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. 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．ただし，このようなユーザは，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][2]．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 のポータルサイトで URL，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となった．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）をサポート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 がサポート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）が OOD でサポートされ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 [3]．その結果，TCS サポートが 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は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．

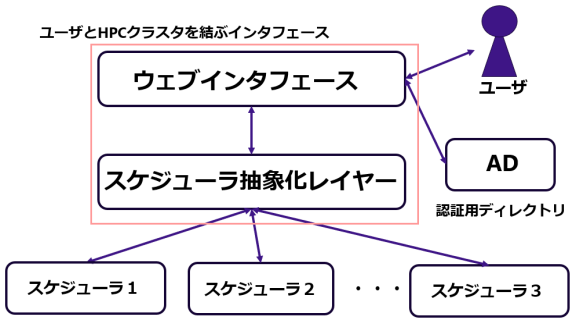

図 1. 提案手法

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[4] 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した．

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 でサポートされ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．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

実装環境として，OOD 用のホストサーバと「AOBA」を想定した NQSV が使用されている HPC クラスタを用いて実装を行った．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）と連携して使用した．

始めに NQSV をスケジューラ抽象化機能側に実装した．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オーバーライドして定義している JobExecutor クラスにより構成されている．今回は新たに 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した．実装において，PSI/J がサポート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が，NQSV ではジョブの終了後にジョブのステータス（COMPLETED, CANCELED, FAILED）を確認できないという仕様上の問題があった．そのため，NQSV をサポートするにあたり，「ジョブ投入メソッド実行の有無」「ジョブ削除メソッドの実行の有無」「キュー内部のジョブの存在の有無」でジョブのステータスを判断するように修正した．ジョブは「ジョブ投入メソッドが実行され，キューにジョブが存在しない場合」に COMPLETED ステータスとなる．また，「ジョブ削除メソッドの実行」により CANCELED ステータスとなる．「ジョブ投入メソッドが実行され，キューにジョブが存在する場合」は QUEUED ステータスまたは ACTIVE ステータスとなる．今回適用した改修は，スケジューラが提供するジョブ管理のためのコマンドが，PSI/J の想定している結果を返さない場合にも用いることができる．そのため，各ジョブスケジューラのコマンド出力に依らず，より汎用性の高い実装を行う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ができるように実装する．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手法を用いた．この実装により，ウェブ機能として OOD を用いると，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．また，PSI/J を仲介することで，OOD 未サポートであった NQSV でのジョブ管理を OOD 上から操作することができるようになった．

図 2 は OOD 上でジョブを作成して投入・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，psij を経由して Slurm クラスタや NQSV クラスタなど任意の HPC システムにジョブの投入を行うことが確認できた．

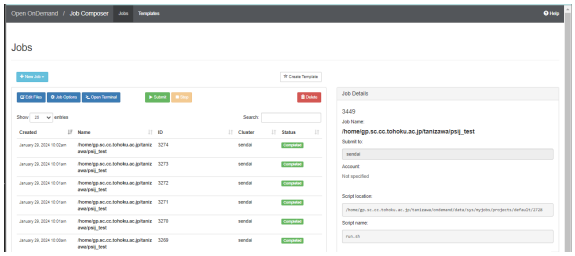

図 2. Job Composer の画面

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る両者の連携時のオーバヘッドを測定する．実装において，シェルを介して python スクリプトを実行するため，その際のオーバヘッドの影響を評価した．測定は，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て実行する．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・投入を行う．ジョブの作成を開始した時刻から，投入したジョブが完了した時刻を計測して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 を経由する時と経由しない時を比較した．以下の図 3，図 4 にその結果を示す．

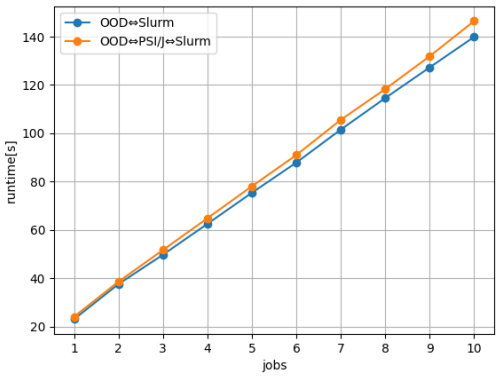

図 3. 実行時オーバーヘッドの比較

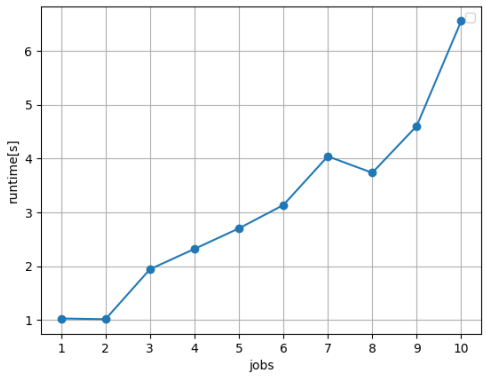

図 4. 実行時オーバーヘッドの差

計測は全部で 20 回行い，その平均値を使用して評価を行った．横軸は連続して投入したジョブの数，縦軸はジョブのターンアラウンドタイムを示している．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．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，ジョブの投入を 0〜100 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 0〜100 回まで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た．このとき，グラフ中のジョブ数 0 の場合は，ウェブ画面上でのページ遷移の時間が含まれているためターンアラウンドタイムが 5 秒程度となっている．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く同様の傾向が見られることがわかった．

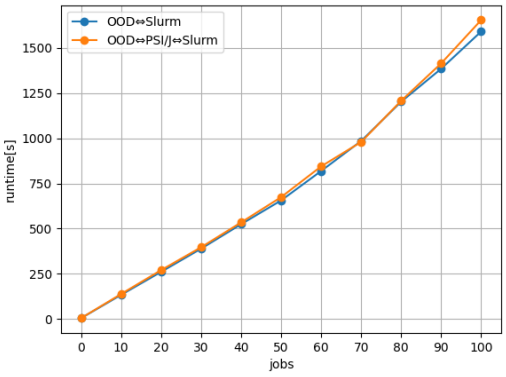

図 5. ジョブ数を増加した際のオーバヘッドの比較

5. 考察

検証の結果，OOD のインタフェースから NQSV を利用してジョブの投入や削除が行われることが確認された．ただし、ジョブのホールドとリリースに関する機能は、OOD では実行されているが PSI/J では未実装であるという課題もある．また，ウェブ機能とスケジューラ抽象化機能に分離する前と後で指定数のジョブを連続投入してスループットを比較した．ジョブ数が 1〜10 の範囲内であれば，両者の差が最大でも 5% を超えないことから，提案手法によって生じるオーバヘッドは十分に小さいことが明らかになった．

6. 結論

ウェブインタフェースを介して HPC システムを利用する環境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により，それぞれの機能を独立に開発していくことが可能となった．それによってソフトウェアの保守性を高めることが可能となり，長期にわたって多様なシステムをサポートしていく労力の削減効果が期待できる．また，機能分離に伴う実行時オーバヘッドは無視できる程度であり，運用上の問題にはならないことが明らかになった．今後の課題として，PSI/J にジョブ管理の他のメソッドを追加することで OOD で実行できる機能を増やすこと，OOD ではサポートされていない他のジョブスケジューラを PSI/J を経由して実装することなどが挙げられる．

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] Robert Settlage, Eric Franz, Doug Johnson, Steve Gallo, Edgar Moore, and David Hudak. Open OnDemand: HPC for Everyone. *ISC 2019 Workshops*, No. 11887, pp. 504–513, 12 2019.

[3] 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.

[4] 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 Submit-

ting, Monitoring, and Managing Jobs. *IEEE 19th International Conference on e-Science*, 2023.