 2018. Open OnDemand: A web-based client portal for HPC centers. *Journal of Open Source Software* 3, 25 (2018), 622. doi:10.21105/joss.00622
[4] George Mason University. 2025. Slurm Script Generator. https://wiki.orc.gmu.edu/mkdocs/slurm_generator/slurm_script_generator.html.
[5] Nakao Masahiro and Keiji Yamamoto. 2023. Introducing Open OnDemand to Supercomputer Fugaku. In *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis* (Denver, CO, USA) *(SC-W '23)*. Association for Computing Machinery, New York, NY, USA, 720–727. doi:10.1145/3624062.3624150
[6] National Energy Research Scientific Computing Center. 2025. Jobscript Generator. https://my.nersc.gov/script_generator.php.
[7] Pennings Marinus et al. 2025. Generating Scientific Workflows With Drona Environments. In *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration (PEARC '25)*. Association for Computing Machinery, New York, NY, USA, Article 39, 5 pages. doi:10.1145/3708035.3736104
[8] RIKEN Center for Computational Science. 2025. Open Composer GitHub repository. https://github.com/RIKEN-RCCS/OpenComposer.
[9] Texas A&M High Performance Research Computing. 2025. Drona Composer. https://hprc.tamu.edu/kb/User-Guides/Portal/Drona_composer/.
[10] Texas A&M High Performance Research Computing. 2025. Drona Workflow Engine GitHub repository. https://github.com/tamu-edu/dor-hprc-drona-composer.
[11] University of Michigan. 2025. Job script generator for Slurm. https://websites.umich.edu/~greatlakes/jobscript_generator/.
[12] University of Virginia. 2025. Slurm Script Generator. https://www.rc.virginia.edu/userinfo/hpc/slurm-script-generator/.