令和 5 年度　　卒業研究前刷　　令和 6 年 2 月 5 日

担当学生　谷澤　悠太
指導教員　滝沢　寛之 教授

題　目:　ウェブインタフェースを介したスーパーコンピュータ利用環境に関する研究

1. 緒論

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，そのようなユーザは，コマンド操作に基づいた利用環境や，利用する HPC システムごとに異なる操作方法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．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ための研究開発が行われている．

2. 課題

代表的な既存研究として，Open OnDemand (OOD) が挙げられる [1]．OOD は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 OOD ポータルサイトの URL と OOD にログインするためのユーザ名・パスワードを用いることで，コマンド操作を介さずにジョブの管理を行うことが可能である．また，世界的に使われている主要なジョブスケジューラ (Tourqe，Slurm，PBS Pro，LSF など) に対応することでシステム間の利用方法の差異も隠蔽している．ただし，OOD が対応していないジョブスケジューラで運用されている HPC システムの場合，OOD を利用するためには OOD 自体を改修する必要がある．例えば，富岳で使われているジョブスケジューラ (Fujitsu Technical Computing Suite, TCS) に OOD が対応していなかったことから，中尾らは OOD を TCS 向けに改修した事例を報告している [2]．その結果，TCS 対応機能が OOD 本体に組み込まれることになったが，他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 本体を直接修正していく方法には保守性に問題がある．

3. 目的

本研究の目的は，HPC 利用者にウェブインターフェスを提供する機能 (ウェブ機能) と，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) を切り分け，それぞれ独立に保守できる構成を実現することである．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 1 に示す．フロントエンドでは，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．バックエンドでは，ウェブ機能から得られた様々なスケジューラに対するリクエスト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，適切な連携方法を検討する．

4. 評価

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ーフェ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，スケジューラ抽象化機能の基盤として PSI/J[3] と呼ばれる Python ライブラリを利用し，両者を組み合わせることで提案手法を実装する．

本研究では，東北大学のスーパーコンピュータ「AOBA」で運用されているジョブスケジューラ (NEC Network Queuing System V, NQSV) 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

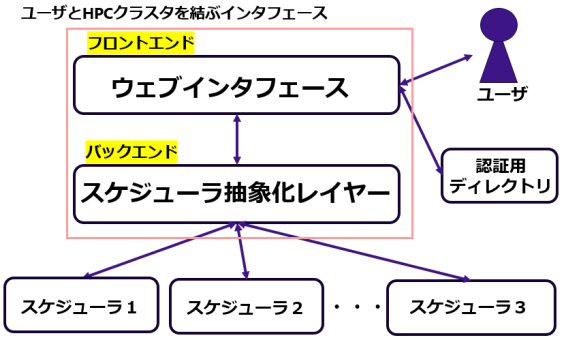

図 1. 提案手法

する．また，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けることで両者間の連携には実行時オーバヘッドが生じることが潜在的に懸念されるため，本実装におけるオーバヘッドを定量的に評価する．

4.1. 機能の実装

実行環境として，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 クラスタを考える．HPC クラスタでは，AOBA と同様に NQSV がジョブスケジューラとして利用されている．OOD はログイン時に必要な認証用のディレクトリである Active Directiory(AD) と連携する．

本研究では、はじめにスケジューラ抽象化機能の NQSV 対応を考える．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，ステータス確認を行うための 3 つのメソッドを実装する．PSI/J が対応している他のスケジューラ (Slumr, PBS Pro, LSF, Flux, Cobalt) はジョブの終了後にジョブのステータス (COMPLETED, CANCELED，FAILED) を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ジョブのステータス (COMPLETED，CANCELED，FAILED) を確認できない．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．この機能を実現するため，本研究の実装ではジョブが投入された後にジョブキューからジョブが無くなった際にそのジョブの状態を COMPLETE に変更する．また，ジョブが削除された際には，そのジョブの状態を CANCELED に変更する．それ以外にジョブキューの状態を定期的に確認し，ジョブが存在する場合には QUEUED あるいは ACTIVE という状態にする．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．

続いて，ウェブ機能側である OOD 側からスケジューラ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．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，単純な実装であるため，PSI/J を用いたジョブの管理のための python スクリプトをシェルを経由して Ruby スクリプト上で直接実行する．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，指定したスケジューラにジョブの投入や削除を行うことがで

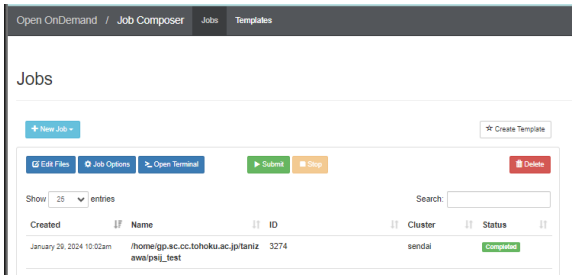

図 2. Job Composer の画面

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．

図 2 は OOD 上でジョブを作成して投入と削除を行う「Job Composer」の画面である．ジョブを作成する際にクラスタを「psij」に設定することで，PSI/J を経由して Slurm クラスタや NQSV クラスタなど任意の HPC システムにジョブを投入できていることが確認できた．

4.2. 実行時オーバヘッド

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け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 と呼ばれるウェブページの自動制御ライブラリを用いる．ローカルホストから別ホストの OOD のポータルサイトにアクセスして，Job Composer 画面でジョブの作成と投入を行う．ジョブの作成を開始した時刻から，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．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した結果を図 3 および図 4 に示す．

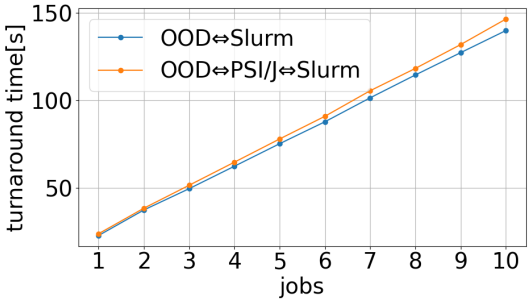

図 3. 実行時オーバーヘッドの比較

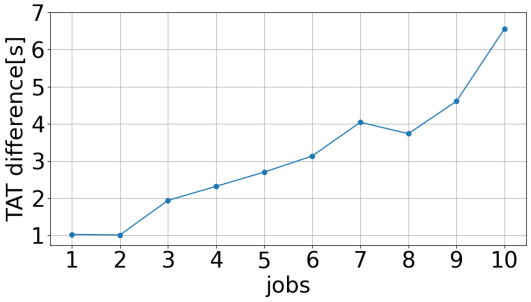

図 4. 実行時オーバーヘッドの差

ターンアラウンドタイムはそれぞれ合計 20 回計測され，その平均値が図 3 および図 4 には示されている．横軸は連続して投入したジョブの数を示す．図 3 の縦軸はジョブのターンアラウンドタイムを示し，図 4 の縦軸はジョブのターンアラウンドタイムの差を示す．図 3 から PSI/J を経由した提案手法の方がわずかにターンアラウンドタイムが大きいことがわかり，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 4 から，ジョブの連続投入回数が多くなればなるほど両者の実行時オーバヘッドの差が大きくなっていることがわかる．

次に，ジョブを 1〜100 回連続投入した際の実行時オーバーヘッドを比較する．図 5 は，1〜100 回までのジョブの連続投入数を 10 回ずつ増やしたときのターンアラウンドタイムの比較を示している．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．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．

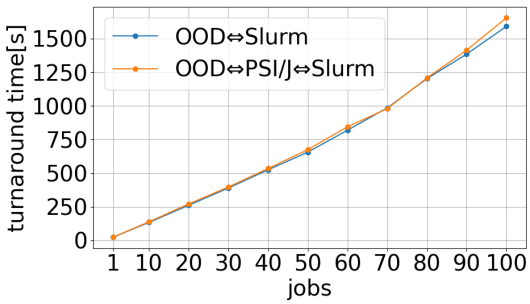

図 5. ジョブ数を増加した際のオーバヘッドの比較

5. 結論

本研究では，HPC システムの利用難易度の高さやスケジューラ細分化に伴うシステムの保守性の問題という背景のもと，HPC 利用者の支援という目的で，HPC システムの簡易な利用環境と充分なシステムの保守性を合わせ持つウェブインタフェースを実装してきた．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．

提案手法では，OOD をインタフェースとして PSI/J を介してジョブの投入と削除を実現している．実行時オーバヘッドを測定して比較することで，OOD と PSI/J の連携のために生じる実行時オーバヘッドは十分に小さいことが明らかになった．また，PSI/J 側で NQSV に対応することによって OOD から NQSV を利用可能となり，ウェブ機能とジョブスケジューラ抽象化機能の分離の実現可能性と有用性を示すことができた．

現在，OOD がジョブの一旦停止 (hold) と再開 (release) の操作も提供しているのに対して，PSI/J ではそれらの操作に対応していない．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．その検討は今後の課題である．

参考文献

[1] 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.

[2] Masahiro Nakao, Masaru Nagaku, Shinichi Miura, Hidetomo Kaneyama, Ikki Fujiwara, Keiji Yamamoto, and Atsuko Takefusa. Introducing Open OnDemand to Supercomputer Fugaku. *SC-W 2023*, No. 1, pp. 720–727, 11 2023.

[3] 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.