東北大学工学部　卒業論文

ウェブインタフェースを介した
スーパコンピュータ利用環境に関する研究

機械知能・航空工学科　滝沢研究室

谷澤悠太

（令和 6 年 3 月）

目次

図目次

表目次

コード目次

第 1 章　緒論

1.1　背景

近年，高性能計算 (High Performance Computing，HPC) システムの用途は多様化し，専門知識を持たない利用者が容易に HPC システムを利用する需要が高まっている．一般的に，コマンド操作に基づいて HPC システムを操作する利用環境や利用する HPC システムごとに異なる操作方法により，HPC を専門としない研究者は HPC システムを使いこなすために多くの学習時間を費やす必要がある．実際に Ping らは，学習目的で HPC システムを初めて利用する学生などは HPC システム利用環境の構築に多くの時間を費やしてしまい，本来の目的である学問のための HPC システムの利用を達成するまでに多大な時間を費やしてしまうという問題を指摘している [1]．そこで，従来のコマンド操作に基づく利用環境や，システムごとに異なる利用方法を利用者から隠蔽し，ウェブブラウザを用いて容易かつ統一的に HPC システムを利用することが可能なウェブインタフェースの研究開発が行われている [2]．

現在用いられているジョブスケジューラには数多くの種類が存在し，今後も多くのジョブスケジューラが開発されることが予想される．そのため，ウェブインタフェースはより多くのジョブスケジューラに対応することが求められる．しかし，既存のウェブインタフェースが新たなジョブスケジューラへの対応をするたびに，開発者はウェブインタフェース本体を改修する必要がある．そのため，ウェブインタフェースの保守性に問題があるといえる．

1.2　目的

本研究では，スーパーコンピュータ利用環境において，ジョブスケジューラの多様化に伴い発生し得るウェブインタフェースの保守性の問題に着目する．そこで，既存のウェブインタフェースの機能を以下の 2 つの機能に分離する．1 つは，ユーザがウェブブラウザ上で HPC システムを利用することを可能とする機能 (ウェブ機能) である．もう 1 つは，多様なジョブスケジューラを統一的に取り扱う機能 (スケジューラ抽象化機能) である．既存のウェブインタフェースをウェブ機能とスケジューラ抽象化機能に分離することで，新たなジョブスケジューラに対応させる際のウェブインタフェースの改修箇所はスケジューラ抽象化機能のみとなる．そのため，提案手法を用いて，ウェブインタフェースの保守性の問題を解決することを目的とする．具体的には，既存のウェブインタフェースの機能を分離し，実際のシステムに実装する．実装における動作の確認を行い，提案手法の実現可能性を示す．さらに，既存のウェブインタフェースと提案したウェブインタフェースのジョブ投入時の

オーバヘッドを定量的に評価することで提案手法の有用性を示す．

1.3 本論文の構成

本論文は全 5 章から構成される．第 1 章では，本研究の背景と目的について述べた．第 2 章では，関連研究について説明し，既存のウェブインタフェースについて述べる．第 3 章では，ウェブインタフェースを介した HPC システム利用環境について説明し，提案手法の実装と動作の確認を行う．第 4 章では，実装した提案手法の評価結果を示し，その考察を行う．第 5 章では，本研究の結論と今後の課題を述べる．

第 2 章　関連研究

2.1　緒言

本章では関連研究について述べる．はじめに，一般的な HPC システムの利用方法について説明する．続いて，関連研究である Open OnDemand と呼ばれるウェブインタフェースについて説明する．ウェブインタフェースの機能やその設計について説明し，既存のウェブインタフェースの利点を整理する．最後に，既存のウェブインタフェースにおける課題を述べる．

2.2　HPC システム利用方法

HPC システムとは，コンピュータクラスタの能力を利用して，デスクトップ型コンピュータやノートブック型コンピュータを遥かに凌ぐ速度で計算課題 (ジョブ) を処理し，実行するシステムを指す．このような計算能力の集約によって，様々な科学分野において他の方法では対処できない大きなジョブを処理することができる．実際に，平均的なデスクトップ型コンピュータは毎秒数十億の浮動小数点演算を実行できる．しかし，HPC システムでは，1 秒に約数千兆の浮動小数点演算を実行可能であることが知られている．例えば，スーパーコンピュータ京は 1 秒間に約 1 京回，スーパーコンピュータ富岳は 1 秒間に約 44 京回の浮動小数点演算を行うことができる [3, 4]．そのため，大規模な計算に対して，HPC システムの利用は効果的であるといえる．

一般的な HPC システムの利用の流れを図 1 に示す．HPC システムは数種類のサーバやデータベースから構成される．HPC システムには，ジョブを実行するために計算を行う計算サーバ (ワーカーノード)，ワーカーノードを管理するためのジョブスケジューラが配備されたジョブ管理サーバ (マスターノード)，ユーザ情報が保存されたデータベースと連携してログイン情報の管理やユーザリクエストの受け渡しを行うログイン用サーバ，実行するジョブの入出力ファイルなどが保存されたファイル管理用サーバなどが存在する．

また，ジョブの実行を依頼するためにはジョブスクリプトファイルを作成する必要がある．ジョブスクリプトファイルとは，ジョブ投入用のシェルスクリプトファイルを指す．ジョブスクリプトファイルは，利用する計算機のリソースや環境を指定する部分と計算機に実行させる処理を記述する部分から構成される．コード 1 では SLURM スケジューラのジョブスクリプトファイルの例を示す．1 行目はシェバンと呼称され，ファイルに記載されたプログラムが/bin/bash で実行されるということを明示的に記載している．2〜6 行目では，各ジョブスケジューラごとに定義されている接頭辞 (Slurm の場合は#SBATCH) を用

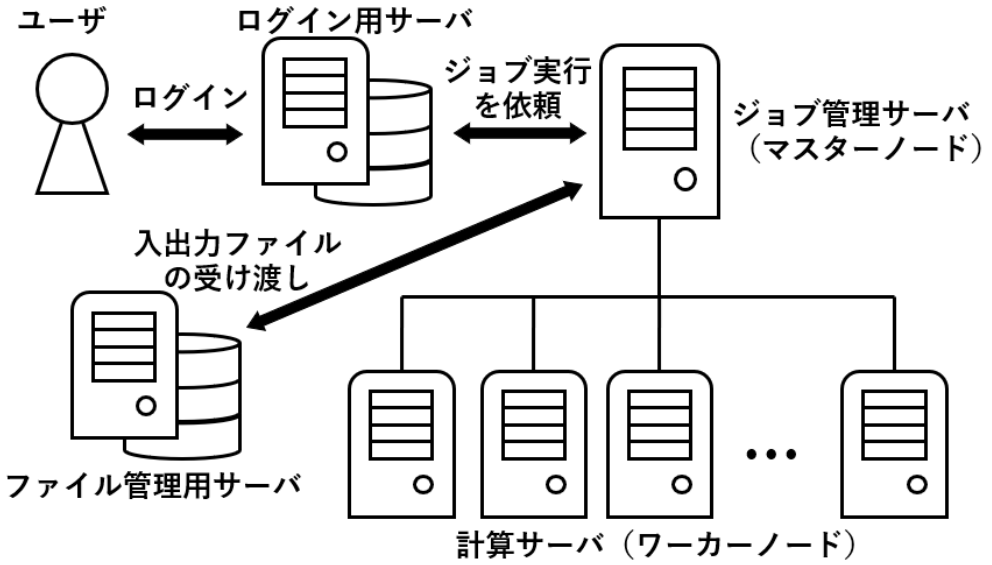

図 1: 一般的な HPC システムの模式図

いて，利用する計算機のリソースや環境を指定する．2 行目では，ジョブを投入するキューの名前を指定している．3 行目では，ジョブで使用するプロセス数を指定している．4 行目では，実行するジョブの名前を指定している．5 行目と 6 行目では，標準出力ファイルと標準エラー出力ファイルを指定している．なお，%J はジョブ ID に変換される．また，7 行目は計算機に実行させる処理を記述する部分であり，プログラム (a.out) を実行する．

ユーザは利用したい HPC システムを遠隔で操作するために自身のコンピュータからユーザ情報を用いて秘密鍵の登録を行った後，ログイン用サーバに SSH 接続を用いてログインする．そして，ユーザは利用するジョブスケジューラの種類に応じた形式でジョブスクリプトファイルを作成する．その後，ユーザは各ジョブスケジューラごとに異なるジョブの実行を依頼するコマンドを用いて，マスターノード上で動作しているジョブスケジューラにジョブの実行を依頼する．ジョブスケジューラはファイル管理用サーバから入力ファイルを受け取る．ワーカーノードはマスターノードの命令に従ってジョブの実行を行う．ジョブ実行時の標準出力，エラー標準出力はファイル管理用サーバに出力され，ジョブ実行後に実行結果を確認することができる．

コード 1: Slurm のジョブスクリプトファイル作成例

```
1 #!/bin/bash
2 #SBATCH -p partition1
3 #SBATCH -n 1
4 #SBATCH -J example
5 #SBATCH -o stdout.%J
6 #SBATCH -o stderr.%J
7
```

2.3 Open OnDemand

代表的な関連研究として，Open OnDemand (OOD) とその機能や設計構成を紹介する [5, 6]．OOD は米国オハイオ・スーパーコンピューティングセンターが開発したオープンソースソフトウェアであり，ウェブインタフェースを介して HPC システムを利用できる環境を提供する．ユーザはウェブブラウザ上で HPC システムを簡単に操作することができ，プラグインやほかのソフトウェアのインストールや設定は不要である．また，リモートデスクトップや Jupyter Notebook[7]，Visual Studio Code[8] などの対話的アプリケーションもウェブブラウザ上から利用することができる．OOD は世界的に使われている様々なジョブスケジューラ (PBS Pro[9]，Slurm[10]，Grid Engine[11]，Torque[12]，LSF[13] など) に対応しているため，システム間の利用方法の差異を隠蔽している．

初めに，OOD の機能について説明する．OOD は，図 2 に示すダッシュボードと図 3 に示すユーザのホームディレクトリを管理する画面，図 4 に示すジョブの状態を確認する画面，図 5 に示すジョブの管理を行う画面，図 6 に示す HPC システムのシェル操作を行う画面，開発者が任意の追加アプリケーションソフトウェアを導入することができる Interactive Apps 画面から構成される．図 3 に示すホームディレクトリ画面では，ユーザのディレクトリを視覚的に操作することができ，ファイルやディレクトリの削除，追加，および編集なども容易に行うことができる．図 4 にはジョブの状態確認を行う Active Jobs 画面を示す．ユーザは投入したジョブの状態をリアルタイムで確認することができる．図 5 の Job Composer 画面では，ユーザはジョブの作成，投入，削除などをすべてこの画面から行うことができ，HPC システム内のジョブの管理を行うことができる．図 6 には，シェル画面を示す．ユーザは連携した HPC システムのシェルをウェブブラウザ上から操作することができる．このように，OOD は様々な機能やアプリケーションソフトウェアと連携して HPC システムの利用者支援を行っている．

続いて，OOD の設計について説明する．なお，OOD の実装言語は Ruby である．OOD は現在多様なジョブスケジューラに対応しているが，各ジョブスケジューラへの対応は adapters ディレクトリ下に配置される．そこで，OOD のジョブスケジューラ対応部分のクラス図を図 7 に示す．指定したジョブスケジューラに対応するために，各ジョブスケジューラ用の Adapter スーパークラスのサブクラスが宣言される．サブクラス内部ではジョブの投入を行う submit メソッド，クラスタの情報を取得する cluster_info メソッド，ユーザ情報を取得する accounts メソッド，ジョブの情報を取得する info メソッド，info_all メソッド，info_where_owner メソッド，ジョブの状態を取得する status メソッド，ジョブの一時停

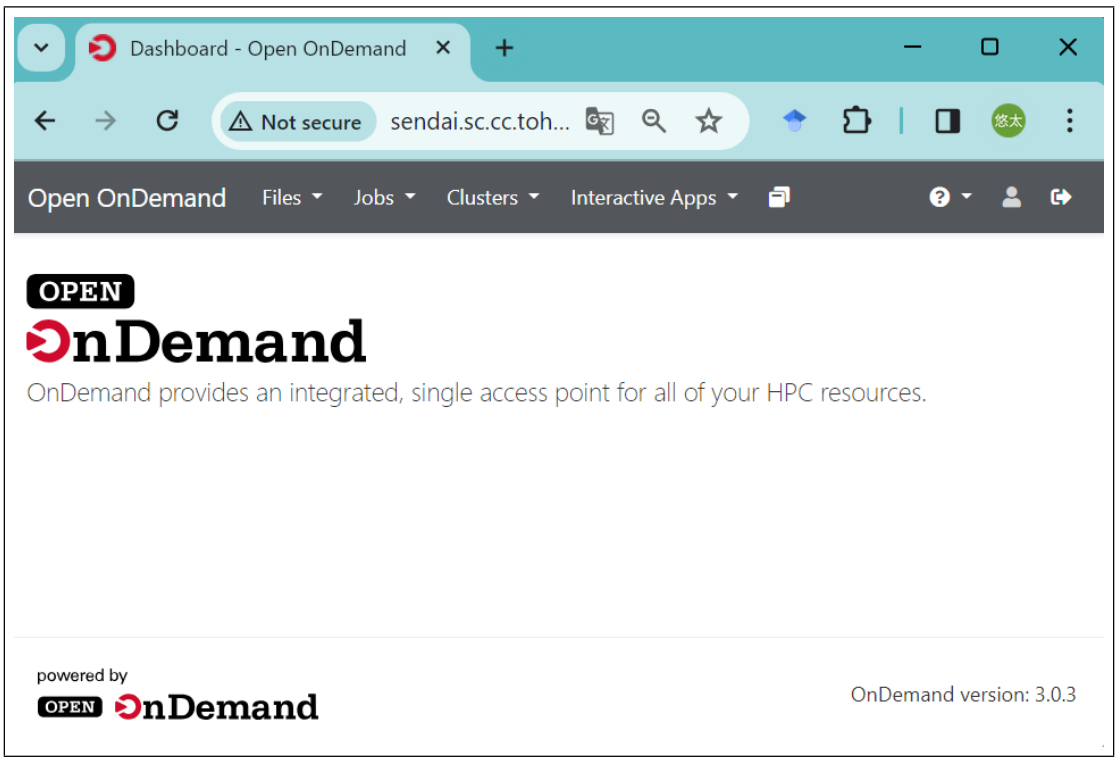

図 2: ダッシュボード画面

止を行う hold メソッド，ジョブの再開を行う release メソッド，ジョブの削除を行う delete メソッドなどが再定義されている．また，各ジョブスケジューラサブクラス内には Batch クラスが定義されており，前述したサブクラス内部のメソッドは Batch クラス内部のメソッドを呼び出すことで実装されている．例えば，Adapter スーパークラスのサブクラスである Slurm クラスは，Slurm クラスタにジョブを投入するためのメソッドである submit メソッドを再定義しており，Slurm クラスで定義された submit クラスは内部クラスである Batch クラスの submit_string メソッドを呼び出してジョブの投入を行う．このように，利用するジョブスケジューラのサブクラスで再定義されたメソッドを用いることで，OOD は多様なジョブスケジューラに対応することができている．

以上のように，OOD は視覚的かつ簡単な操作を用いて HPC システムを利用することができるという利点を持つ．また，OOD は対応しているジョブスケジューラごとに Adapter クラスのサブクラスが宣言されている．そのため，各 Adapter クラスのサブクラスを新規作成および改修することで，OOD のジョブスケジューラへの対応機能を改修することができる．そのため，多くのコンピューティングセンターで実用化され，OOD のユーザ数も年々増加しており，HPC 利用環境を提供するウェブインタフェースとして多くの研究開発が行われている．

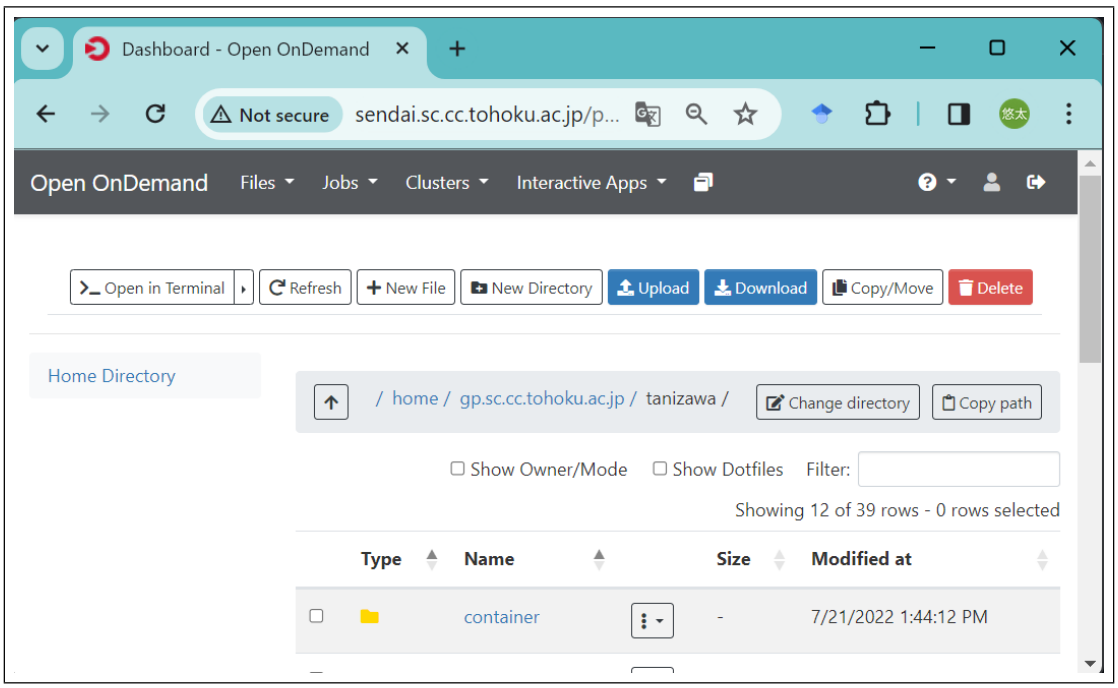

図 3: ホームディレクトリ画面

2.4 既存のウェブインタフェースにおける課題

はじめに，既存のウェブインタフェースである OOD を介した HPC システム利用環境について説明する．従来手法の模式図を図 8 に示す．既存のウェブインタフェースは，各ジョブスケジューラと連携するための機能がウェブインタフェース本体と一体的に実装されている．また，OOD はログイン時に認証機構を必要としており，Dex との OpeID コネクト [14][15] や Shibboleth[16]，CAS[17] などの手法を用いて認証を行う．OOD は，前述した認証方法に必要な外部の認証用ディレクトリと連携してユーザ情報を管理することで，ウェブブラウザ上での安全で快適な HPC システム利用環境を提供する．また，主要なジョブスケジューラの抽象化を行うことで，統一的な操作環境を提供する．

続いて，既存のウェブインタフェースにおける課題について説明する．OOD は視覚的かつ簡単な操作を用いて HPC システムを利用することができるという利点を持っている．しかし，システム設計上の課題も懸念される．

国内でのウェブインタフェースの実装事例として，スーパコンピュータ富岳での OOD の実装事例が挙げられる．OOD は多くの主要なジョブスケジューラに対応しているが，富岳で用いられているジョブスケジューラ (Fujitsu Technical Computing Suite, Fujitsu TCS) に対応していなかったことから，中尾らは OOD を Fujitsu TCS 向けに改修した事例を報告している [18]．改修では Adapter スーパークラスで定義されている下記のメソッドを Fujitsu TCS 用に実装している．

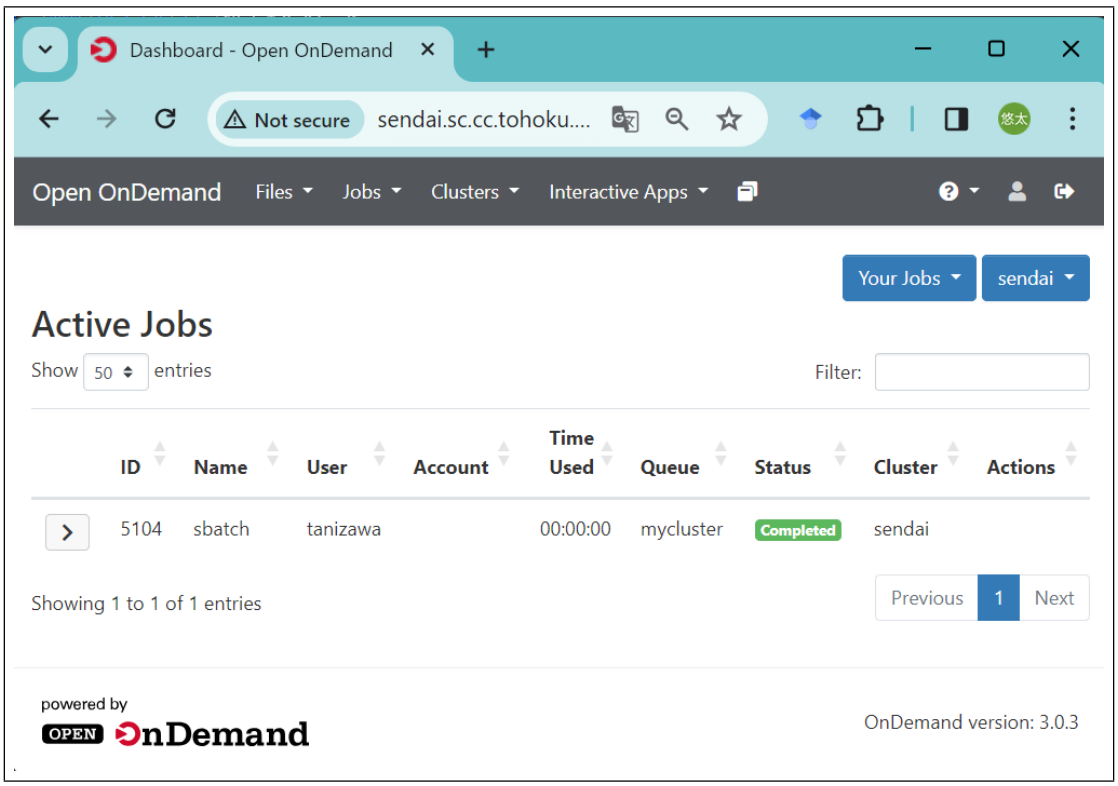

図 4: Active Jobs 画面

submit ジョブの投入

delete ジョブの削除

status ジョブの状態を取得

hold ジョブの一旦停止

release ジョブの再開

info ジョブの情報を取得

info_all 全ジョブの情報を取得

cluster_info HPC システムの情報を取得

supports_job_arrays バルクジョブのサポートの可否

directive_prefix ジョブスケジューラで用いられる接頭辞の取得 (Fujitsu TCS の場合は PJM)

さらに，本来 OOD に対応しているジョブスケジューラでは図 4 に示す Active Jobs 画面の ID 番号の左にあるボタンをクリックすると，ジョブの詳細情報が表示される．しかし，Fujitsu TCS のような OOD に未対応であるジョブスケジューラは，ジョブの詳細が表示されない．そこで中尾らは Fujitsu TCS 用に詳細情報が表示されるように Active Jobs 画面の改修を行ったことも報告している [18].

ほかにも様々なジョブスケジューラが存在し，今後も登場することを考えると，ジョブス

ケジューラの種類が増えるごとに OOD 本体を直接改修する方法では保守性に問題があるといえる．

2.5 結言

本章では，関連研究について述べた．はじめに一般的な HPC システムの利用方法について説明した．HPC システムの利用方法について説明し，基礎的な知識を解説した．その後，Open OnDemand と呼ばれるウェブインタフェースについてその機能と設計について説明した．OOD を用いることによる影響は大きく，HPC システムの利用者に多くの利点をもたらすことができると考えられる．また，内部の設計を述べることで，OOD がジョブスケジューラへ対応する際の改修方法を説明した．最後に，既存のウェブインタフェースについて説明し，国内での OOD の実装例を参考にして課題点を述べた．ユーザに快適な HPC 利用空間を提供するという利点に反して，ウェブインタフェースの保守性が課題として挙げられる．次章では，本章で述べた既存のウェブインタフェースの課題である保守性に関する問題を考慮した手法を提案し，その実装を行う．

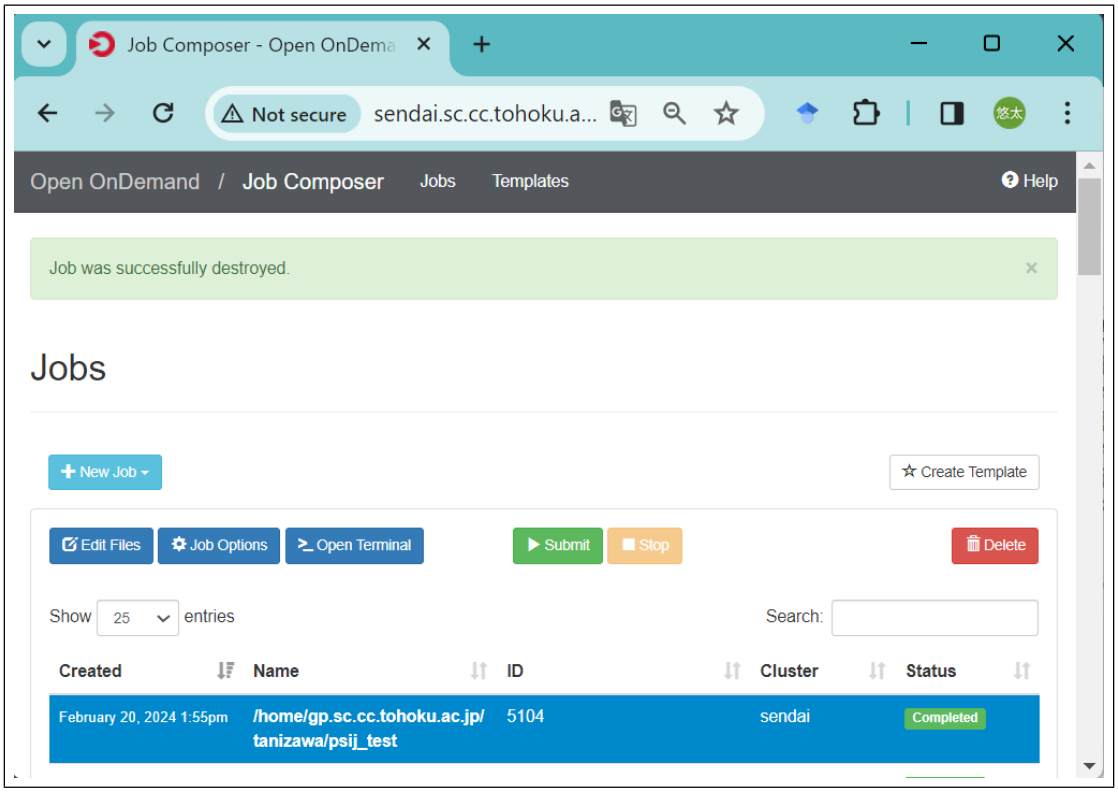

図 5: Job Composer 画面

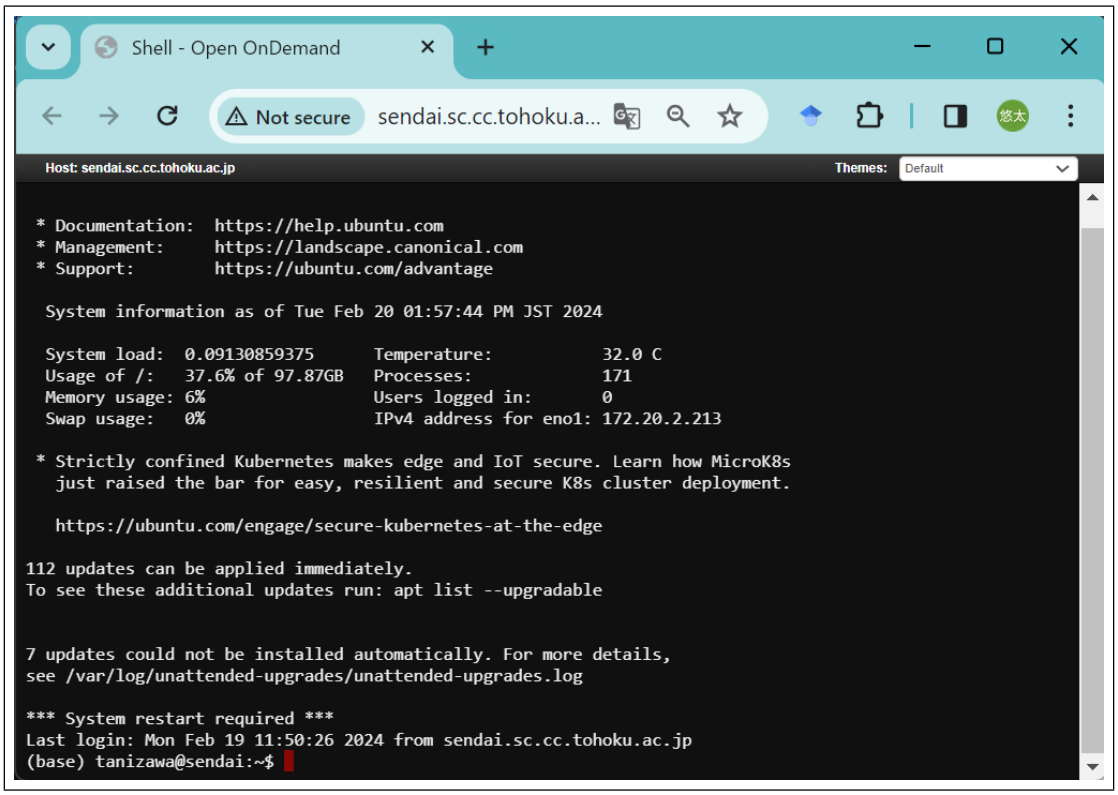
Shell - Open OnDemand
Not secure
sendai.sc.cc.tohoku.a...
Host: sendai.sc.cc.tohoku.ac.jp
Themes:
Default
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
System information as of Tue Feb 20 01:57:44 PM JST 2024
System load: 0.09130859375
Usage of /: 37.6% of 97.87GB
Memory usage: 6%
Swap usage: 0%
Temperature: 32.0 C
Processes: 171
Users logged in: 0
IPv4 address for eno1: 172.20.2.213
* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
just raised the bar for easy, resilient and secure K8s cluster deployment.
https://ubuntu.com/engage/secure-kubernetes-at-the-edge
112 updates can be applied immediately.
To see these additional updates run: apt list --upgradable
7 updates could not be installed automatically. For more details,
see /var/log/unattended-upgrades/unattended-upgrades.log
*** System restart required ***
Last login: Mon Feb 19 11:50:26 2024 from sendai.sc.cc.tohoku.ac.jp
(base) tanizawa@sendai:~$

図 6: シェル画面

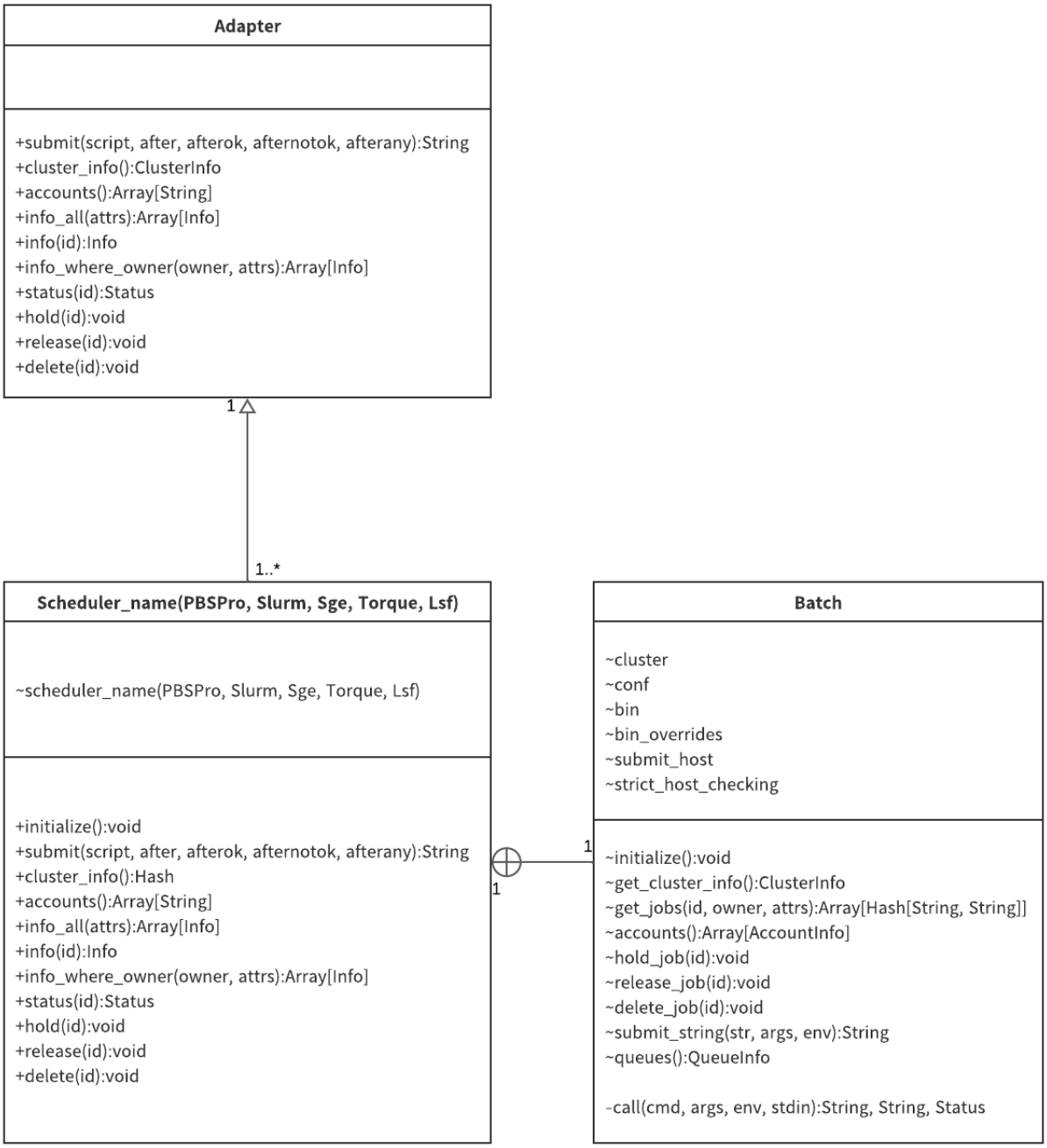
Adapter
+submit(script, after, afterok, afternotok, afterany):String
+cluster_info():ClusterInfo
+accounts():Array[String]
+info_all(attrs):Array[Info]
+info(id):Info
+info_where_owner(owner, attrs):Array[Info]
+status(id):Status
+hold(id):void
+release(id):void
+delete(id):void
1
1..*
Scheduler_name(PBSPro, Slurm, Sge, Torque, Lsf)
~scheduler_name(PBSPro, Slurm, Sge, Torque, Lsf)
+initialize():void
+submit(script, after, afterok, afternotok, afterany):String
+cluster_info():Hash
+accounts():Array[String]
+info_all(attrs):Array[Info]
+info(id):Info
+info_where_owner(owner, attrs):Array[Info]
+status(id):Status
+hold(id):void
+release(id):void
+delete(id):void
1
1
Batch
~cluster
~conf
~bin
~bin_overrides
~submit_host
~strict_host_checking
~initialize():void
~get_cluster_info():ClusterInfo
~get_jobs(id, owner, attrs):Array[Hash[String, String]]
~accounts():Array[AccountInfo]
~hold_job(id):void
~release_job(id):void
~delete_job(id):void
~submit_string(str, args, env):String
~queues():QueueInfo
-call(cmd, args, env, stdin):String, String, Status

図 7: OOD のクラス図

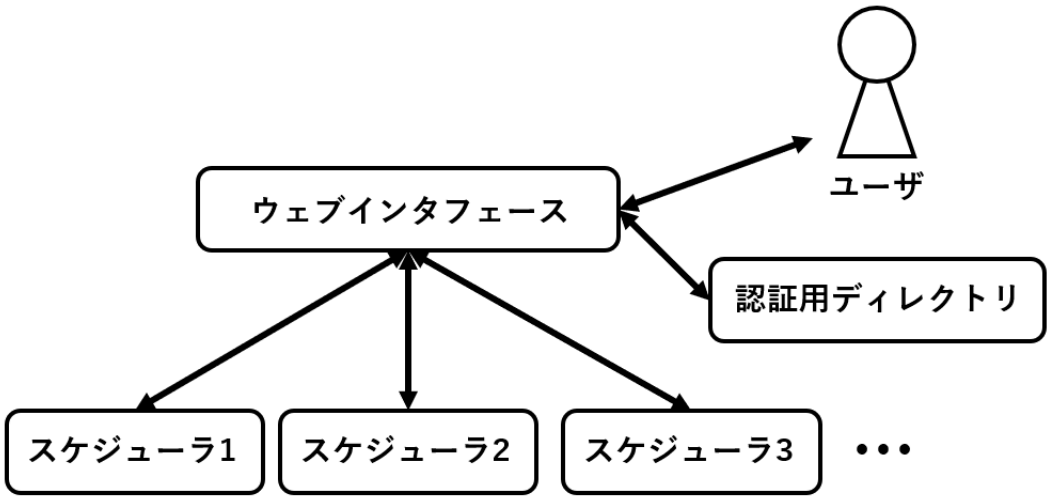

図 8: 従来手法の模式図

第 3 章　ウェブインタフェースを介した HPC システム利用環境

3.1　緒言

前章では，本研究の関連研究について述べた．本章では，ウェブインタフェースを介した HPC システム利用環境の提案手法について説明し，その実装を行う．はじめに，提案手法について説明する．その後，実装の概要，具体的な実装の手順について説明する．

3.2　提案手法

本研究の目的は，ウェブインタフェースに新たなジョブスケジューラを対応させた際に発生し得る保守性の問題を解決するために，既存のウェブインタフェースを以下の 2 つの機能に分離することである．1 つは，HPC 利用環境をウェブインタフェースに提供する機能 (ウェブ機能) である．もう 1 つは，ジョブスケジューラ間の差異を抽象化する機能 (スケジューラ抽象化機能) である．この機能の分離により，それぞれの機能を独立に保守管理できる構成を実現する．このために，本研究ではウェブ機能からは統一的にシステムを利用し，スケジューラ抽象化機能でシステム間の差異を埋める構成の利用環境を提案する．

この提案手法の模式図を図 9 に示す．ウェブ機能では，ユーザはウェブ機能のみとやり取りを行い，ユーザ情報を管理する外部の認証用ディレクトリを用いて安全に HPC システムを利用することができる．スケジューラ抽象化では，ウェブ機能から得られた様々なジョブスケジューラに対する要求をスケジューラ抽象化機能が受け取り，処理を行う．この実現のためには，ウェブ機能とスケジューラ抽象化機能を連携させる必要があることから，両者間に求められる情報のやり取りを整理し，適切な実装方法を検討する．

3.3　実装

3.3.1　実装の概要

ウェブ機能とスケジューラ抽象化機能をそれぞれ独立に実装し，連携させることでウェブインタフェースを介して様々なシステムを統一的に利用できる環境を実現する．そのために，ウェブ機能の基盤として OOD を利用する．さらに，スケジューラ抽象化機能の基盤として PSI/J[19] と呼ばれる Python ライブラリを利用する．PSI/J は複数のジョブスケジューラを統一的に取り扱うことを可能とするライブラリである．両者を組み合わせることで提案手法の実装を行う．

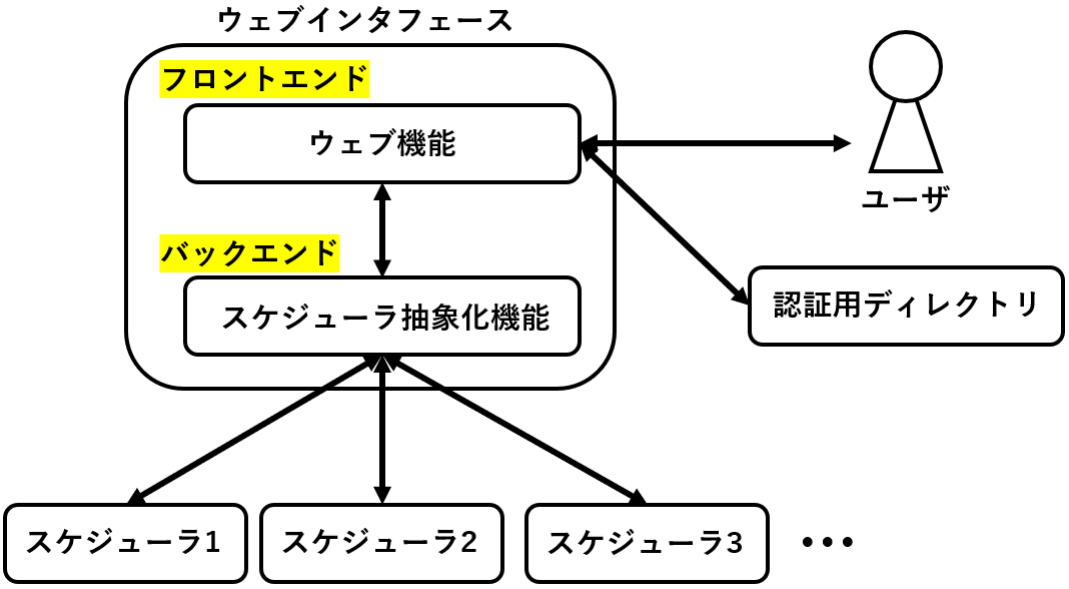

図 9: 提案手法の模式図

本研究では，東北大学のスーパーコンピュータ「AOBA」[20] で運用されているジョブスケジューラ (NEC Network QueuingSystem V, NQSV) [21] が OOD に対応していないという事実に着目して，NQSV をスケジューラ抽象化機能側に実装することと，それをウェブ機能側から利用できることを検証する．

実装環境として，OOD 用のホストサーバとスーパーコンピュータ AOBA を模した HPC クラスタ (疑似 AOBA クラスタ) を考える．疑似 AOBA クラスタの模式図を図 10 に示す．OOD 用のホストサーバでは，OOD の動作が保証されている Ubuntu20.04 LST を OS として用いる．疑似 AOBA クラスタは，マスターノードと二つのワーカーノードから構成される小規模なクラスタであり，AOBA と同様に NQSV をジョブスケジューラとして配備する．疑似 AOBA クラスタでは NQSV の動作確認が行われている CentOS7 を用いる [22]．本研究の実装では，公式が推奨している LDAP サーバを用いた「Dex との OpenID コネクト」を利用して認証を行う [23][24]．OpenID コネクトとは，ユーザ認証用のプロトコルのことを指す．また，Dex とは，OpenID コネクト認証プロバイダーのことを指す．LDAP とは，ユーザ ID やパスワードの管理を行うディレクトリサービスの維持およびアクセスを行うプロトコルである．また，LDAP サーバとは，LDAP に基づいてディレクトリサービスを提供するサーバのことである．

3.3.2 スケジューラ抽象化機能と NQSV の連携

スケジューラ抽象化機能と NQSV の連携を考える．スケジューラ抽象化機能の基盤である PSI/J は，ジョブの情報を格納する Job クラスとジョブの投入や削除などのメソッドを

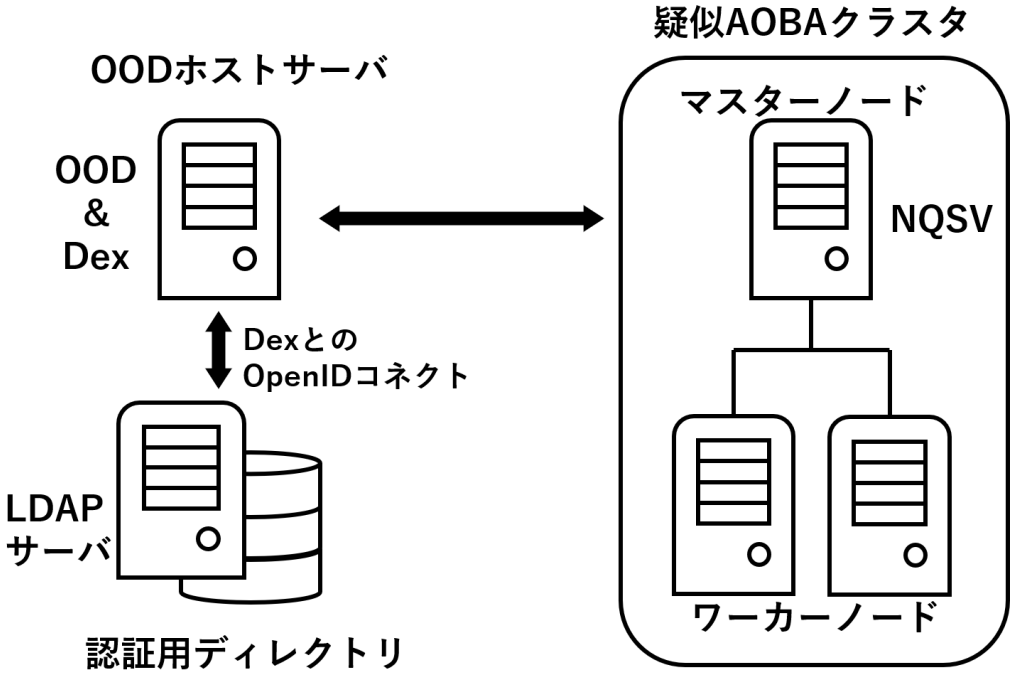

図 10: 実装環境

ジョブスケジューラごとに再定義している JobExecutor クラスにより構成されている．本研究では新たに NQSV 用の JobExecutor クラスを作成し，ジョブの投入，削除，及びジョブの状態確認を行うための 3 つのメソッドを実装する．

はじめに，ジョブの投入を行うメソッドを実装する．実装内容をコード 2 に示す．generate_submit_script メソッドでは，Job クラスのオブジェクトである job，ジョブ実行に関する情報が格納された context，ジョブスクリプトが書き込まれるファイルオブジェクトである submit_file を引数に持ち，generator オブジェクトの generate_submit_script メソッドを用いて投入するジョブスクリプトファイルの作成を行う．また，get_submit_command では，投入するジョブスクリプトファイルの絶対パスを用いて，ジョブ投入用のコマンドを作成する．さらに，job_id_from_submit_output メソッドでは，ジョブ投入時の標準出力が格納されている引数 out からからジョブ ID を抽出している．NQSV のジョブ投入時の標準出力は「Request ジョブ ID.NQSV ホスト名 submitted to queue: キュー名」である．そのため，ジョブ ID を正規表現を用いて抽出し，返り値として出力する (14〜16 行目)．また，ジョブが正常に投入されたことを示すために，11 行目では submit_flag を有効化している．

コード 2: ジョブの投入メソッド

```
1  class NQSVJobExecutor(BatchSchedulerExecutor):
2      def generate_submit_script(self, job: Job,
3                  context: Dict[str, object], submit_file: TextIO) -> None:
4          self.generator.generate_submit_script(job, context, submit_file)
```

```
5
6       def get_submit_command(self, job: Job, submit_file_path: Path)
7                                                               -> List[str]:
8           return ['qsub', str(submit_file_path.absolute())]
9
10      def job_id_from_submit_output(self, out: str) -> str:
11          self.submit_flag = True
12          s = out.strip().split()[1]
13          out = ""
14          match = re.search(r'\b(\d+)\b', s)
15          out = match.group() if match else None
16          return out
```

続いて，ジョブの削除を行うメソッドを実装する．実装内容をコード 3 に示す．get_cancel_command メソッドでは，引数であるジョブ ID が格納された native_id を用いてジョブ削除用のコマンドを作成する．ジョブが削除されたことを示すために，3 行目では cancel_flag を有効化している．

コード 3: ジョブの削除メソッド

```
1   class NQSVJobExecutor(BatchSchedulerExecutor):
2       def get_cancel_command(self, native_id: str)-> List[str]:
3           self.cancel_flag = True
4           return ['qdel', native_id]
```

最後に，ジョブの状態確認を行うメソッドを実装する．PSI/J で用いられてるジョブの状態名とその説明を表 1 に示す．

PSI/J が対応している他のジョブスケジューラ (Slurm[10]，PBS Pro[9]，LSF[13]，Flux[25]，Cobalt[26]) は，ジョブの終了後に COMPLETED 状態，CANCELED 状態，及び FAILED 状態のいずれかの状態であることをジョブの状態を確認するコマンドの出力結果から確認できる．しかし，NQSV ではジョブの終了後に COMPLETED 状態，

表 1: PSI/J で用いられているジョブの状態名

状態名	説明
NEW	ジョブがジョブキュー投入前である状態
ACTIVE	ジョブが実行中であるという状態
QUEUED	ジョブがジョブキュー内で待機中である状態
COMPLETED	ジョブが実行完了した状態
CANCELED	ジョブが削除された状態
FAILED	ジョブの投入が失敗した状態

CANCELED 状態，FAILED 状態を確認できないという仕様上の違いがある．そのため，NQSV に対応するためには，ジョブの投入，ジョブの削除，および待機中のジョブの存在確認に基づいてジョブの状態を PSI/J 側で把握する必要がある．図 11 には，NQSV に対応した場合のジョブの状態遷移図を示す．ジョブが新規作成された瞬間は，ジョブの状態は NEW 状態となる．ジョブの投入を行う qstat コマンドが正常に実行されると，ジョブがジョブキュー内に投入され QUEUED 状態となる．その後，ジョブの実行が始まると ACTIVE 状態に遷移する．NQSV では，qstat コマンドの出力結果により QUEUED 状態と ACTIVE 状態を判別することができる．また，NEW 状態，QUEUED 状態，及び ACTIVE 状態において，「ジョブキュー内にジョブが存在しない」かつ「ジョブの削除を行う qdel コマンドが実行されている」場合は，ジョブの状態を CANCELED 状態に遷移させる．さらに，「ジョブキュー内にジョブが存在しない」かつ「ジョブの削除を行う qdel コマンドが実行されていない」場合は，ジョブの状態を COMPLETED 状態に遷移させる．一方，NEW 状態において，ジョブの投入を行う qsub コマンドが失敗した場合はジョブの状態を FAILED 状態に遷移させる．

具体的な実装をコード 4 に示す．NQSVJobExecutor クラス内で定義されている get_status_now メソッドは Job クラスのインスタンスを引数にとり，ジョブの状態を保持する JobStatus クラスを返り値に持つ (21 行目)．引数として受け取った Job クラスのインスタンス変数 job.native_id からジョブの ID を抽出し，string 型変数 native_ids に代入する (22 行目)．そして，qsub コマンドとそのオプションを用いた「qstat -F rid,stt -n -l native_ids」コマンドを実行して，ジョブの ID とジョブの状態を取得する (23〜25 行目)[27]．コマンドの出力は行ごとに List 型変数の lines に代入し，中身を for ループで走査する (27〜65 行目)．指定した ID のジョブがジョブキューにない場合は，ジョブの ID とジョブの状態ではなく「Batch Request: ジョブ ID does not exist on クラスタのホスト名.」という出力が変数 line に代入されるため，条件文「”does not exist on” in line」が真であれば指定した ID のジョブがジョブキューにないということがわかる．そのため，前述した条件文と cancel 時に有効化される cancel_flag により，ジョブキュー内にジョブがなく，ジョブの削除が正常に行われていることがわかるため，ジョブの状態を CANCELED 状態に遷移させる (33〜39 行目)．一方，条件文が真であり，cancel_flag が無効である場合は，ジョブキュー内にジョブがないが，ジョブの削除が行われていないので，ジョブの状態を COMPLETED 状態に遷移させる (41．47 行目)．また，ジョブの投入時に立つ submit_flag が立っていない場合はそもそもジョブの投入が成功していないのでジョブの状態を FAILED にする (49〜55 行目)．以上の 3 つの場合分けにおいて，「Batch Request: ジョブ ID does not exist on クラスタのホスト名.」という出力からジョブ ID のみを抽出する．抽出したジョブ ID ごとに Dict 型変数 r を用意して，JobStatus クラスのジョブの

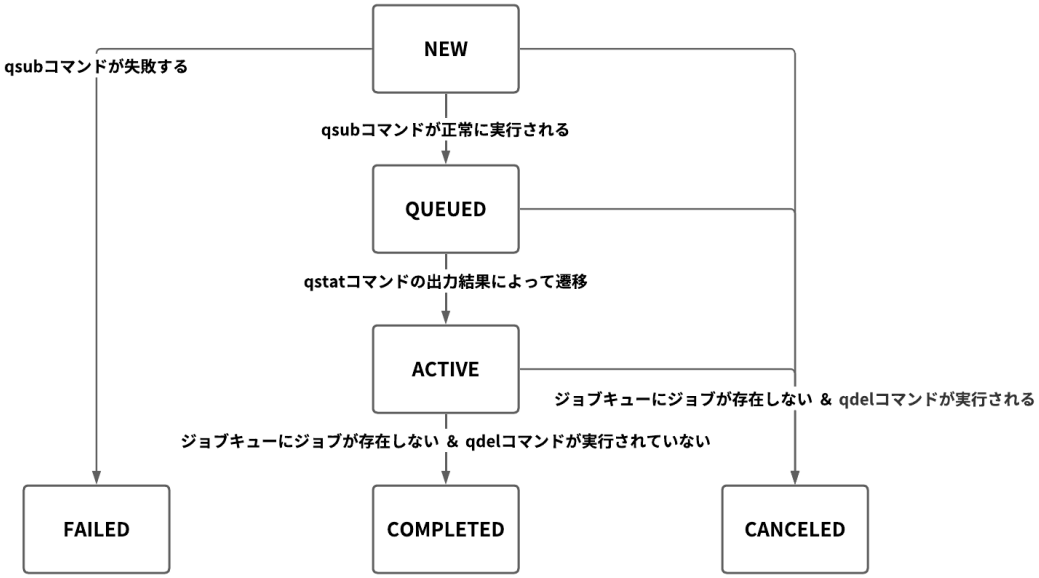

図 11: NQSV 対応時のジョブの状態遷移図

状態情報を代入する．また，前述した 3 種類の場合分けのいずれにも該当しない場合は，ジョブキュー内にジョブが存在する場合であり，ジョブ ID とジョブの状態を変数 cols に代入する (58 行目)．ここで出力されているジョブの状態は NQSV によって定められた状態名でありジョブキューに投入され，実行待ちである状態は QUE，ジョブが実行されている状態は RUN など固有の名称が定められている．これら NQSV 独自のジョブ状態名から PSI/J で用いる各ジョブスケジューラで共通の状態名に変換するために，_STATE_MAP と_get_state メソッドを用いる (3〜16 行目，70〜72 行目)．なお，他のジョブスケジューラではジョブ状態の詳細メッセージが出力される場合があるが，NQSV には詳細メッセージの出力機能がないため message 変数は None としている．

このように PSI/J 自身がジョブの状態を管理することにより，さらに広い範囲のジョブスケジューラに対応することができることから，ジョブスケジューラ抽象化機能の汎用性を高めることができたといえる．

コード 4: ジョブの状態取得メソッド

```
1  class NQSVJobExecutor(BatchSchedulerExecutor):
2
3      _STATE_MAP = {
4          'QUE': JobState.QUEUED,
5          'RUN': JobState.ACTIVE,
6          'WAT': JobState.QUEUED,
7          'HLD': JobState.QUEUED,
8          'SUS': JobState.QUEUED,
9          'ARI': JobState.QUEUED,
```

```
10          'TRS': JobState.QUEUED,
11          'EXT': JobState.ACTIVE,
12          'PRR': JobState.QUEUED,
13          'POR': JobState.ACTIVE,
14          'MIG': JobState.QUEUED,
15          'STG': JobState.QUEUED,
16      }
17
18      def get_status_command(self, native_ids: Collection[str]) -> List[str]:
19          return ['qstat', '-F', 'rid,stt', '-n', '-l'] + list(native_ids)
20
21      def get_status_now(self, job: Job) -> Job.status:
22          native_ids = ''.join(str(job.native_id))
23          command = ['qstat', '-F', 'rid,stt', '-n', '-l', native_ids]
24          out =
25          subprocess.run(command, capture_output=True, text=True).stdout
26          r = {}
27          lines = iter(out.split('\n'))
28
29          for line in lines:
30              if not line:
31                  continue
32
33              if(("does not exist on" in line) and self.cancel_flag):
34                  cols = line.split()
35                  match = re.search(r'\b(\d+)\b', cols[2])
36                  native_id = match.group() if match else None
37                  state = JobState.CANCELED
38                  r[native_id] = JobStatus(state=state, message=None)
39                  return r[native_id]
40
41              elif(("does not exist on" in line) and not(self.cancel_flag)):
42                  cols = line.split()
43                  match = re.search(r'\b(\d+)\b', cols[2])
44                  native_id = match.group() if match else None
45                  state = JobState.COMPLETED
46                  r[native_id] = JobStatus(state=state, message=None)
47                  return r[native_id]
48
49              elif(not(self.submit_flag)):
50                  cols = line.split()
51                  match = re.search(r'\b(\d+)\b', cols[2])
52                  native_id = match.group() if match else None
53                  state = JobState.FAILED
54                  r[native_id] = JobStatus(state=state, message=None)
55                  return r[native_id]
56
57              else:
58                  cols = line.split()
```

```
59                  match = re.search(r'\b(\d+)\b', cols[0])
60                  native_id = match.group() if match else None
61                  native_state = cols[1]
62                  state = self._get_state(native_state)
63                  msg = None
64                  r[native_id] = JobStatus(state=state, message=msg)
65                  return r[native_id]
66
67      def _get_messsage(*args, **kwargs):
68          return None
69
70      def _get_state(self, state: str) -> JobState:
71          assert state in NQSVJobExecutor._STATE_MAP
72          return NQSVJobExecutor._STATE_MAP[state]
```

3.3.3 ウェブ機能とスケジューラ抽象化機能との連携

続いて，ウェブ機能を提供する OOD からスケジューラ抽象化機能を用いることを考える．実装における問題点として，OOD が Ruby で実装されていることに対して，PSI/J は Python で実装されているという点が挙げられる [28][29]．そのため，Ruby スクリプト上で Python ライブラリを使用する必要がある．本実装では，PSI/J を経由する際のオーバヘッドが小さく，単純な実装であるという理由から，PSI/J を用いたジョブの管理用の Python スクリプトをシェルを経由して Ruby スクリプト上から呼び出す手法を用いる．この実装により，ウェブ機能として OOD を用い，スケジューラ抽象化機能である PSI/J を経由して，指定したジョブスケジューラにジョブの投入や削除を行うことができる．また，PSI/J を仲介することで，OOD が未対応であった NQSV でのジョブ管理を OOD 上から操作することを実現している．

実装をコード 5 に示す．PSI/J と連携するためのアダプタファイルは Ood-Core/Job/Adapters モジュール内で定義され，Adapter スーパークラスのサブクラスとして PSIJ クラスを定義する．コード 6 には OOD と PSI/J を接続するための設定ファイル psij.yml の中身を示す．設定ファイルには，OOD からログインを行う際のホスト名，選択する Adapter 名，PSI/J で利用する JobExecutor クラス名，ユーザが一般的に使用するコマンドが置かれたバイナリファイルのパスと利用するジョブスケジューラの設定ファイルのパス，ジョブを投入する HPC システムのホスト名を設定する．initialize メソッド内では，設定ファイル psij.yml から与えられた情報をそれぞれインスタンス変数に代入する (4〜13 行目)．

submit メソッドは，引数でジョブスクリプトの内容を文字列として受け取り，投入したジョブの ID を返す．引数で与えらえれたジョブスクリプトを一時的なジョブスク

リプト保管用のディレクトリに保存する (16〜19 行目)．その後，scp コマンドを用いて，ユーザのホームディレクトリに PSI/J を用いたジョブ投入用の Python スクリプト submit_script.py を転送する (20〜22 行目)．そして，ssh コマンドを用いてジョブ投入先のホストで JobExecutor と投入するジョブのパスを指定して submit_script.py を実行することで，ジョブを選択したホストに投入する (23〜30 行目)．

delete メソッドは，引数でジョブ ID を受け取り，ジョブの削除を行う (33〜34 行目)．submit メソッドと同様に，scp コマンドを用いて，ユーザのホームディレクトリに PSI/J を用いたジョブ削除用の Python スクリプト delete_script.py を転送する (35〜37 行目)．そして，ssh コマンドを用いてジョブ投入先のホストで JobExecutor と削除するジョブの ID を指定して delete_script.py を実行することで，選択したジョブを削除する (38〜43 行目)．

コード 5: PSI/J と OOD の連携

```
1
2  class PSIJ < Adapter
3    class Batch
4      def initialize(cluster: nil, bin: nil, conf: nil, bin_overrides: {},
5                     submit_host: "", strict_host_checking: true, executor: nil)
6        @cluster               = cluster && cluster.to_s
7        @conf                  = conf    && Pathname.new(conf.to_s)
8        @bin                   = Pathname.new(bin.to_s)
9        @bin_overrides         = bin_overrides
10       @submit_host           = submit_host.to_s
11       @strict_host_checking = strict_host_checking
12       @executor              = executor
13     end
14
15   def submit(script, after: [], afterok: [], afternotok: [], afterany: [])
16     job_path = "/Temporary/Path/to/run.sh"
17     file = File.open(job_path, "w")
18     file.puts(script.content.to_s)
19     file.close
20     scp_command = "sshpass -p 'password' scp /Path/to/submit_script.py
21                    username@#{@psij.submit_host}:/Path/to/submit_script.py"
22     system(scp_command)
23     psij_submit =
24     "python3 /Path/to/submit_script.py"
25                             + " " + @psij.executor + " " + job_path
26     ssh_submit =
27     "sshpass -p 'password'
28                    ssh username@#{@psij.submit_host} '#{psij_submit}'"
29     o, e, s = Open3.capture3(ssh_submit)
30     return o
31   end
32
```

```
33    def delete(id)
34      ids = id.to_s
35      scp_command = "sshpass -p 'password' scp /Path/to/delete_script.py
36                     username@#{@psij.submit_host}:/Path/to/delete_script.py"
37      system(scp_command)
38      psij_delete =
39      "python3 /ood_tmp/delete_script.py" + " " + @psij.executor + " " + ids
40      ssh_delete =
41      "sshpass -p 'password'
42                     ssh username@#{@psij.submit_host} '#{psij_delete}'"
43      system(ssh_delete)
44    end
```

コード 6: OOD と PSI/J の接続ファイル

```
1  v2:
2      metadata:
3        title: "PSIJ"
4      login:
5        host: "host adress"
6      job:
7        adapter: "psij"
8        bin: /path/to/bin
9        conf: /path/to/nqsv.conf
10       executor: "nqsv"
11       submit_host: "submit_host adress"
```

3.4 結言

本章では，ウェブインタフェースを介した HPC システム利用環境の提案手法について説明し，その実装を行った．はじめに，提案手法の概要を説明した．本研究では，ウェブインタフェースをウェブ機能とスケジューラ抽象化機能をそれぞれ独立して実装することで，新たなジョブスケジューラに対応する際の保守性に関する問題を解決することを目指す．その後，実装の概要，具体的な実装の手順について説明した．ウェブ機能の基盤として OOD，スケジューラ抽象化機能の基盤として PSI/J を用いる．両者を連携させ，ウェブ機能からスケジューラ抽象化機能を介して NQSV を利用できるように実装を行った．次章では，本章で実装した提案手法の評価を行い，提案手法の実現可能性と有用性を考察する．

第 4 章　実装評価

4.1　緒言

本章では，前章で実装した提案手法を用いて，評価を行った結果について説明する．はじめに，評価環境と評価条件について説明する．その後，提案手法の実装において考慮すべき実行時のオーバヘッドの測定結果について，定量的に説明する．

4.2　動作確認

4.3　評価環境

はじめに，評価環境について説明する．ウェブインタフェースをウェブ機能とスケジューラ抽象化機能に分離したことによって両者の連携時のオーバヘッドの発生が懸念されるため，その影響を定量的に評価する．本実装においては，OOD がシェルを介して PSI/J の Python スクリプトを実行するため，そのオーバヘッドによる影響を評価する．

評価には，Selenium[30] と呼ばれるウェブブラウザの自動制御ライブラリを用いる．機能の分離前と分離後を比較したいため，OOD が NQSV に対応していないことから評価環境として疑似 AOBA クラスタを用いることはできない．そこで，実験用に用意した Slurm クラスタを用いて機能分離に伴うオーバヘッドの評価を行う．Slurm クラスタのマスターノードとワーカーノードの詳細情報を表 2，表 3 に示す．

評価環境を図 12 に示す．ローカルホストから OOD ホストサーバのウェブブラウザにアクセスして，ユーザ名とパスワードを用いてログインする．ダッシュボード画面から Job Composer 画面に移動し，New Job ボタンを押下する．選択肢の中から From Specified Path を選択してジョブが配置されたディレクトリ，ジョブスクリプトのファイル名，実行するクラスタ名を選択して，ジョブの作成と投入を行う．投入したジョブの状態は squeue

マスターノード	
CPU	Intel Xeon W-2102
メモリ容量	512GB
コア数	4
OS	Ubuntu 22.04.2 LST
RAM	16GB

表 2: マスターノード

ワーカーノード	
CPU	Intel Xeon E-2378
メモリ容量	512GB
コア数	8
OS	Ubuntu 22.04.2 LST
RAM	64GB

表 3: ワーカーノード

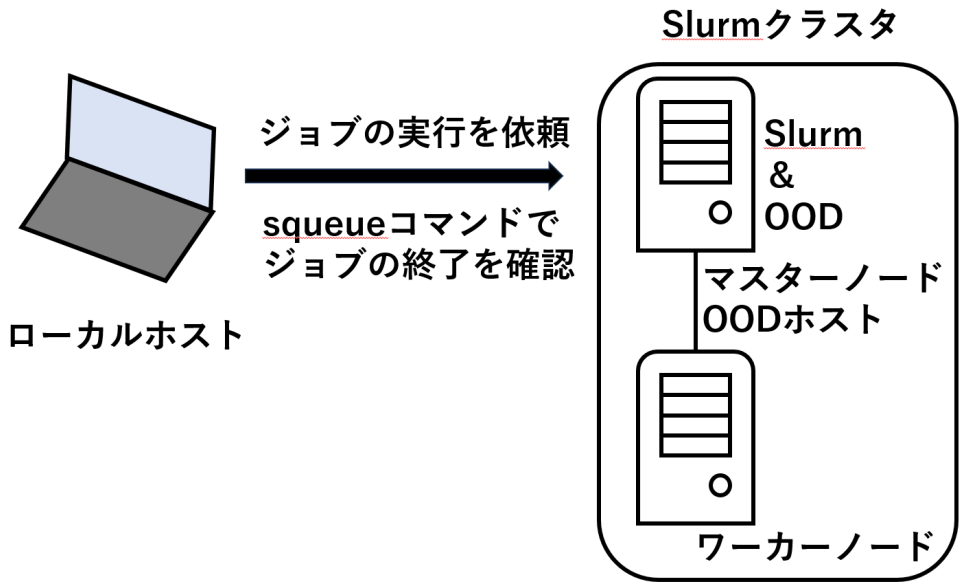

図 12: 評価環境

コマンドにより確認し，コマンドの出力結果によってジョブの実行完了を確認する．New Job ボタンを押下した時刻から，そのジョブの実行を完了した時刻までを計測し，ジョブのターンアラウンドタイムとする．OOD は過去に投入したジョブの履歴が残るという仕様があるため，ターンアラウンドタイムの計測終了後に Job Composer 画面の delete ボタンからジョブの履歴を削除する．また，本研究では，機能分離後のターンアラウンドタイムと機能分離前のターンアラウンドタイムの差分をオーバヘッドと定義する．

4.4　評価条件

評価条件について説明する．はじめに，単一ジョブを機能分離前後で 50 回ずつ投入したときのターンアラウンドタイムを比較する．続いて，ジョブの投入を 1〜10 回連続で行い，そのターンアラウンドタイムを計測して PSI/J を経由する場合と経由しない場合を比較する．ターンアラウンドタイムは 1〜10 回の連続投入でそれぞれ 50 回ずつ計測る．その後，ジョブを 1〜100 回連続投入した際のターンアラウンドタイムの比較を行う．ジョブの連続投入数は 1 回から開始して 10 回ずつ増やしていき，ジョブ数を大きくした場合の結果を得る．

4.5　実行時オーバヘッドの評価

はじめに，単一ジョブを機能分離前後で 50 回ずつ投入したときのターンアラウンドタイムの箱ひげ図を図 13 に示す．左側のデータセットが機能分離前，右側のデータセットが機

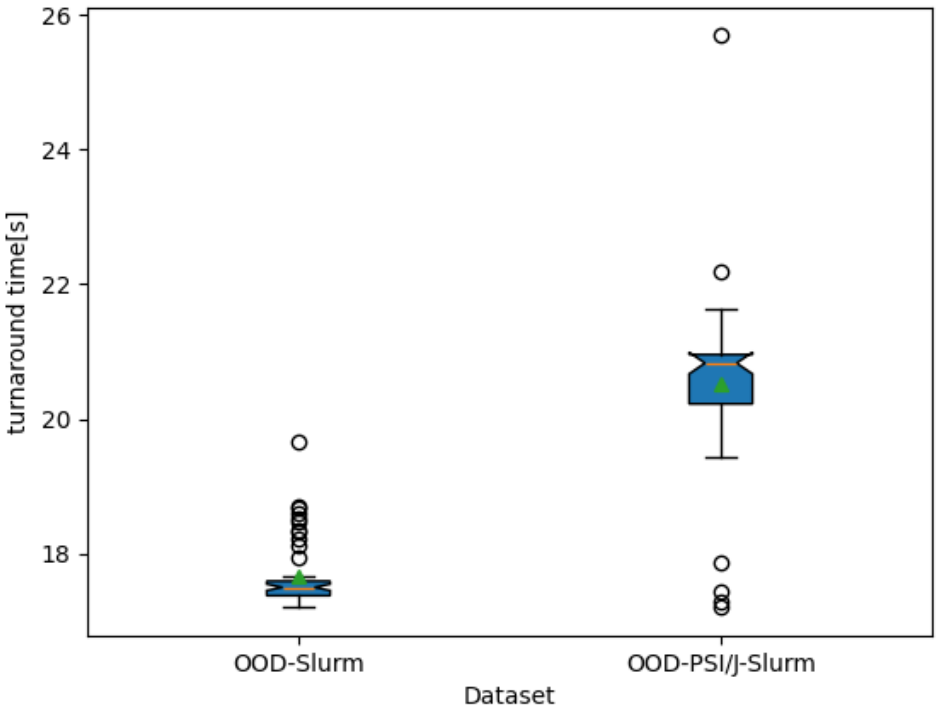

図 13: 機能分離前後での単一ジョブ実行によるターンアラウンドタイム

能分離後を示し，縦軸はターンアラウンドタイムを示す．外れ値を除外すると両データセットには有意差が見られる．機能分離前のターンアラウンドタイムの平均値は 17.67 秒であり，機能分離後のターンアラウンドタイムの平均値は 20.53 秒である．結果から，単一ジョブ実行時のオーバヘッドは 20.53-17.67=2.86 秒と算出される．オーバヘッドは機能分離前のターンアラウンドタイムの 16% 程度であることがわかる．

続いて，1〜10 回のジョブの連続投入により得られた評価結果を図 14 および図 15 に示す．図 14 では機能分離前後でのターンアラウンドタイムをエラーバー付きで比較する．横軸は連続して投入したジョブの数，縦軸はターンアラウンドタイムを示す．図 15 は機能分離によるオーバヘッドを示す．横軸は連続して投入したジョブの数，縦軸はジョブ実行時のオーバヘッドを示す．図 14 から機能分離後の方がターンアラウンドタイムが大きいことがわかり，機能分離前後での結果に有意な差があることがわかる．機能の分離にかかわらず，どちらの場合も連続投入したジョブの数に線形比例して増加していることがわかる．また図 15 から，ジョブの連続投入回数が増加するほど，機能分離によるオーバヘッドも線形増加していることがわかる．したがって，図 14，図 15 から 1 ジョブあたりのオーバヘッドの大きさは変化していないことがわかる．

ジョブを 1〜100 回連続投入した際の機能分離前後のターンアラウンドタイムの比較を図

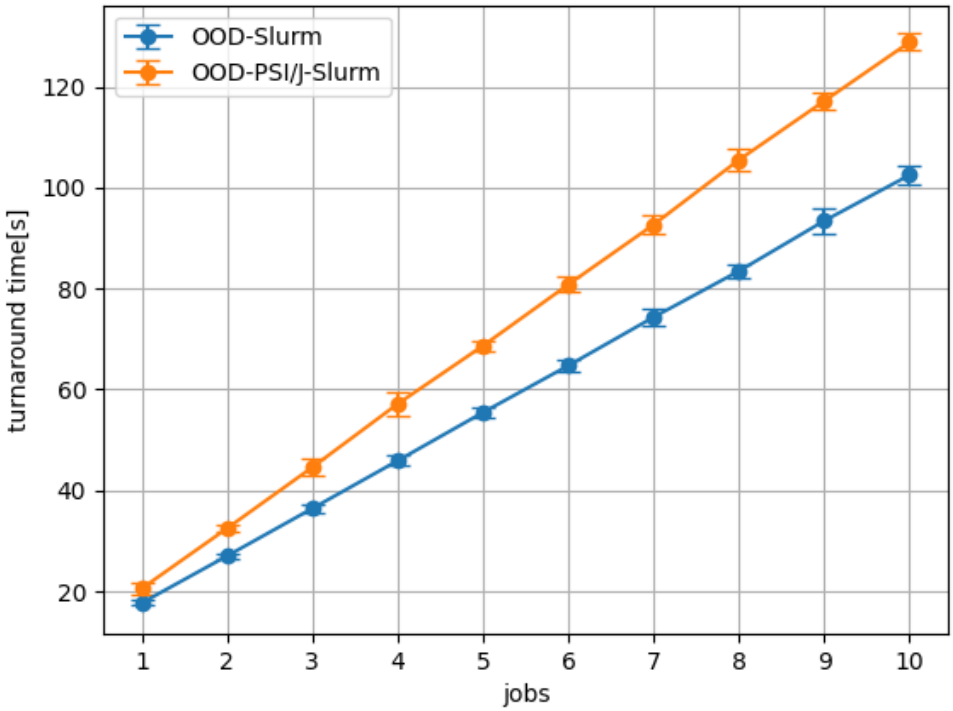

図 14: 機能分離前後でのターンアラウンドタイムの比較

16 に示す．横軸は連続して投入したジョブの数，縦軸はターンアラウンドタイムを示す．1〜10 回の連続投入の場合と同じく，PSI/J を経由した場合の方がわずかにターンアラウンドタイムが大きくなっており，連続投入するジョブ数を大きくしても極端にオーバヘッドに差が出ることはないということがわかった．

様々なジョブの連続投入回数でのターンアラウンドタイムの結果から，ジョブ実行時のオーバヘッドを測定した．その結果から，ジョブの連続投入回数に依らず，PSI/J を経由した場合のオーバヘッドは，PSI/J を経由しない場合のターンアラウンドタイムの 5 %以内に収まり，提案手法によって生じるオーバヘッドは十分無視できるといえる．

4.6 結言

本章では，実際にジョブの投入を行った際の提案手法の実装の評価結果を示した．はじめに，評価を行う環境について説明した．その後，機能の分離前後で生じるオーバヘッドを測定する際の評価条件について説明した．これらの評価環境と評価条件により得られた評価結果により，提案手法により生じるオーバヘッドは充分小さいため実用上は問題ないということを示した．

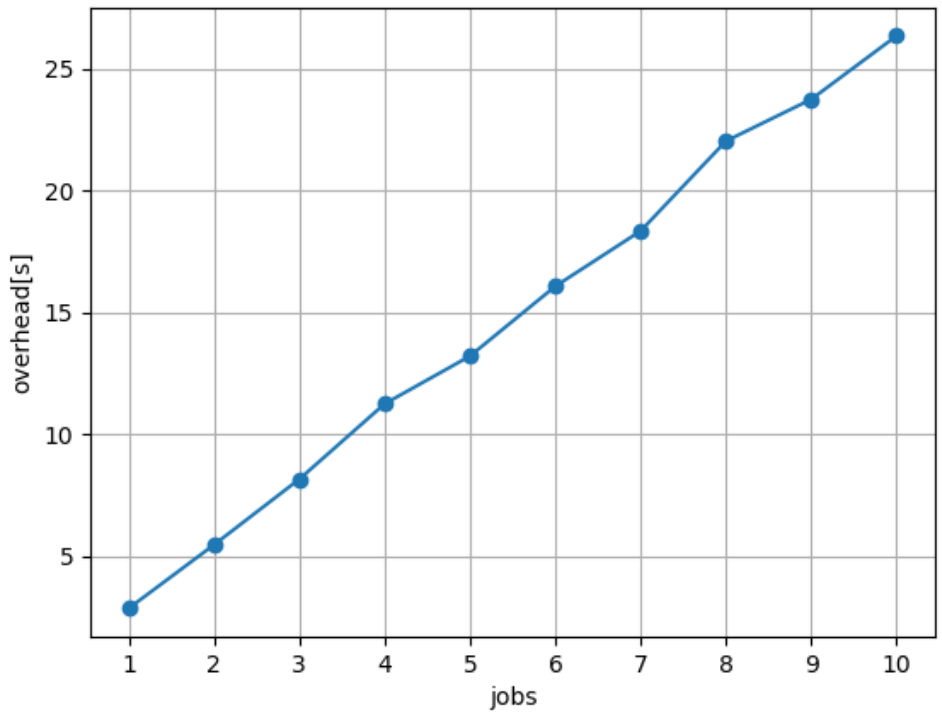

図 15: 機能分離によるオーバヘッド

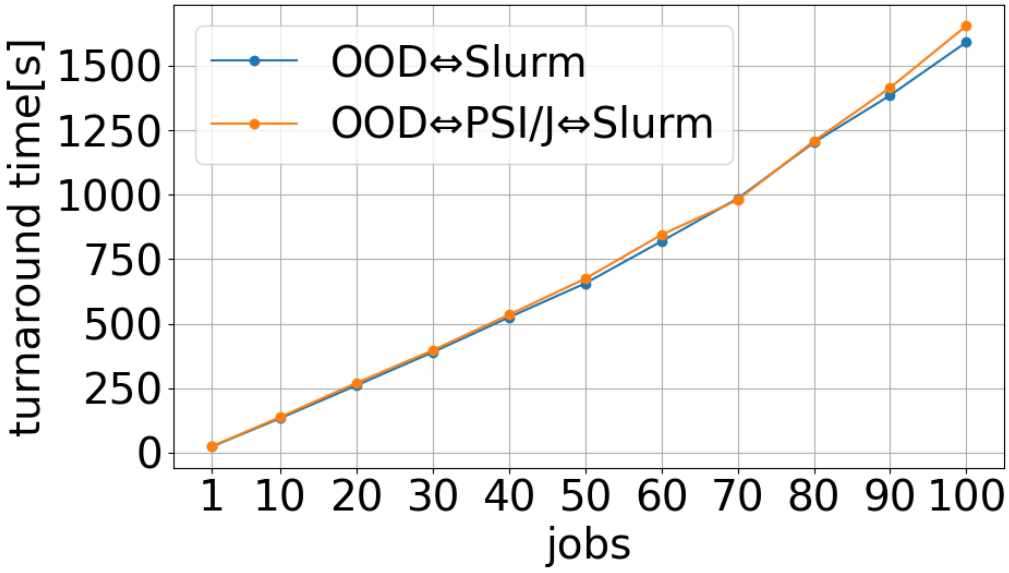

図 16: ジョブ数を増加した際のターンアラウンドタイムの比較

第 5 章　結論

近年，HPC システムの用途は多様化し，専門知識を持たない利用者が HPC システムを利用する需要が高まっている．HPC の利用にはコマンド操作に基づいた利用環境や，システムごとに異なる操作方法など HPC システムを利用するためには多くの学習時間が必要とされている．このような課題点を解決するためにウェブブラウザを用いて容易かつ統一的に HPC システムを利用することが可能なウェブインタフェースについての研究開発が多く行われている．しかし，既存のインタフェースは新たなジョブスケジューラへの対応を行うたびに，ウェブインタフェース本体を改修する必要があるため，保守性に大きな問題を抱えている．

本研究では，多様なジョブスケジューラへの対応に伴い発生し得るウェブインタフェースの保守性の問題に着目し，HPC システムの簡易な利用環境とウェブインタフェースの保守性を併せ持つ実装について考えた．この実現のために，ウェブ機能とスケジューラ抽象化機能を分離し，それぞれ独立に保守管理する手法を提案した．

第 2 章では，本研究の関連研究について説明した．はじめに，HPC システム利用環境について説明した．その後，ウェブインタフェースである OOD について説明し，OOD の機能と設計について説明した．さらに，既存のインタフェースの課題点について説明し，本研究で着目すべき点を明らかにした．

第 3 章では，ウェブインタフェースを介した HPC システム利用環境として提案手法の説明と実装を行った．はじめに，従来手法と提案手法の説明を行った．従来手法と提案手法を比較して，提案手法の利点や設計の変更点について説明した．その後，提案手法の実装を行った．はじめに，実装の概要について説明した．その後，スケジューラ抽象化機能と NQSV の連携を行った．スケジューラ抽象化機能自身がジョブの状態を管理する設計を行ったことにより，スケジューラ抽象化機能の汎用性をより高めることができたといえる．続いて，ウェブ機能とスケジューラ抽象化機能との連携を行った．ウェブ機能がスケジューラ抽象化機能を呼び出すことで，ウェブ機能として OOD を用い，スケジューラ抽象化機能である PSI/J を経由して，指定したスケジューラにジョブの投入や削除を行うことができるようになった．

第 4 章では，機能分離に伴いオーバヘッドの発生が懸念されるため，実装の評価を行った．はじめに，評価環境，評価条件について説明した．その後，ジョブ実行時のオーバヘッドの評価を行った．提案手法で発生が懸念されたオーバヘッドは機能分離前のターンアラウンドタイムの約 5 %以下であることがわかり，提案手法の実装によって生じたオーバヘッドは充分に小さいということが明らかになった．

以上の結果から，HPC システム利用環境におけるウェブインタフェースをウェブ機能とスケジューラ抽象化機能の二つに分離し，それぞれ独立に保守管理することの実現可能性と有用性をしめすことができた．

今後の課題として，PSI/J へのジョブの一旦停止 (hold) 機能と再開 (release) 機能の実装が挙げられる．OOD では hold 機能と release 機能が実装されていることに対して，PSI/J ではそれらの機能に対応していない．そのため，これらの操作に必要な両者の連携を設計して実装していくことで，より多様なジョブスケジューラやその使い方に対応可能となると期待される．

参考文献

[1] Ping Luo, Benjamin Evans, Tyler Trafford, Kaylea Nelson, Thomas J. Langford, Jay Kubeck, and Andrew Sherman. Using Single Sign-On Authentication with Multiple Open OnDemand Accounts: A Solution for HPC Hosted Courses. *IEICE TRANS. INF. SYST*, No. 9, pp. 2307–2314, 9 2018.

[2] Alan Chalker, Eric Franz, Morgan Rodgers, and Trey Dockendorf. Open OnDemand:State of the platform, project, and the future. *Concurrency and Computation: Practice and Experience*, 2021.

[3] 宮崎 博行, 草野 義博新庄 直樹, 庄司 文由, 横川 三津夫, 渡邊 貞. スーパーコンピュータ「京」の概要. *FUJITSU*, Vol. 3, No. 63, pp. 237–246, 2012.

[4] 杉原 英治. スーパーコンピュータ「富岳」の本格運用. 電気学会誌, Vol. 2, No. 141, p. 104, 2021.

[5] David E. Hudak, Thomas Bitterman, Patricia Carey, Douglas Johnson, Eric Franz, Shaun Brady, and Piyush Diwan. OSC OnDemand: A Web Platform Integrating Access to HPC Systems, Web and VNC Applications. *XSEDE '13,*, No. 49, pp. 1–6, 7 2013.

[6] Robert Settlage, Eric Franz, Doug Johnson, Steve Gallo, Edgar Moore, and David Hudak. Open OnDemand: HPC for Everyone. *ISC 2019 Workshops*, Vol. 11887, pp. 504–513, 12 2019.

[7] Bernadette M. Randles, Irene V. Pasquetto, Milena S. Golshan, and Christine L. Borgman. Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study. *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pp. 1–2, 2017.

[8] Documentation for Visual Studio Code. `https://code.visualstudio.com/docs`.

[9] Bill Nitzberg, Jennifer M. Schopf, and James Patton Jones. PBS PRO: GRID COMPUTING AND SCHEDULING ATTRIBUTES. *Grid Resource Management. International Series in Operations Research and Management Science*, Vol. 64, pp. 183–190, 2021.

[10] Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: Simple Linux Utility for Resource Management. *JSSP (Job Scheduling Strategies for Parallel Processing)*, pp. 44–60, 2003.

[11] W. Gentzsch. Sun Grid Engine: towards creating a compute power grid. *Proceedings*

First IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 35–36, 2001.

[12] Dalibor Klusacek, Vaclav Chlumsky, and Hana Rudova. Planning and Optimization in TORQUE Resource Manager. *HPDC '15: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 203–206, 6 2015.

[13] Xiaohui Wei, Wilfred W. Li, Osamu Tatebe, Gaochao Xu, Liang Hu, and Jiubin Ju. Integrating Local Job Scheduler - LSF with Gfarm. *ISPA 2005: Parallel and Distributed Processing and Applications*, Vol. 3758, pp. 196–04, 2005.

[14] dexidp/dex: dex library. `https://github.com/dexidp/dex`.

[15] Sven Hammann, Ralf Sasse, and David Basin. Privacy-Preserving OpenID Connect. *ASIA CCS '20*, pp. 277–289, 10 2020.

[16] Francis Jayakanth, AnandaT. Byrappa, and Raja Visvanathan. Off-campus Access to Licensed Online Resources through Shibboleth. *Information Technology and Libraries*, Vol. 40, , 6 2021.

[17] Marike Kondoj, Tineke Saroinsong, and Anritsu Polii. Single Sign On (SSO) System with Application of Central Authentication Service (CAS) at Manado State Polytechnic. *International Conference on Applied Science and Technology on Engineering Science*, pp. 698–702, 2022.

[18] Masahiro Nakao, Masaru Nagaku, Shinichi Miura, Hidetomo Kaneyama, Ikki Fujiwara, Keiji Yamamoto, and Atsuko Takefusa. Introducing Open OnDemand to Supercomputer Fugaku. *SC-W 2023*, pp. 720–727, 12 2023.

[19] Mihael Hategan-Marandiuc, Andre Merzky, Nicholson Collier, Ketan Maheshwari, Jonathan Ozik, Matteo Turilli, Andreas Wilke, Justin M. Wozniak, Kyle Chard, Ian Foster, Rafael Ferreira da Silva, Shantenu Jha, and Daniel Laney. PSI/J: A Portable Interface for Submitting, Monitoring, and Managing Jobs. *IEEE 19th International Conference on e-Science*, 2023.

[20] 高橋 慧智, 藤本 壮也長瀬 悟, 磯部 洋子, 下村 陽一, 江川 隆輔, 滝沢寛之. ベクトル型スーパーコンピュータ「AOBA-S」の性能評価. 研究報告ハイパフォーマンスコンピューティング（HPC）, Vol. 1, No. 191, pp. 1–9, 9 2023.

[21] NQSV スケジューラー. `https://www.nec.com/en/global/solutions/hpc/articles/tech08.html`.

[22] NEC Network Queuing System V (NQSV) 利用の手引き 導入編 . `https://sxauroratsubasa.sakura.ne.jp/documents/nqsv/pdfs/`

g2ad01-NQSVUG-Introduction.pdf.

[23] OpenOnDemand 3.0.3 Docuentation. https://osc.github.io/ood-documentation/latest/.

[24] install Open OnDemand — Open OnDemand. https://openondemand.org/install-open-ondemand.

[25] Dong H. Ahn, Jim Garlick ans Mark Grondona, Don Lipari, Becky Springmeyer, and Martin Schulz. Flux: A Next-Generation Resource Management Framework for Large HPC Centers. *2014 43rd International Conference on Parallel Processing Workshops*, 2014.

[26] Narayan Desai. FlCobalt: an open source platform for hpc system software research. *Edinburgh BG/L System Software Workshop*, 2005.

[27] NEC Network Queuing System V (NQSV) 利用の手引き リファレンス編. https://sxauroratsubasa.sakura.ne.jp/documents/nqsv/pdfs/g2ad04-NQSVUG-Reference.pdf.

[28] OSC/ood-core: Open OnDemand core library. https://github.com/OSC/ood_core.

[29] ExaWorks/psij-python: psij-python library. https://github.com/ExaWorks/psij-python.

[30] S. Nyamathulla, Dr. P. Ratnababu, Nazma Sultana Shaik, and Bhagya Lakshmi. N. A Review on Selenium Web Driver with Python. *Annals of the Romanian Society for Cell Biology*, Vol. 25, pp. 16760–16768, 6 2021.

謝辞

本研究を進めるにあたり，滝沢寛之教授には，研究方針から論文の執筆まで様々な点でご支援いただきました．心より感謝申し上げます．また，下村陽一特任准教授と高橋慧智助教授にも，研究活動に関する様々な助言を頂きました．心より感謝申し上げます．また，本研究室所属の先輩方に於かれましても，日頃のゼミ等で研究に対する助言を頂きました．心より感謝申し上げます．最後に日頃から支えて頂いていた家族や友人に感謝申し上げます．

令和 6 年 3 月 4 日谷澤悠太