# スーパーコンピュータ「富岳」における HPCクラスタ用Webポータル Open OnDemandの導入

中尾 昌広[1,a)]　三浦 信一[1]　山本 啓二[1]

**概要**：「富岳」などの HPC クラスタの問題点として，HPC クラスタを用いるための前提知識が多いため，初心者にとって利用するまでの学習コストが大きい点が挙げられる．また，近年では，対話的操作を伴う GUI（Graphical User Interface）アプリケーションを計算ノード上で動作させることを望まれているが，その手順は煩雑である．そこで，本稿では HPC クラスタの計算資源を簡易に利用可能にする Web ポータル Open OnDemand を「富岳」に導入する．その導入を実現するため，「富岳」で用いられているジョブスケジューラを Open OnDemand から利用できるアダプタの開発を行った．本稿では，アダプタの開発および「富岳」における Open OnDemand の利用例について述べる．

## 1. はじめに

理化学研究所 計算科学研究センター（R-CCS：RIKEN Center for Computational Science）[1] は，日本におけるフラグシップスーパーコンピュータとして「富岳」を運用している [2]．また，R-CCS は「富岳」の利便性を向上させるため，可視化やデータ変換等を行うプリポスト環境も提供している．**図 1** に「富岳」とプリポスト環境の概念図を示す．プリポスト環境は，GPU を搭載したノードと大容量メモリを搭載したノードで構成される．ジョブスケジューラは「富岳」とプリポスト環境とで異なり，「富岳」は Fujitsu Software Technical Computing Suite（Fujitsu TCS）[3] であるのに対し，プリポスト環境は Slurm[4] である．各システムの利用手順としては，ユーザはまず Secure Shell（SSH）を用いて共通のログインノードにログインし，次にジョブスケジューラを用いて各システムにジョブを投入する．

「富岳」などの HPC クラスタを利用するためには，Shell による CLI（Command Line Interface），SSH の鍵ペアの生成と公開鍵の登録，ジョブスケジューラなどの知識が必要であるため，初心者にとって学習コストが大きいという問題点がある．また，近年では，対話的操作を伴う GUI（Graphical User Interface）アプリケーションを HPC アプリケーションとして動作させることを望まれているが，そのようなアプリケーションを HPC クラスタ上で動作させる手順は煩雑である．例えば，JupyterLab のような Web ベースのアプリケーションの場合，次のような手順をアプリケーションを実行する毎に CUI で行う必要がある．(1) SSH でログインノードにログインする．(2) ジョブスケジューラを通して JupyterLab を計算ノード上で実行する．(3) 計算ノードの IP アドレスと JupyterLab が利用するポート番号を取得する．(4) 取得した IP アドレスとポート番号に SSH トンネリングでローカルポートと接続する．(5) ローカルポートを Web ブラウザで開く．このように，多くの手間を要するだけでなく，これらの作業には前述した知識も必要であるため，GUI に慣れたユーザにとって大きな負担となっている．

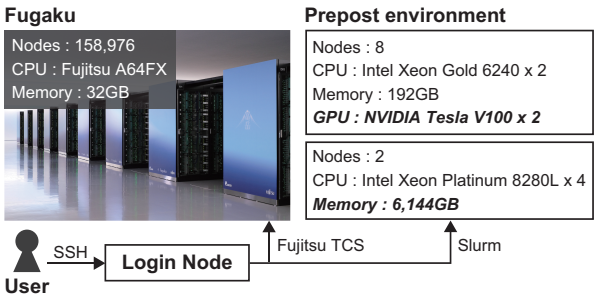



**図 1** 「富岳」とプリポスト環境の概念図

本稿では HPC クラスタ用の Web ポータル Open OnDemand[5] を「富岳」に導入する．Open OnDemand を用いると，SSH ではなく Web ブラウザから HPC クラスタの計算資源を利用できる．さらに，HPC クラスタの計算ノード上で動作する GUI アプリケーションの対話的操作を簡易に実行できる．ここで，Open OnDemand は様々なスケ

[1] 理化学研究所 計算科学研究センター 兵庫県神戸市中央区港島南町 7-1-26
[a)] masahiro.nakao@riken.jp


**図 2** Open OnDemand のダッシュボード

ジューラ（Slurm，PBS Pro，Torque など）に対応しているが，Fujitsu TCS には未対応であった．そのため，本稿では Open OnDemand において Fujitsu TCS を利用するためのアダプタの開発も行う．

本稿の構成は次の通りである．2 章では，Open OnDemand の概要について述べる．3 章では，Fujitsu TCS に対するアダプタの開発について述べる．4 章では，「富岳」における Open OnDemand の設定と利用例について述べる．5 章では，本稿のまとめと今後の課題について述べる．

## 2. Open OnDemand

### 2.1 概要

Open OnDemand は HPC クラスタの計算資源を簡易に利用可能にするためのオープンソースソフトウェアであり，Ohio Supercomputing Center[6] の Web ポータルが元となっている [7]．Open OnDemand の目的の 1 つは，HPC クラスタの利用のための学習コストを小さくすることである．Settlage らは，ユーザの最初のログインからジョブを投入するまでの時間の中央値が従来の SSH を用いた方法では約 22 時間であるのに対し，Open OnDemand を用いた方法では約 2 時間であることを示している [8]．

**図 2** に「富岳」に導入した Open OnDemand のダッシュボードを示す．Interactive Apps にあるアイコンは，計算ノード上で動作する対話的操作を伴う GUI アプリケーションである（リモートデスクトップ，JupyterLab，RStudio，VSCode）．Passenger Apps にあるアイコンは，Open OnDemand がインストールされたサーバで動作するアプリケーションである．Home Directory はファイルのアップロード・ダウンロード・編集を行う（**図 3**）．Active Jobs はジョブの監視を行う（**図 4**）．Job Composer はジョブの作成・投入を行う（**図 5**）．Shell はターミナルによる CUI

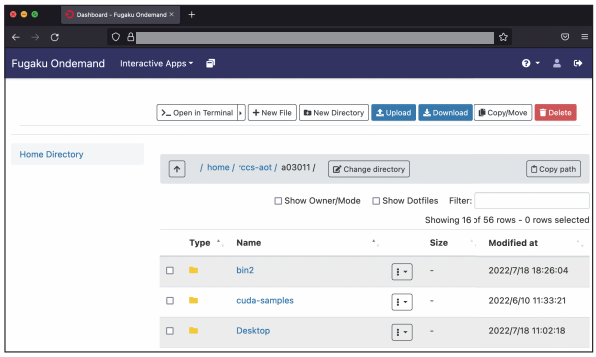



**図 3** Home Directory

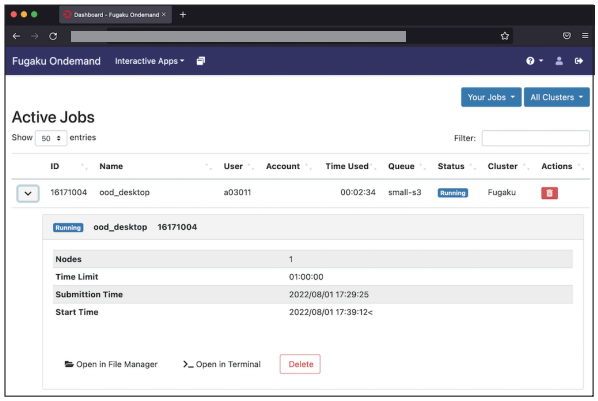



**図 4** Active Jobs

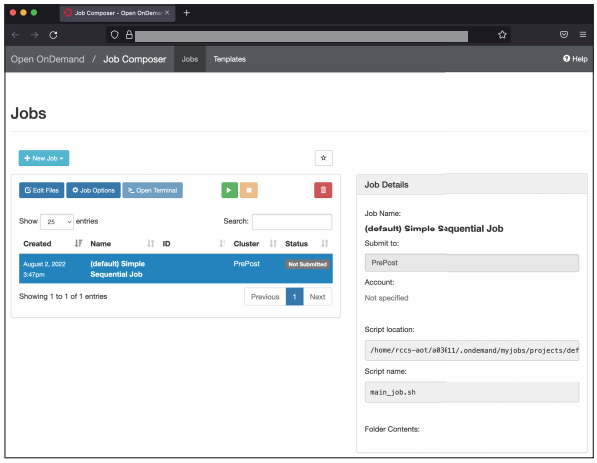



**図 5** Job Composer

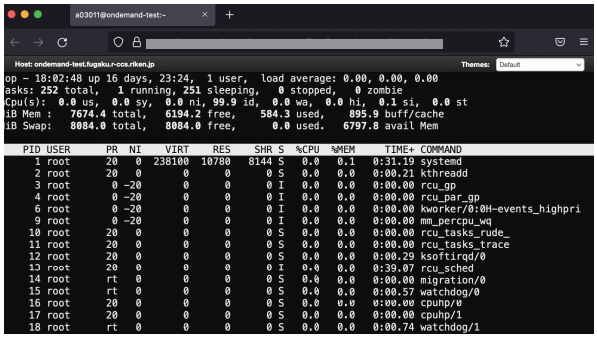



**図 6** Shell

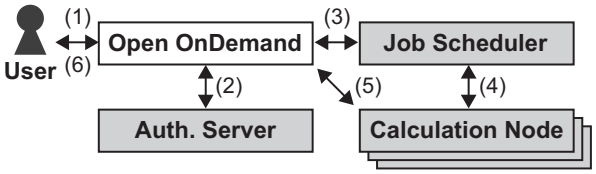



**図 7** Interactive Apps のアプリケーションの動作フロー

を提供する（**図 6**）．これらのアプリケーションは管理者が自由に追加・削除可能である．また，Open OnDemand が提供するフレームワークを用いることで，新しいアプリケーションの開発・導入も簡易に行うことができる．

### 2.2 動作フロー

**図 7** と下記に Interactive Apps のアプリケーションの動作フローを示す．図において灰色の矩形で示した認証サーバ，ジョブスケジューラ，計算ノードは HPC システムが提供するサービスであり，Open OnDemand とは独立している．(1) ユーザは Web ブラウザを用いて Open OnDemand が動作している Web サーバにログインする．なお，Web ブラウザ以外のソフトウェアや Web ブラウザ用のプラグインなどは必要ない．(2) Open OnDemand はログインのための認証を行う．(3) Web ブラウザからアプリケーションの実行命令が発行されると，ジョブスケジューラは計算ノードにジョブを投入する．(4) ジョブスケジューラは計算ノードでジョブが実行されるまで待機する．(5) ジョブが実行されると，確保された計算ノードの情報や GUI アプリケーションが利用しているポート番号などが Open OnDemand に送られる．Open OnDemand は，その情報から確保された計算ノードに接続するためのリバースプロキシの設定を行う．(6) ユーザはリバースプロキシ用の URL を用いて，Web ブラウザから HPC システム内部にある計算ノードに接続する．

### 2.3 関連研究

中田らは Jupyter Notebook を HPC クラスタの計算ノード上で動作させる方法を 2 つ提案している [9]．1 つ目は，1 章で述べた SSH トンネリングの手順を自動化するものである．その問題点として，ユーザの計算機環境にスクリプトをインストールし，さらに手動でそのスクリプトを実行する点が挙げられる．2 つ目は，JupyterHub を HPC クラスタとは別のクラウドサービス（AWS などのパブリッククラウド）に設置する方法である．この方法は HPC クラスタの 1 つのアカウントを複数のユーザで共有することを前提としているため，「富岳」の運用ポリシでは許可できない．

野村らは JupyterLab を HPC クラスタの計算ノード上で動作させるための Web アプリケーション実行基盤を開発した [10]．その動作フローは図 7 の Open OnDemand の動作フローとほぼ同じである．しかしながら，その Web アプリケーション実行基盤はオープンソースソフトウェアではなく，また利用可能な認証サーバやジョブスケジューラの種類，JupyterLab 以外のアプリケーションへの適用については制限がある．

## 3. Fujitsu TCS への対応

### 3.1 概要

本章では，Open OnDemand において Fujitsu TCS を利用するためのアダプタを実装する．さらに，Fujitsu TCS に関連するジョブの詳細情報の表示およびジョブ投入のオプションについての既存のコードの変更も行う．本章で行った変更は，Open OnDemand の GitHub リポジトリの master ブランチにマージされている[*1]．そのため，Fujitsu TCS をジョブスケジューラに採用している他の HPC クラスタ（東京大学の「Wisteria/BDEC-01」，名古屋大学の「不老」，九州大学の「ITO」など）では，そのまま Open OnDemand を利用することができる．

### 3.2 ジョブスケジューラのアダプタ

Open OnDemand は，様々なジョブスケジューラに対応するためのアダプタインタフェースを提供している．そこで，アダプタの親クラスで定義されている下記のメソッドを Fujitsu TCS 用に実装する．なお，実装言語は Ruby である．

**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**　ジョブのホールド
**release**　ホールドされたジョブの開放
**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 クラスタのシステム情報を取得
**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を取得（Fujitsu TCS の場合は#PJM）

本節では，**delete** メソッドと **info_all** メソッドの実装についてコードを用いて説明する．ただし，説明の簡易化のため，例外処理の箇所はコードから削除している．

**delete** メソッドの実装を**図 8** に示す．8 行目でオーバーライドされた **delete** メソッドが定義されている．9 行目の**@fujitsu_tcs** は **Batch** クラスのインスタンスである．5 行目でコールしている `pjdel` コマンドはジョブを削除する Fujitsu TCS のコマンドであり，その引数としてジョブ ID が渡されている．

[*1] `https://github.com/OSC/ood_core/pull/766` と `https://github.com/OSC/ondemand/pull/2194`

```
1  module Adapters
2   class Fujitsu_TCS < Adapter
3    class Batch
4     def delete_job(id)
5      call("pjdel", id.to_s)
6     end
7
8     def delete(id)
9      @fujitsu_tcs.delete_job(id.to_s)
10    end
```

**図 8　delete** メソッドのコード

```
1  def get_jobs(id: "", owner: nil)
2   args = ["-s", "--data", "--choose=jid,jnam,rscg,st,std,
        stde,adt,sdt,nnumr,usr,elpl,elp"]
3   args.concat ["--filter␣jid=" + id.to_s] unless id.to_s.empty?
4   args.concat ["--filter␣usr=" + owner.to_s] unless owner.to_s
        .empty?
5
6   StringIO.open(call("pjstat", *args)) do |output|
7    output.gets() # Skip header
8    jobs = []
9    output.each_line do |line|
10    l = line.split(",")
11    jobs << {:JOB_ID => l[1], :JOB_NAME => l[2],
12       :RSC_GRP => l[3].split("␣")[0], :ST => l[4],
13       :STD => l[5], :STDE => l[6], :ACCEPT => l[7],
14       :START => l[8], :NODES => l[9].split(":")[0],
15       :USER => l[10], :ELAPSE_LIM => l[11],
16       :ELAPSE_TIM => l[12].split("␣")[0]}
17   end
18  jobs
19  end
20 end
21
22 def parse_job_info(v)
23  Info.new(id: v[:JOB_ID], job_name: v[:JOB_NAME],
24   queue_name: v[:RSC_GRP], status: get_state(v[:ST]),
25   submission_time: v[:ACCEPT], dispatch_time: v[:START],
26   wallclock_limit: duration_in_seconds(v[:ELAPSE_LIM]),
27   wallclock_time: duration_in_seconds(v[:ELAPSE_TIM]),
28   job_owner: v[:USER], native: v
29  )
30 end
31
32 def info_all(attrs: nil)
33  @fujitsu_tcs.get_jobs().map do |v|
34   parse_job_info(v)
35 end
```

**図 9　info_all** メソッドのコード

**info_all** メソッドの実装を**図 9** に示す．32 行目でオーバーライドされた **info_all** メソッドが定義されている．1 行目で定義された **get_jobs** メソッドにおいて，6 行目でコールされている `pjstat` コマンドはジョブ情報を取得する Fujitsu TCS のコマンドであり，そのオプションは 2〜4 行目で定義されている．2 行目の”-s”は詳細情報の出力，”--data”は CSV 形式で出力，”--choose”は出力する項目を選択するオプション（左から，ジョブ ID，ジョブ名，リソースグループ，状態，標準出力ファイルのパス，標準エラーファイルのパス，投入時間，開始時間，ノード数，ユーザ名，制限時間，経過時間）である．3〜4 行目で用いられている”--filter”は，それぞれジョブ ID とユーザ名で `pjstat` コマンドの出力をフィルタリングするオプションである．11〜16 行目で，`pjstat` コマンドの出力から得られたジョブの情報が整形され，ハッシュ配列 **jobs** に格納される．22 行目で定義された **parse_job_info** メソッドにおいて，**get_jobs** メソッドから得られたハッシュ配列 **jobs** を元に **Info** オブジェクトがジョブごとに生成される．24 行目の **get_state** メソッドは，`pjstat` コマンドから得られたジョブの状態を，Open OnDemand で定義されているジョブの状態（undetermined，completed，queued_held，queued，running，suspended のいずれか）に変換する．26〜27 行目の **duration_in_seconds** メソッドは，期間を秒数に変換（例えば，”00:01:23”を 83 に変換）する．

```
1  def extended_data_fujitsu_tcs(info)
2   return unless info.native
3   attributes = []
4   attributes.push Attribute.new "Nodes", info.native[:NODES]
5   attributes.push Attribute.new "Time␣Limit", pretty_time(
        info.wallclock_limit)
6   attributes.push Attribute.new "Submittion␣Time", info.native
        [:ACCEPT]
7   attributes.push Attribute.new "Start␣Time", info.native[:
        START]
8   ...
```

**図 10**　Active Jobs のコード

```
1  def submit(fmt: nil)
2   slots = value.blank? ? 1 : value.to_i
3   case fmt
4   when "slurm"
5    native = ["-N", slots]
6   when "fujitsu_tcs"
7    native = ["-L", "node=#{slots}"]
8    ...
```

**図 11**　ジョブ投入オプションのコード

### 3.3　ジョブの詳細情報

図 4 に示した Active Jobs の画面の ID 番号の左にあるボタンをクリックすると，ジョブの詳細が表示される．しかしながら，Fujitsu TCS のように未サポートのスケジューラの場合は，詳細には何も表示されない．そこで，その詳細に **pjstat** コマンドから得られる情報であるジョブのノード数，制限時間，投入時間，開始時間を表示させることにした．変更を行ったコードを**図 10** に示す．コード中で用いられている変数 **info** は，図 9 で作成された **Info** オブジェクトである．

### 3.4　ジョブ投入オプション

複数の計算ノードを用いる場合，ジョブ投入コマンドに

ノード数を指定するオプションが必要である．例えば，Fujitsu TCS で 2 ノードを用いる場合は，”`pjsub -L node=2 ...`”である．それを実現するために変更したコードを**図 11** に示す．既存のコードに 6〜7 行目を追加したのみである．

## 4. 「富岳」における Open OnDemand

### 4.1 概要

本章では，「富岳」における Open OnDemand の設定と利用例について述べる．4.3 節と 4.4 節で説明する Singularity のレシピおよび Web フォームの設定ファイルは `https://github.com/RIKEN-RCCS/ondemand_fugaku` で公開している．

### 4.2 認証について

「富岳」に SSH でログインするまでの従来の手順は次の通りである．(1) クライアント証明書がユーザに送付される．(2) クライアント証明書をユーザの Web ブラウザにインストールする．(3) Web ブラウザを用いて「富岳」の利用者ポータルにログインする．(4) ローカル PC で SSH の鍵ペアを作成する．(5) SSH の公開鍵を「富岳」の利用者ポータルに登録する．(6) ログインノードに SSH でログインする．

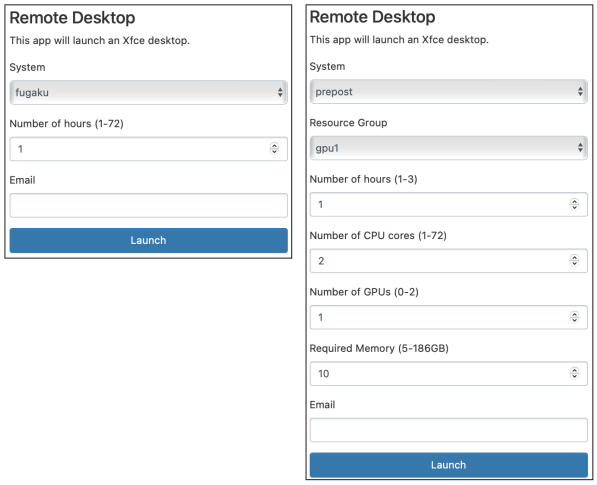
上記の通り，「富岳」ではクライアント証明書を用いたユーザ認証を行っている．そこで，図 7 の Open OnDemand に用いる認証サーバとして，Web 上におけるシングルサインオンを実現するための認証ソフトウェアである Keycloak を導入し，クライアント証明書を使った認証を行った．Keycloak と Open OnDemand を用いる場合は，上記の (2) までの手順を行うのみで，Open OnDemand の Web ポータルにログインすることができる．このことから，アカウント発行から「富岳」を利用するまでの時間を大幅に短縮できると考えられる．

### 4.3 Singularity コンテナの利用

計算ノード上で動作させるアプリケーション（図 2 の Interactive Apps）の環境には，アプリケーションイメージの管理を容易にするために，HPC 向けコンテナ環境である SingularityPro[11] を利用する．ここで，図 1 にある通り，「富岳」の CPU は ARM アーキテクチャに基づく Fujitsu A64FX[12] であるのに対し，プリポスト環境の CPU は一般的な x86_64 アーキテクチャである．そのため，アーキテクチャ毎・アプリケーション毎にコンテナイメージを構築した．各アーキテクチャにおけるレシピファイルは基本的には同じであるが，一部のアプリケーションの ARM アーキテクチャ版の RPM パッケージは存在しない場合がある（例えば，リモートデスクトップで用いる TurboVNC など）．そのような場合は，レシピファイル中で `wget` コマンドを用いて該当アプリケーションのソースコードを取得し，コンパイルとインストールを行っている．

**表 1** 各リソースグループで指定できる項目の最大値

| System | Resource group | Time (hours) | CPU Cores | Memory (GB) | GPUs |
|---|---|---|---|---|---|
| Fugaku | small-s3 | 72 | - | - | - |
| Prepost | gpu1 | 3 | 72 | 186 | 2 |
| | gpu2 | 24 | 36 | 93 | 2 |
| | mem1 | 3 | 224 | 6,045 | - |
| | mem2 | 24 | 56 | 1,511 | - |




(a) Fugaku (b) Prepost

**図 12** Dynamic Form Widgets を用いた Web フォーム

### 4.4 Web フォームの動的変更

「富岳」とプリポスト環境におけるジョブスケジューラの各リソースグループとユーザが指定できる設定項目の最大値を**表 1** に示す．プリポスト環境では oversubscribe（1 台の計算ノードに複数のジョブを同時に実行できること）が可能であるため，ユーザは CPU コア数[*2]，メモリ量，GPU 数の指定が可能である．「富岳」では複数の計算ノードを用いた並列計算も可能であるが，今回の導入では 1 ノードのみの利用を想定しているため，表 1 ではノード数の項目は省略している．

ここで，「富岳」とプリポスト環境という異なるシステムに対して，同じ URL の Web フォームで図 2 の Interactive Apps を利用できると便利である．それを実現するために，Open OnDemand の Dynamic Form Widgets の機能を利用する．この機能を用いると，Web フォームの項目を動的に変更させることができる．

次に，Dynamic Form Widgets の機能を用いて構築した Web フォームによる Open OnDemand の利用手順を説明する．まず，例えば図 2 のリモートデスクトップのアイコンをクリックすると，**図 12**-(a) の「富岳」用の Web フォームが表示される．ここで，システムの項目を「prepost」に

[*2] プリポスト環境の Hyper-Threading は有効になっている

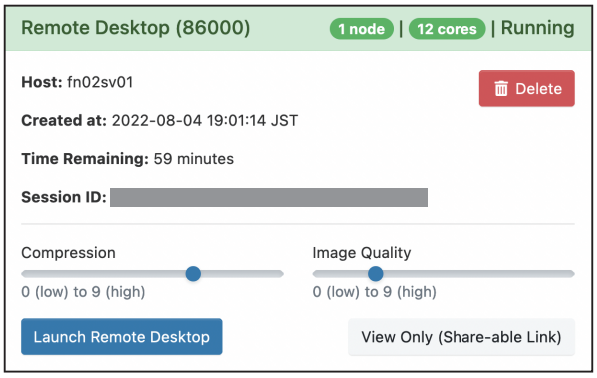



**図 13** 計算ノードに接続するためのリンク

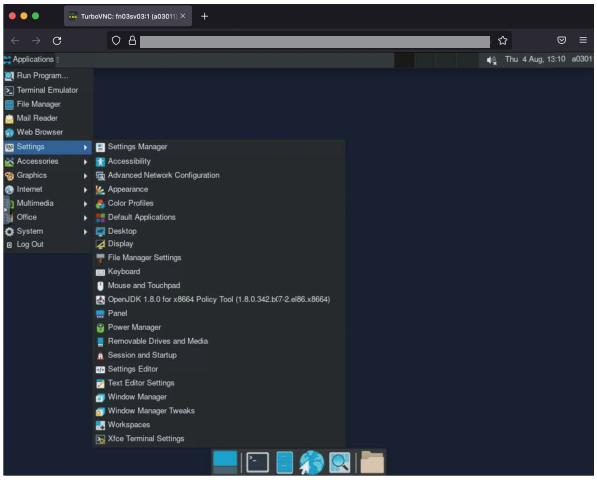


**図 14** リモートデスクトップ

すると，図 12-(b) に動的に切り替わる．図 12-(b) のリソースグループの項目についても同様であり，例えば「gpu1」から「mem1」に変更すると，図 12-(b) から「Number of GPUs」の項目は見えなくなり，各項目の最大値は表 1 で示した通りに設定される[*3]．図 12 の「Email」にメールアドレスを入力すると，ジョブが計算ノード上で起動した際に，そのメールアドレスに通知が行われる[*4]．図 12 の「Launch」をクリックすると，指定したシステムにジョブが投入される．そのジョブが計算ノード上で実行されると，その計算ノードに接続するためのリンクがユーザに表示される（**図 13**）．圧縮率と品質を設定した後，「Launch Remote Desktop」をクリックすると，リモートデスクトップの画面がユーザの Web ブラウザに表示される（**図 14**）．

## 5. まとめと今後の課題

本稿では，HPC クラスタが持つ計算資源を簡易に利用可能にする Open OnDemand を「富岳」に導入するため，「富岳」のジョブスケジューラである Fujitsu TCS を利用するためのアダプタの作成を行った．Open OnDemand を「富岳」に導入したことにより，ユーザの利便性を向上させる事ができたと考えている．

今後の課題としては，「富岳」上で Open OnDemand を実サービスとして提供し，その導入の効果を定量的に明らかにすることが必要である．また，「富岳」のアカウント数は 2,822（2022 年 7 月時点）であるため，Open OnDemand の利用者が多くなった場合，複数台の Open OnDemand サーバを用いた負荷分散の工夫が必要になると考えられる．さらに，対話的操作を伴う GUI アプリケーションは少ない CPU コア数の利用で十分な場合が多いため，oversubscribe を許可するスケジューリングを「富岳」に追加することにより，計算資源を有効に利用できると考えられる．Open OnDemand を用いると，様々な HPC クラスタに共通のフロントエンドでシステムを構築できるため，システムの開発コストの低減およびユーザに対する統一されたインタフェースの提供が可能になる．そのため，「富岳」における Open OnDemand の経験を今後も公開していくことで，Open OnDemand の普及を促進したいと考えている．

**謝辞** Open OnDemand の導入にあたりご助言いただいた富士通エンジニアの方々に感謝します．

[*3] 本稿で利用した Open OnDemand-2.0.27 では Dynamic Form Widgets のいくつかの機能にバグがあるため，想定通りには動作しなかった．`https://github.com/OSC/ondemand/issues/2115`

[*4] 認証サーバからユーザ名とメールアドレスが紐付いた情報を取得可能であれば，テキスト入力ではなくチェックボックスを用いる方が便利である．

## 参考文献

[1] Riken center for computational science. `https://www.r-ccs.riken.jp/en/`.
[2] Mitsuhisa Sato et al. Co-Design for A64FX Manycore Processor and "Fugaku". In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 651–665, Los Alamitos, CA, USA, 2020. IEEE Computer Society.
[3] Fujitsu software technical computing suite (in japanese). `https://www.fujitsu.com/downloads/JP/jsuper/tcs-v4-datasheet.pdf`.
[4] Slurm. `https://slurm.schedmd.com/`.
[5] Dave Hudak et al. Open ondemand: A web-based client portal for hpc centers. *Journal of Open Source Software*, Vol. 3, No. 25, p. 622, 2018.
[6] Ohio Supercomputer Center. `https://www.osc.edu`.
[7] Supercomputing. Seamlessly. Open, Interactive HPC Via the Web. `https://github.com/OSC/ondemand`.
[8] Settlage, Robert et al. Open ondemand: Hpc for everyone. In *High Performance Computing: ISC High Performance 2019 International Workshops, Frankfurt, Germany, June 16-20, 2019, Revised Selected Papers*, pp. 504–513, 2019.
[9] 中田 秀基 他. 大規模計算クラスタにおけるユーザ利便性向上. 研究報告ハイパフォーマンスコンピューティング（HPC）, No. 2020-HPC-174, 2020.
[10] 野村 哲弘 他. Tsubame3 のインタラクティブ利用の利便性向上にむけた取り組み. 研究報告ハイパフォーマンスコンピューティング（HPC）, No. 2020-HPC-175, 2020.
[11] Sylabs. `https://sylabs.io/`.
[12] A64fx microarchitecture manual. `https://github.com/fujitsu/A64FX/`.