外部計算資源の利用について

リリース ***1.0.1***

SIP-MI

2021 年 10 月 21 日

目次:

第1章　はじめに

MInt には、ワークフローを構成するモジュール内の一部分の処理を、MInt の計算ノード以外の「外部計算機」に行わせる機能がある。本機能によって、ユーザには下記に挙げる利点がある。

- 秘匿プログラムの使用
- 秘匿データの使用
- 特殊構成 (MInt の計算ノードでは対応できない) の計算機を使用できる
- 商用ソフトの使用 (MInt の計算ノードにも商用ソフトがインストールされているが、ライセンスの規定上、ほとんどの場合 NIMS 外からは利用できない)

外部計算資源の利用に際しては、MInt、外部計算機の双方が後述のセキュリティ水準を満たす必要がある。

1. MI コンソーシアム会則（秘密保持誓約書含む)、MInt システム利用規定など、MInt 利用に関わる契約・規定を遵守すること。
2. MInt 側は、下記のセキュリティ対策を実施すること。
    - 第三者による MInt のセキュリティ分析・セキュリティリスク診断を実施し、リスクを避ける設計を維持すること。
    - MInt を構成するサーバの OS・ミドルウェア・ライブラリ等に対し、継続的に脆弱性データベースを確認し、必要なアップデートを実施すること。
    - 不正アクセス監視やネットワーク負荷監視を実施すること。
3. 外部計算機側は、外部計算機として利用されるコンピュータに対し、十分なセキュリティ対策を実施すること。継続的に利用する場合には、定期的に対策状況を確認し、セキュリティレベルを維持すること。

外部計算資源利用には、SSH 方式と WebAPI 方式がある。前者は MInt から外部計算機へ SSH で必要なデータとコマンドをプッシュする方式である。単純で、外部計算を遅延なく開始できる利点があるが、外部計算機側で MInt に対し SSH のポートを開放してプッシュを受け入れる必要がある点は、特に企業ユーザではハードルが高いことが想定される。これに対し、後者は数分間隔で外部計算機側から MInt に WebAPI(https) でポーリングし、処理すべき案件が存在した場合は、必要なデータとコマンドがプルされる方式である。この方式では外部計算機側にポート開放の必要が無いが、外部計算の開始までにポーリング間隔に相当する遅延が生じる。

本機能は外部計算機内にユーザが持つ秘匿データの扱いにも十分な配慮が行われている。まず、ユーザが持つ秘匿データに関して、MInt が収集する情報はワークフローの各モジュールの入出力ポートの情報のみであり、モジュールの内部で完結する本機能のために、モジュールと外部計算機の間で送受信される情報は収集対象外である。外部計算機側は、MInt にあらかじめ定められたコマンドのみ実行できるように設定することができる。また、MInt に処理結果を返送する前に不必要なデータをワーキングディレクトリから削除することができる。

上記の機構によって、安全な外部計算が保証される。下記の各章で具体的な利用方法について記す。また、外部計算資源の利用に際して本書では不明な点は、ユーザと MInt 運用チームとの協議で決定するものとする。

第2章　概要

2.1 利用イメージ

外部計算資源の利用イメージを (図 2.1) に示す。

- MInt は NIMS 構内の DMZ[1] に存在する。
- ユーザは MInt 上に、外部計算を利用するモジュールを含んだワークフローを持つ。当該モジュールやワークフローの参照・実行権限は自機関内などに限定できる。
- ユーザが当該ワークフローを実行すると、外部計算を利用するモジュールで一部の処理が外部計算機に受け渡される。
- 外部計算機は処理の過程で、MInt に置けないデータやプログラムにアクセスできる。これらのアクセスを外部計算機の内部で完結させることで、安全な利用が可能となる。

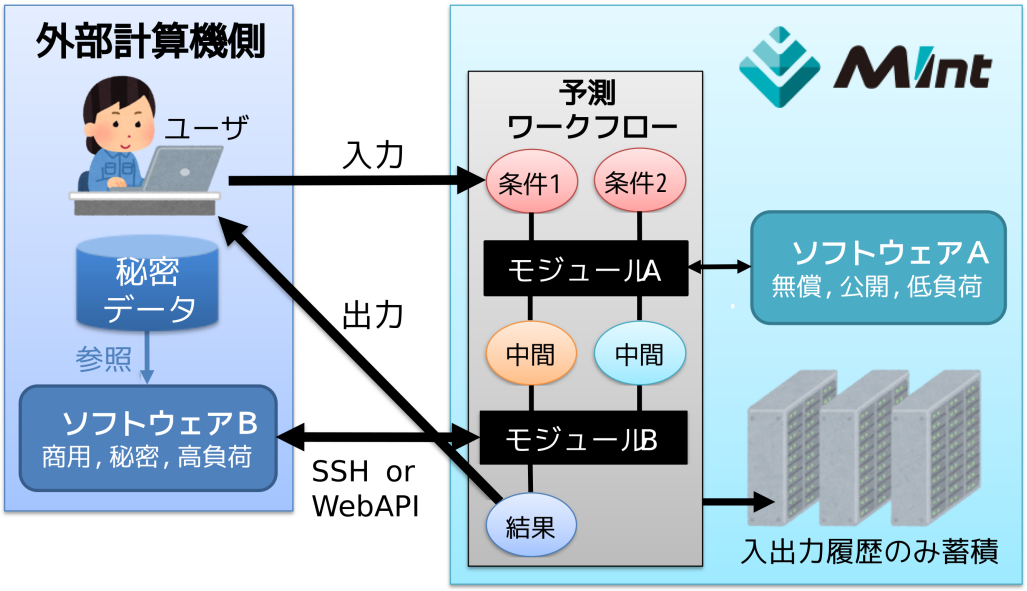

図 2.1: 外部計算資源の利用イメージ

[1] 物理的には NIMS 構内のサーバ室に存在するが、ネットワーク的には機構内 LAN とインターネットの双方からファイアウォールで切り離された領域。

2.2 SSH 方式と WebAPI 方式の比較

- **SSH 方式**
  - MInt から SSH で外部計算機にアクセスし、必要なファイルとコマンドをプッシュし、コマンドを発行し、結果を得る。
  - ファイルは内部で rsync -av を利用して送受信され、サイズは無制限である。
  - コマンドラインなどの文字列は Base64 エンコード無しで送受信される。
  - 外部計算機側 SSH サーバのポート (TCP/22 以外でも可) のインバウンドアクセスの開放が必要である。
  - 通信障害には弱い。計算中などで通信が切れると復帰できず、本バージョンでは実行プロセスが簡易なため計算続行が不可能である。
- **WebAPI 方式**
  - 外部計算機から MInt の WebAPI サーバにポーリングを行い、要処理案件の有無を確認する。ポーリング間隔は数分程度を想定している。案件があれば必要なデータとコマンドをプルし、自らコマンドを実行し、API で結果を送信する。
  - ファイルは Base64 エンコードされ、サイズはエンコード後に 2GiB[2] 未満である必要がある。
  - コマンドラインなどの文字列は Base64 エンコード無しで送受信される。
  - MInt の WebAPI サーバへの https(TCP/50443) のアウトバウンドアクセスの許可が必要である。
  - サーバー側クライアント側で状態を保持しているため、通信障害が起きても再接続と計算続行が可能である。

[2] GiB はギビバイトといい、コンピュータの容量や記憶装置の大きさを表す情報単位の一つである。1GiB は 2 の 30 乗バイトであり、1,073,741,824B である。

第3章　動作原理

3.1 SSH 方式

3.1.1 動作イメージ

SSH 方式での外部資源利用のイメージを (図 3.1) に示す。

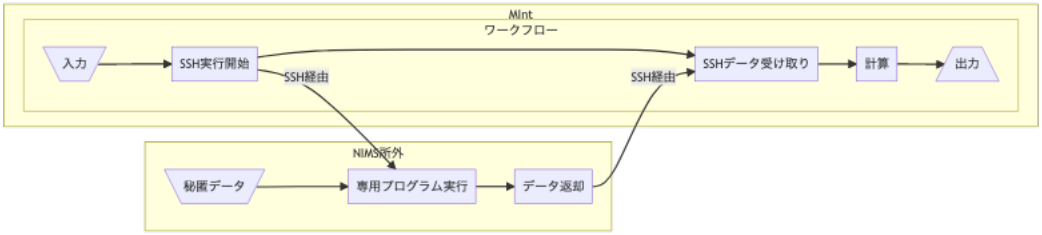

図 3.1: SSH 方式の外部資源利用のイメージ

3.1.2 ワークフロー例

SSH 方式の外部資源利用を含むワークフローを、MInt のワークフローデザイナで表示した例を示す。赤枠の部分が遠隔実行の行われるモジュールである。なお、本ワークフローは動作検証用サンプルとして、4章の**使用方法** で説明するインストール資材に含まれている。

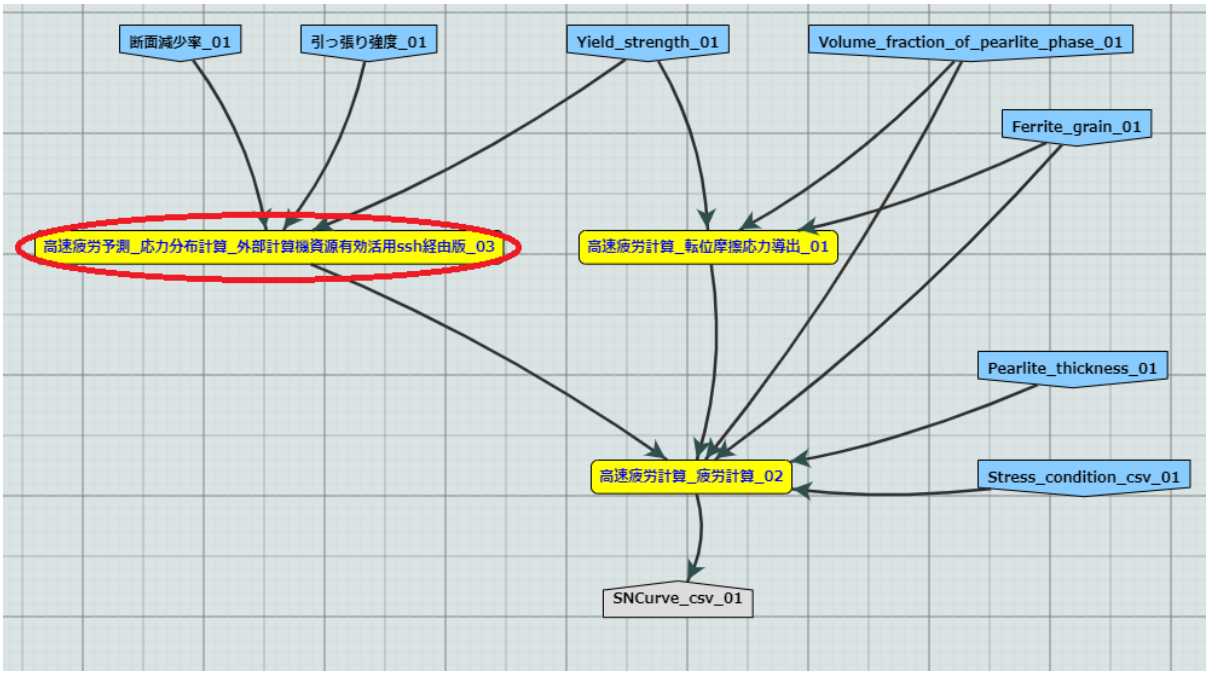

図 3.2: 動作検証用のワークフロー

3.1.3 モジュール内の処理

外部資源利用を行うモジュール内で、外部計算機側の処理が実行されるまでの流れを下記に示す。

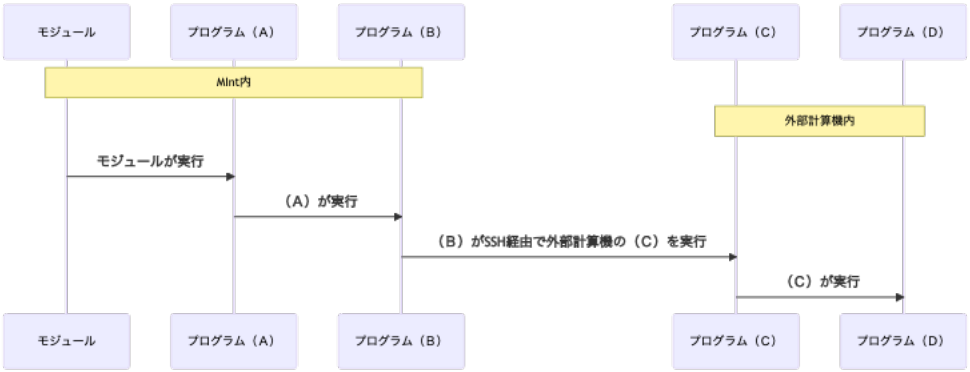

図 3.3: SSH 接続経由によるコマンド実行の流れ

- モジュール
  - MInt のワークフローシステムによって実行されるモジュール
  - プログラム（A）を実行する
- プログラム（A）
  - モジュールによって実行されるプログラム
  - モジュール固有の前処理を行う。
  - モジュールごとに任意の名前で用意する。
  - 4 章の**使用方法** で説明する編集を行う。
  - （B）を実行する。
- プログラム（B） このプログラムが外部計算機と通信を行う。
  - 外部計算の準備を行う。
  - 名前は任意の名前を使用可能。
  - テンプレートは **execute_remote_command.sample.sh** をコピーして使用する。
    - 4 章の**使用方法** で説明する編集を行う。
    - （A）が実行するプログラム名とコピーしたプログラム名は同名としておく。
  - SSH 経由で（C）を実行する。
    - 送信するファイルはパラメータとして記述する。
    - 外部計算機上の一時ディレクトリ[3] の内容を全部受信するため、MInt に送信しないデータは外部計算機側で（C）の実行終了前に削除する。
- プログラム（C）
  - 名前は プログラム（B）用のテンプレートで**execute_remote-side_program_ssh.sh** となっているが変更可能である。
    - 同名のテンプレートが用意されているので、複雑な処理を必要とする場合は、コピーして使用する。

[3] 外部計算機では、処理は/tmp などに作成した一時ディレクトリで実行される。

        * （B）で実行されるプログラム名とコピーしたプログラム名は同名としておく。
- プログラム（D）
  - 外部計算機上のプログラムを（C）のみで完結させ、本スクリプト群は用意しない運用も可能である。

3.2 WebAPI 方式

3.2.1 動作イメージ

WebAPI 方式での外部計算の実行イメージを (図 3.4) に示す。

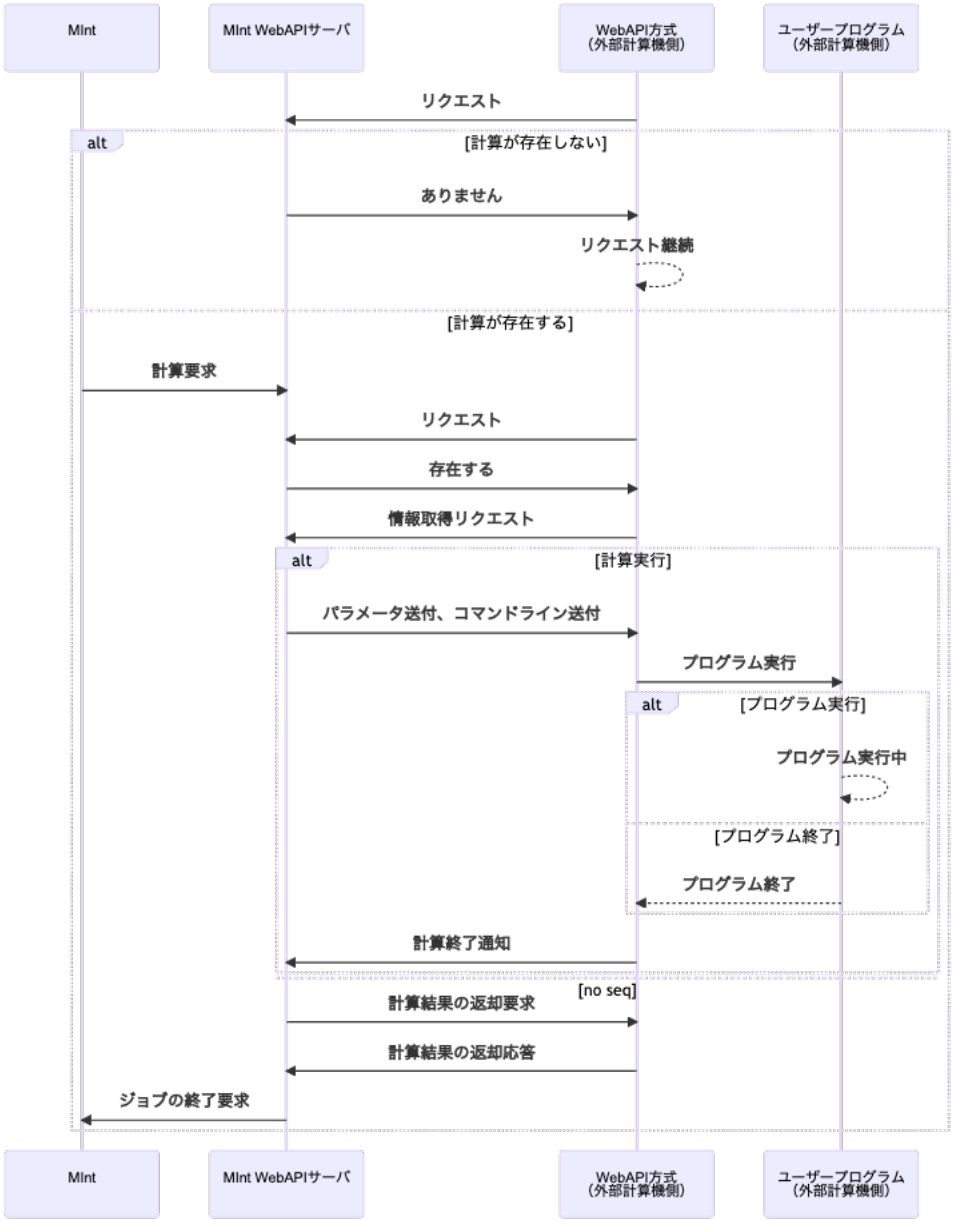

図 3.4: WebAPI 方式の流れ

3.2.2 ワークフロー例

WebAPI 方式の外部資源利用を含むワークフローを、MInt のワークフローデザイナで表示した例を示す。赤枠の部分が遠隔実行の行われるモジュールである。なお、本ワークフローは動作検証用サンプルとして、4 章の**使用方法** で説明するインストール資材に含まれている。

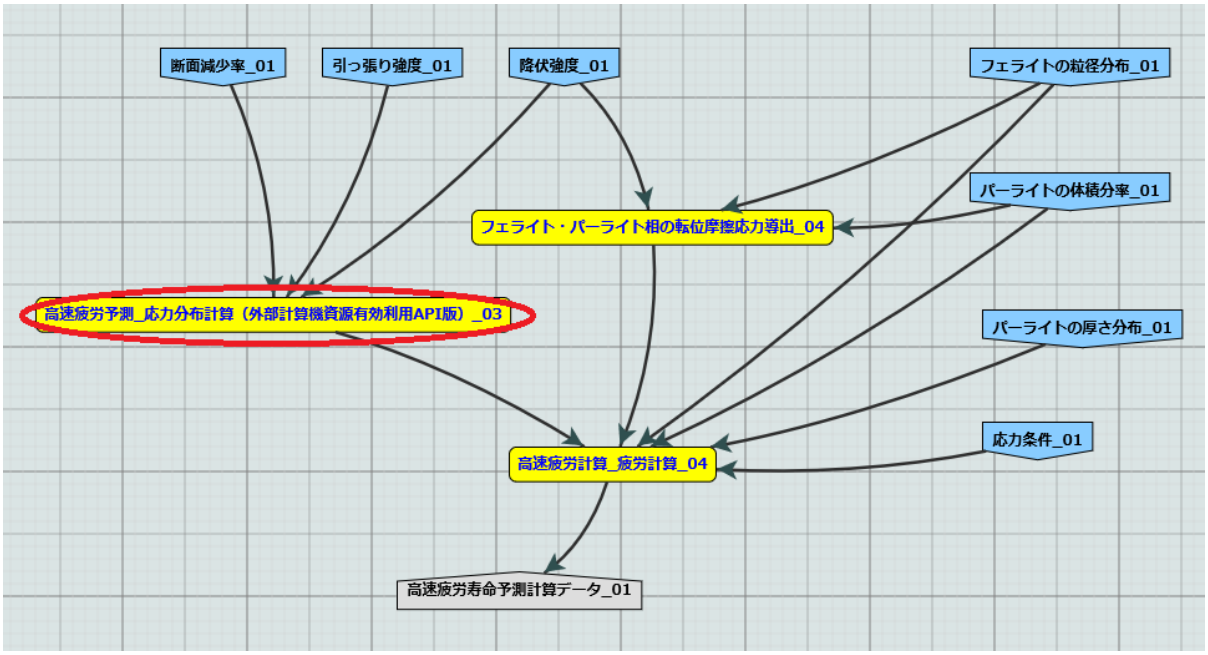

図 3.5: 検証用ワークフロー

※赤枠の部分が外部計算資源を利用するモジュールである。

3.2.3 モジュール内の処理

ワークフローの当該モジュール内で外部計算機側の処理が実行されるまでの流れを下記に示す。

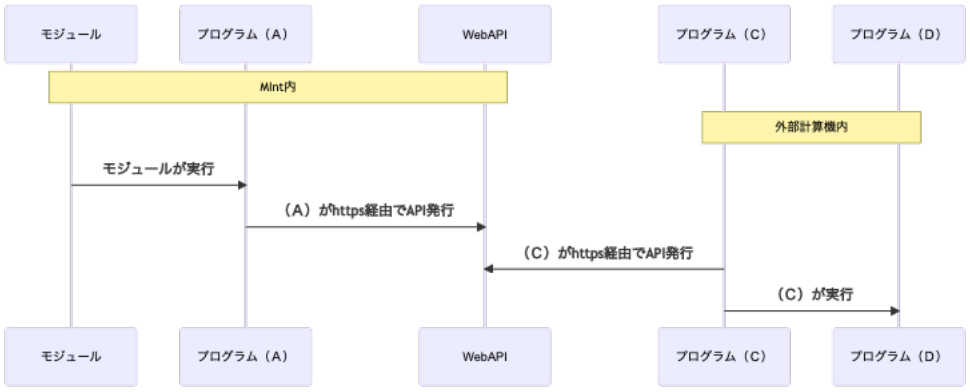

図 3.6: WebAPI 方式でのコマンドの流れ

- モジュール
  - MInt のワークフローシステムによって実行されるモジュール
  - プログラム（A）を実行する
- プログラム（A）
  - モジュールによって実行されるプログラム。モジュールごとに任意の名前で用意する。
  - モジュール固有の前処理を行う。
  - **misrc_distributed_computing_assist_api/debug/mi-system-side/mi-system-wf.py** を実行しておく。- WebAPI へ計算の情報が登録される。- 以降このプログラムが外部計算機資源側（以下の（C）と API を介して計算を行う）
- WebAPI (このプログラムが MInt システムと外部計算機との通信を中継する。)
  - **外部計算の準備を行う。**
    - 送受信するファイルはパラメータとしてあらかじめ設定しておく。
  - WebAPI 経由で（C）からのアクセスを受け付ける
  - （A）から計算の情報登録が無い限り、（C）からアクセスがあっても計算は始まらない。
  - ワークフローを実行したユーザーのトークンと（C）からのトークンが合致しないと（C）は適正な通信相手とならない。
- プログラム（C）
  - ポーリングプログラムである。
  - **misrc_distributed_computing_assist_api/debug/remote-side/mi-system-remote.py** を実行しておく。
  - 外部計算機上で実行するプログラム名は、このプログラム経由で MInt システムから受信され、このプログラムが実行する。
  - 認証情報はこのプログラム（C）が使用する。認証情報が無いと WebAPI にアクセスできない。詳細は 4.4.1 章の**認証関連情報の用意** で説明する。
- プログラム（D）
  - （C）から実行される外部計算用スクリプト。

- 名前は任意。（プログラム（C）経由で伝えられるため、あらかじめ MInt システム側に設定が必要）
- **execute_remote_command_api.sh** を参考にして作成しておく。

第4章　使用方法

SSH 方式、WebAPI 方式それぞれのインストールおよびプログラムの実行までを外部計算機側で作業が必要な項目について説明する。なお、外部計算機側は bash スクリプトと Python スクリプトの動作する Linux ホストを想定しているが、MInt 側との通信が正常に確立できるならば、これ以外の環境でも構わない。その場合はパスワードなしログインの設定などは環境に合わせて適宜同様の処置を行うことになる。また、外部計算機側で秘匿データを扱う際は、これに関する仕様を MInt 側に開示する必要も無い。

4.1 事前決定事項

事前に決定しておく項目は以下の通り。

1. 環境構築
   - 外部計算機側, MInt システムのユーザアカウントの準備
   - SSH or WebAPI の方式選択
   - 認証関連情報の用意
2. ワークフロー・モジュールの仕様策定 (実装調査書の作成)
   - MInt と外部計算機の役割分担の決定
   - MInt と外部計算機の間を受け渡すパラメータ・ファイルの設計
   - MInt 側の前処理・後処理の設計
   - 外部計算機側スクリプトの設計

4.2 SSH, WebAPI 方式共通

4.2.1 資材の入手

外部計算資源の利用に必要な資材は GitHub 上のリポジトリ[5] https://github.com/materialsintegration に用意されている。

- misrc_remote_workflow
  - 主に外部計算機側で実行されるスクリプトのサンプルが同梱されている。
- misrc_distributed_computing_assist.api
  - WebAPI 方式用のプログラムおよびサンプルが同梱されている。
  - MInt 側資材は **debug/mi-system-side** 、外部計算機側資材は **debug/remote-side** にある。
- ワークフローの外部実行に必要なリポジトリ
  - 上記リポジトリのサンプルスクリプト以外に外部実行機能のあるワークフローを利用する場合に必要なリポジトリ
  - 通常は github にアップされていないので、MInt 運用チームに依頼して入手する。

注釈: 特別なリポジトリを利用する場合の利用方法については別途 MInt 運用チームまで問い合わせること。

---

[5] 本機能を実現する資材などを格納したサーバ。GitHub を利用しているが、アカウントが無くともダウンロードは可能である。MInt 運用チームがアカウントを発行したユーザのみアップロードが可能である。

ユーザは外部計算機上にこれらを展開し、必要なカスタマイズを行う。資材展開後の外部計算機側のディレクトリ構造は以下のようになる。

- ユーザーディレクトリ

```
~/ユーザーディレクトリ
  + remote_workflow
    + scripts
      + input_data
  + misrc_distributed_computing_assist_api
    + debug
      + remote-side
```

- ワーキングディレクトリ

```
/tmp/<uuid>
```

注釈: ワーキングディレクトリは展開した資材のうち、外部計算機資源側のプログラムが自動的に作成、使用するため、ユーザーは意識しなくて良い。

ユーザが外部計算機側でカスタマイズするファイルは、通常、SSH 方式では **misrc_remote_workflow/scripts/execute_remote-side_program_ssh.sample.sh** 、WebAPI 方式では事前に名称を決めホスト情報ともども登録しておく。カスタマイズの方法については後述する。これ以外のファイルも改変可能であるが、その改変が原因で外部計算を利用するワークフローが動作しなかった場合、MInt 運用チームは責を負わない。
資材展開後の MInt 側のディレクトリ構造は以下のようになる。

- ユーザーディレクトリ

```
~/misystem ディレクトリ
 + remote_workflow
   + scripts
 + misrc_distributed_computing_assist_api
   + debug
     + mi-system-side
```

- **ワーキングディレクトリ**
  - 複雑なので省略する。

4.2.2 ワークフローサンプル

misrc_remote_workflow/sample_data に、Abaqus 実行環境が用意可能な場合に使用可能なワークフローおよび、そのサンプル入力ファイルが用意されている。これを利用して、MInt 側と外部計算機側のテストが可能である。また、**misrc_remote_workflow/scripts** に、この時のモジュール実行プログラムがある。これを参考に、他のモジュール実行プログラムを作成することが可能である。利用する際には MInt システム側と調整が必要である。

- **kousoku_abaqus_ssh_version2.py** : SSH 方式のモジュール実行スクリプト
- **kousoku_abaqus_api_version2.py** : WebAPI 方式のモジュール実行スクリプト

もっと簡易な例題ワークフローは現在準備中である。

4.3 SSH 方式

主に、外部計算機資源側の準備について記述する。

4.3.1 公開鍵の用意

パスフレーズ無しの公開鍵認証を原則とする。MInt 運用チームに依頼して、パスワードなしログイン用の公開鍵を入手し、以下の手順に沿ってファイルを作成しておく。SSH 方式は MInt システムから外部計算機資源への一方通行の SSH 通信で行われるため、パスワード無しログイン設定を外部計算機資源側に実施する。

```
$ cd .ssh
$ cat <入手した公開鍵暗号ファイル> >> authorized_keys
$ chmod 600 authorized_keys
```

注釈: ワークフロー実行前に MInt 運用チームに連絡してパスワードなしログインが可能なことを確認すること

4.3.2 資材の展開

1. **misrc_remote_workflow** リポジトリを展開する。

```
$ git clone https://github.com/materialsintegration/misrc_remote_workflow.
↪git
$ cd misrc_remote_workflow
$ ls
README.md  documents  inventories  misrc_remote_workflow.json  modulesxml ␣
↪sample_data  scripts
$ cd scripts
$ ls
abaqus                                      execute_remote_command.sample.sh
 kousoku_abaqus_ssh.sh
create_inputdata.py                         input_data
 kousoku_abaqus_ssh_version2.py
execute_remote-side_program_api.sample.sh   kousoku_abaqus_api_version2.py
 kousoku_abaqus_ssh_version2.sh
execute_remote-side_program_ssh.sample.sh   kousoku_abaqus_api_version2.sh
 remote-side_scripts
execute_remote_command.sample.py            kousoku_abaqus_http.py
```

2. 外部計算機側で実行するスクリプトがあれば **remote-side_scripts** に配置する。
3. MInt が外部計算機へログインして最初に実行するプログラム名は前述のとおり **execute_remote-side_program_ssh.sh** に固定されている。このため **execute_remote-side_program_ssh.sample.sh** をこの名前でコピーするか、新規に作成して、必要な手順をスクリプト化する。

4.3.3 (参考)MInt 側作業

1. 外部計算資源を利用するモジュールが実行可能なスクリプトを **misrc_remote_workflow/scripts/execute_remote_command.sample.sh** をコピーして専用スクリプトを作成する。
2. 予測モジュールの modules/resouceRequest/pbsNodeGroup タグに ssh-node01 という値をセットする。
3. 予測モジュールの modules/objectPath タグに 1. で作成したスクリプトをセットする。
4. 1. で作成したスクリプトを各行のコメントに従い適宜修正する。
5. 1. を実行可能な予測モジュールを組み込んだワークフローを作成する。

4.4 WebAPI 方式

主に、外部計算機資源側の準備について記述する。

4.4.1 認証関連情報の用意

MInt 側担当者に問い合わせて下記の情報を用意する。

- ホスト情報
  - MInt 側で API の発行者を識別するための文字列。ユーザ企業のドメインなどと一致させる必要は無い。
- API トークン
  - MInt の API 認証システムを使用するためのトークン。MInt システムログイン後、ユーザープロファイル管理システムメニューで表示される、「API トークン」を使用する。
    - ユーザーの環境でポーリングスクリプトを動作させるときに必要であるが、後述のログイン方式を利用する場合はトークン自体は必要ない。
- MInt の URL
  - MInt の URL(エンドポイントは不要) を、MInt 運用チームに問い合わせておく。
- WebAPI 方式を利用できるように MInt 運用チームに設定を依頼する。

4.4.2 資材の展開

1. **misrc_distributed_computing_assist_api** リポジトリを展開する。

```
$ git clone https://github.com/materialsintegration/misrc_distributed_
↪computing_assist_api.git
$ cd misrc_distributed_computing_assist_api
$ ls
README.md  logging.cfg     mi_dicomapi_infomations.py           syslogs
debug      mi_dicomapi.py  mi_distributed_computing_assist.ini
$ cd debug
$ ls
api_status.py  api_status_gui.py  api_status_gui.pyc  mi-system-side ␣
↪remote-side
$ cd remote-side
$ ls
api-debug.py  debug_gui.py  mi-system-remote.py
```

2. **authentication_operator** リポジトリを展開、環境変数を設定する。（ログイン方式を選択する場合）

```
$ git clone https://gitlab.mintsys.jp/midev/authentication_operator.git
$ export AUTHENTICATION_OPERATOR=<path to authentication_operator
```

注釈: 環境変数 AUTHENTICATION_OPERATOR はログインシェルの自動設定ファイルに設定しておく。

3. 計算に必要なスクリプトの準備
   - ファイル名は MInt 運用チームに伝えて、API に登録しておく。
   - 特別なリポジトリを利用する場合はこの作業が必要ないこともある。

4.4.3 実行

認証情報と共にポーリングプログラムを動作させておく。事前に設定した情報に従って MInt システム側と通信し、入力ファイルの受信、計算、出力ファイルの送信が自動的に行われる。認証情報が無い、間違っている、などの場合はポーリングは失敗し、計算は行われない。また **ワークフローを実行したユーザーと同じユーザーのトークンまたはログイン方式での同じユーザー** で実行しないとこちらもポーリングは失敗し、計算は行われない。

1. 以下 2. または 3. のどちからの方法で、**mi-system-remote.py** を実行する。
2. トークン指定方式

```
$ python mi-system-remote.py <ホスト情報> https://nims.mintsys.jp <API token>
site id = <ホスト情報> で指定した識別子
base url = https://nims.mintsys.jp:50443
```

3. ログイン方式

```
$ python mi-system-remote.py <ホスト情報> https://nims.mintsys.jp login
site id = <ホスト情報> で指定した識別子
base url = https://nims.mintsys.jp:50443
nims.mintsys.jp へのログイン
ログイン ID: <MInt システムのログイン名>
パスワード: <同、パスワード>
 token = <ログイン名のトークンの表示>
...
```

注釈: ホスト情報と API token は 4.4.1 章の**認証関連情報の用意** で入手したそれぞれの認証情報を指定する。

4.4.4 (参考)MInt 側作業

1. **misrc_distributed_computing_assist_api** リポジトリを展開する。
2. 構成ファイル **mi_distributed_computing_assist.ini** に必要な設定を行う。
3. **mi_dicomapi.py** を動作させて待ち受け状態にする。

```
$ python mi_dicomapi.py
```

または

```
$ systemctl start distcomp_api
```

4. モジュールの実行プログラム内で、**misrc_distributed_computing_assist_api/debug/mi-system-side/mi-system-wf.py** を必要なパラメータとともに実行するように構成する。

注釈: ワークフロー側から計算登録時に構成ファイルは再読込されるので、WebAPI プログラムが現在動作中であっても読み込ませるための特別な動作は必要ない。

4.5 その他 MInt 側注意事項

SSH 方式、WebAPI 方式共通の注意事項など。

- pbsNodeGroup 設定で ssh-node01 を設定する。他の計算機では外へアクセスすることができないため。
- pbsQueue など CPU 数などは指定できない。
- 外部計算機側で別途 Torque などのバッチジョブシステムに依存する。

4.6 エラーが発生した場合

ワークフローを本実行する前に MInt 運用チームと連携して動作確認を行っておくが、予期せず異常終了した場合などは以下の方法で対策を検討することができる。

- SSH 方式、WebAPI 方式ともワークフローの出力ポートとは別に外部計算機で計算が行われた際の処理のログがある。MInt 運用チームに連絡して、それを入手する。
- 同様に、外部計算機資源側で 4.2.1 章 **資材の入手** で説明したワーキングディレクトリに計算結果およびログが残っているのでこれを利用する。
  - ワーキングディレクトリ[6] は UUID で構成されたディレクトリ名のディレクトリの作成時間などで該当ディレクトリかどうか判断する。

4.6.1 通信異常

インターネット経由であるので、通信異常は発生するものとして対処してある。WebAPI 方式では外部計算機側からの通信となるため、外部計算機側のプログラムでリトライ方式を採用している。デフォルトはリトライ間隔 60 秒のリトライ回数 5 回で通信失敗として終了する。この値は以下の書式で上書き指定することも可能である。

```
$ python mi-system-remote.py <ホスト情報> https://nims.mintsys.jp <API token> retry:<リトライ回数>,<リトライ間隔>
```

[6] 外部計算機側のワーキングディレクトリは/tmp ディレクトリに作成されるので、OS の設定に変更がなければ 30 日後に削除される。このため問題が発生した場合は発生から 30 日以内に調査を開始する必要がある。

SSH 方式ではその性質上処理中に通信異常が起きると復帰できない。現バージョンでは SSH 方式での処理中の通信異常を復帰させる手段は実装されていない。どちらの場合も通信が途絶えて処理続行不能と判断されれば、MInt システム側に異常を通知し、異常終了となる ように構成されている。

注釈: リトライ回数は整数で指定し、リトライ間隔は整数または実数で指定する。

4.7 ワークフローの廃止

ユーザが MInt 運用チームにワークフローの廃止届を提出する。当該ワークフローは MInt 上で「無効」のステータスを付与され参照・実行不能となる。

以上