
外部計算資源の利用について

リリース **0.1.0**

SIP-MI

2021 年 03 月 11 日

目次:

第 1 章	概要	1
1.1	扱う方式	1
1.2	使用するリポジトリ	1
1.3	SSH アクセスのために	2
1.4	WEB API でのアクセスのために	2
1.5	実行されるコマンド	2
第 2 章	外部計算機資源を MInt システムから有効に活用するための手法とは	3
2.1	SSH を利用した遠隔実行	3
2.1.1	概要	3
2.1.2	実行のイメージ	3
2.1.3	システム要件	4
2.1.4	MInt 側詳細	4
2.1.5	外部計算機資源の詳細	5
2.1.6	用意されているサンプルワークフロー	5
2.1.7	外部計算機でのディレクトリ	6
2.1.8	コマンドの流れ	6
2.1.9	MInt システムと送受信されるデータ	7
2.2	API を利用したポーリング方式	8
2.2.1	概要	8
2.2.2	実行のイメージ	9
2.2.3	ポーリングシステムの流れ	10
2.2.4	システム要件	11
2.2.5	用意されているサンプルワークフロー	11
2.2.6	MInt システムでのディレクトリ	13
2.2.7	外部計算資源でのディレクトリ	13
2.2.8	外部計算機で MInt システムから実行されるプログラム	14
2.2.9	MInt システムで送受信されるデータ	14
第 3 章	使用方法	15
3.1	事前決定事項	15
3.1.1	API 方式の場合の設定事項	16
3.2	SSH 方式	17

3.2.1	外部計算機資源側	17
3.2.2	MInt システム側	17
3.3	API 方式	18
3.3.1	外部計算機資源側	18
3.3.2	MInt システム側	18
3.4	ワークフローについて	18
3.4.1	共通事項	19
3.4.2	SSH 方式	19
3.4.3	API 方式	19
3.4.4	サンプル	19

第 1 章

概要

MI システムから任意のモジュールの計算を MI システム外の計算機 (これを外部計算機資源と言う) を使用してワークフローを実行することを外部計算機資源の有効活用として SIPI 期の検証項目として実装し動作検証を行った。本システムはこの検証を踏まえ、より簡易に実装が可能なシステムを構築した。

- NIMS DMZ に配置した MInt システムを対象とする。
 - SSH 等で外部の計算機を計算資源として計算を行う
 - 専用 API システムを構築して SSH 接続が不可能な外部の計算機を計算資源として計算を行う
- 外部計算機資源においての秘匿データなどの扱い
 - 秘匿データの指定は簡易な方法
 - MInt システム側からはその存在は感知できない

本ドキュメントは外部計算機資源の有効活用について、動作原理などを説明し、次いで簡単にインストール、実行する方法を説明する。インストール、実行は MInt システム側と外部計算機資源側に分かれている。

1.1 扱う方式

本書で扱う外部計算機資源の有効化強うの方法は以下の 2 つの形式である。

- ssh などで MI システムの任意のモジュールから直接遠隔計算を実行する。
- WEBAPI を利用して MI システムの任意のモジュールからポーリング形式で遠隔計算を実行させる。

1.2 使用するリポジトリ

外部計算機資源の有効利用のために、以下 2 つのリポジトリを用意してある。外部計算機資源側はこれらを外部資源計算用計算機に配置し、必要なコマンドを埋め込むだけでよい。

- `misrc_remote_workflow`
 - 主に外部計算機資源側で実行されるスクリプトのサンプルが登録されている。
- `misrc_distributed_computing_assist.api`
 - API 方式のためのシステム構築用のプログラム、サンプルが登録されている。
 - API 本体は直下に
 - ワークフローで使用するプログラムは “`debug/mi-system-side`” に
 - 外部計算機側で使用するプログラムは “`debug/remote-side`” に
 - * ここから実行されるプログラムのサンプルは上記 `misrc_remote_workflow` リポジトリを使用している。

1.3 SSH アクセスのために

ワークフローから `ssh` コマンドを利用して、外部計算機にアクセスするために外部計算機を設置する企業または機関には `ssh` 接続が可能な処置が必要である。

1.4 WEB API でのアクセスのために

ワークフローから外部計算機を利用する場合、`ssh` でのアクセスが不可能な場合、本方式を使用するが、ワークフロー側は直接外部計算機にアクセスせず、外部計算機側で定期的に問い合わせ（ポーリング）する必要がある。ポーリングには通常の `https` 通信を用いる。

1.5 実行されるコマンド

SSH の場合、リポジトリにあるコマンドしか外部計算機上では実行しない。その場所も事前に取り決めた場所となる。API も同様である。

第 2 章

外部計算機資源を MInt システムから有効に活用するための手法とは

最初に各手法の動作原理を説明する。

2.1 SSH を利用した遠隔実行

最初に ssh を利用して、MI システムの任意のモジュールから外部計算機資源を利用する方法を説明する。

2.1.1 概要

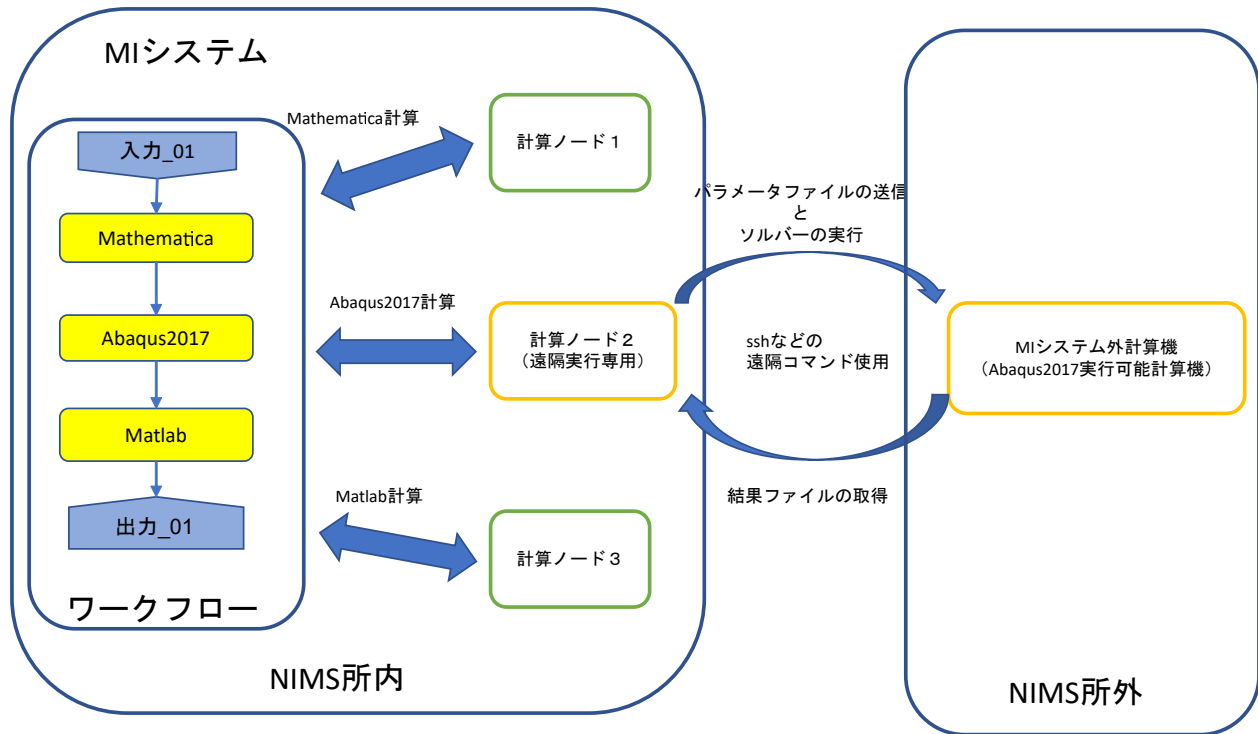
ssh を利用した遠隔実行とは、ssh プロトコルを利用してネット上でアクセス可能な場所にある計算機をあたかも MInt システムの計算機の一部として使用すること。この場合 ssh パケットが到達可能な場所であればどこでも対象となり得る。ssh アクセスではパスワードなしでの運用も可能であり、本システムも基本的にパスワードなし接続での運用が前提であるが、必ずしも必須ではない。スクリプト内で実現可能であればパスフレーズ付きなど多彩なアクセス方法を採用可能である。

図 2.1 SSH 実行のイメージ

2.1.2 実行のイメージ

この方式では、以下のようなシステムで動作サンプルが用意されている。

このようにして、特定のモジュール（Abaqus2017）と特定の計算ノード（計算ノード 2）を用意し、計算ノード 2 が MI システム外にある計算機を遠隔実行できるように設定して、使用することで MI システム外の計算機または計算機群を MI システム内にあるかのごとく計算（ワークフロー）を実行することが可能になる。また Abaqus2017 と謳ってはいるが実行するプログラムはこれに限らず、様々なコマンド、プログラム、アプリケーションを実行することが可能なように作られている。



2.1.3 システム要件

- MInt システム側。
 - 遠隔実行専用の計算ノードを設置してある。
 - 遠隔実行用予測モジュールを作成。
 - このモジュールは専用計算ノードを指定して計算を行うよう設計。
 - モジュールおよび専用計算ノードに ssh 操作の設定。
- MInt システム外（主に要件）
 - 外部から到達可能な場所。
 - 可能なら Linux（Mac でも可能。Windows は ssh 到達に問題があるため非推奨）。
 - 必要な資材を取得、展開。
 - 必要な情報を設定（主にソルバーパス、パラメータ、秘匿データの配置）

2.1.4 MInt 側詳細

専用計算ノードでは以下のような動作が行われるように、専用モジュールが定義するプログラムを実行する。

必要な資材は `gitlab` に登録してある。

- パラメータ類の遠隔計算機へ送信（遠隔計算機側にあるパラメータまたはファイルを指定することも可）。
- 遠隔計算機でソルバー（プログラム）の実行。
- 実行が終了したら結果ファイルの取得。

2.1.5 外部計算機資源の詳細

外部計算機資源側計算機では、必要なファイルの配置が主な手順である。

必要な資材は `github` に登録してある。

- 資材の展開
- ソルバーパスの調整
- 秘匿データ（ある場合）に指定のディレクトリへの配置

2.1.6 用意されているサンプルワークフロー

この方式ではサンプルとして下記のようなイメージの動作検証環境用ワークフローを用意した。

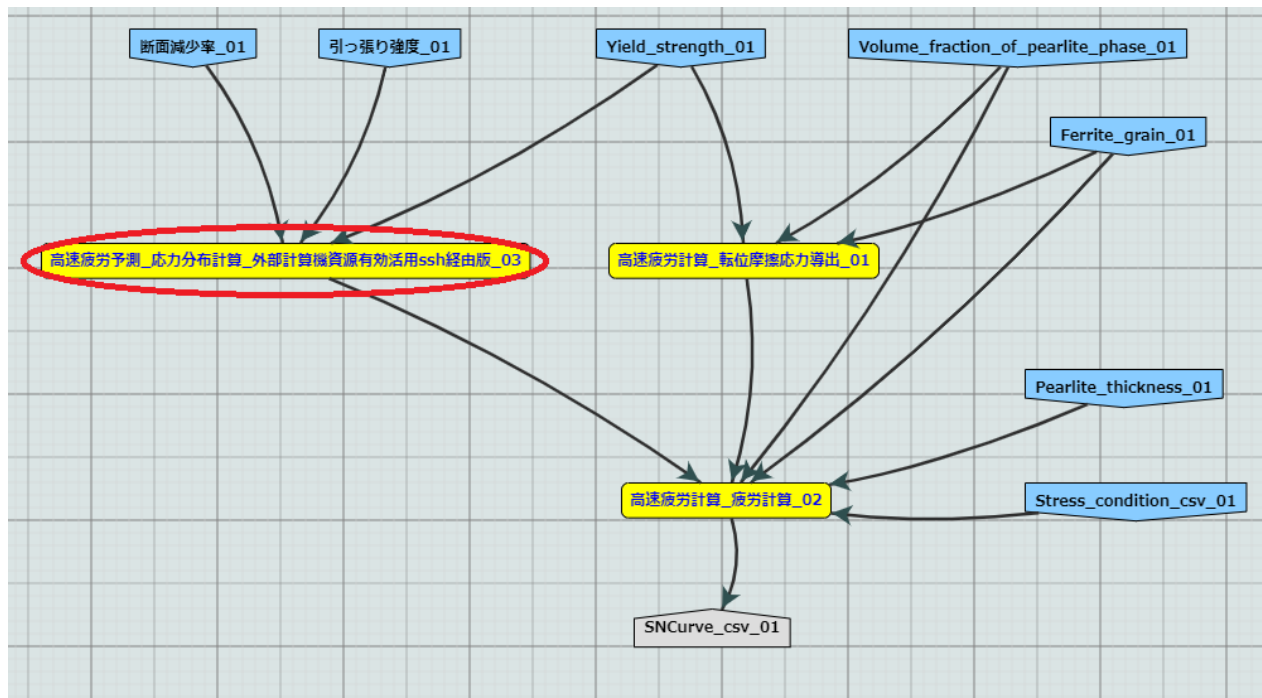


図 2.2 動作検証用のワークフロー

※赤枠の部分が遠隔実行が行われるモジュールである。

2.1.7 外部計算機でのディレクトリ

外部計算機のディレクトリ構造は以下のようにになっている。インストール方法については後述する。

- 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
```

- ワーキングディレクトリ

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

2.1.8 コマンドの流れ

ワークフローの該当モジュールから外部計算機のコマンドが実行されるまでの流れを下記に示す。

図 2.3 SSH 接続経路によるコマンド実行の流れ

- ワークフロー [予測モジュール]
 - MInt システムが実行する予測モジュール
 - (A) を実行する
- プログラム (A) : kousoku_abaqus_ssh_version2.sh (サンプル用)
 - MInt システムの予測モジュールが実行する。
 - 予測モジュールごとに用意する。名前は任意。使用方法 で説明する編集を行う。
 - 予測モジュール定形の処理などを行い、(B) を実行する。
 - * (B) の名前は固定である。
- プログラム (B) : execute_remote_command.sample.sh
 - (A) から実行された後、外部計算機実行のための準備を行い、ssh 経由で (C) を実行する。
 - 名前は固定である。このプログラムが外部計算機資源との通信を行う。
 - 使用方法 で説明する編集を行う。
 - * 送信するファイルはパラメータとして記述。
 - * (C) の名前は固定である。

- 受信するファイルは外部計算機資源上の計算用ディレクトリ^{*1} のファイル全部。
- プログラム (C) : `execute_remote-side_program_ssh.sh`
 - (B) から `ssh` で実行される。
 - 外部計算機で実行されるプログラムはここへシェルスクリプトとして記述する。
 - インストール時は `execute_remote-side_program_ssh.sample.sh`^{*2} となっている。
- プログラム (D) : `remote-side_scripts`
 - (D) から実行されるようになっており、いくつかのスクリプトを実行するよう構成されている。
 - サンプル専用であり、必ず使うものではない。(C) に依存する。

2.1.9 MInt システムと送受信されるデータ

MInt システムへ送受信されるデータは、「`execute_remote_command.sample.sh`」で決まっており、以下の通り。

- 送信されるデータ
 - 「`execute_remote_command.sample.sh`」にパラメータとして記述したファイル。(モジュール内)
- 返信されるデータ
 - 計算結果としての出力ファイル。
 - * 計算専用ディレクトリを作成して計算され、そのディレクトリ以下のファイルは全て
 - * このディレクトリでの計算は、「`execute_remote-side_program_ssh.sh`」で行われるので、返信不要のファイルはあらかじめこのスクリプト終了前に削除しておくようにスクリプトを構成しておく。

※ 秘匿データを配置してあるディレクトリまたはインストール後のセットアップで実行に必要なファイル、データとして指定されたものは MInt システムで感知できないこと、およびシステムの記録 (GPDB など) するための設定がなされていないため送り返されることは無い。

^{*1} 外部計算機では計算は `/tmp` などに一時的なディレクトリを作成し計算が実行される。

^{*2} 本システムでは、MInt システムは「`execute_remote_command.sample.sh`」を実行し、外部計算機で実行を行うプログラムとして「`execute_remote-side_program_ssh.sh`」を呼び出す。外部計算機側ではインストール後にこのファイル (インストール直後は、`execute_remote_program_ssh.sample.sh` という名前) を必要に応じて編集して使用することで、別なコマンドを記述することが可能になっている。

2.2 API を利用したポーリング方式

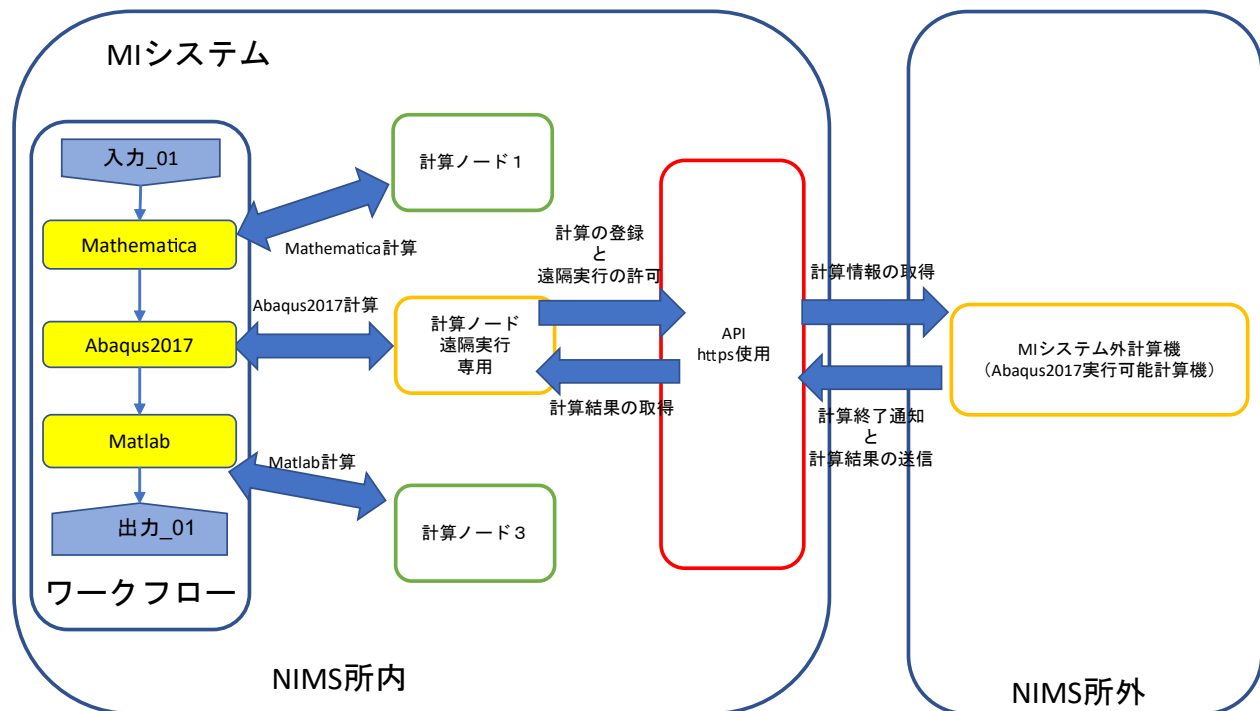
続いては API(MInt システムの API ではない) を利用したポーリング方式による方式の説明である。ssh などでも直接通信が行えない組織間でも http または https での通信は可能なことが多く、これを利用することで外部計算資源の有効活用できることを狙った。

2.2.1 概要

API を利用したポーリングシステムとは外部計算資源を ssh などでも直接操作するのではなく、中間に計算を仲介する API を立て、MInt システム側、外部計算資源側がその API を利用して http または https 通信で計算の依頼、実行などを行うシステムである。この場合、外部計算資源側、MInt システム側（予測モジュール）は計算工程の随所で定期的に通信する必要がある（ポーリング）ので、ポーリングシステムと言う。ssh の場合と比べて外部計算資源の利用および実行のための手続きが多くなり、用意するプログラムも複雑になる。

2.2.2 実行のイメージ

この方式では以下のようなシステムを想定している。



2.2.3 ポーリングシステムの流れ

この方式でのポーリングシステムのフロー概要。

図 2.4 ポーリングシステムの流れ

2.2.4 システム要件

この方式における必要な条件を記す。おもに外部計算資源側の条件となる。

- 双方で設定必要な事項
 - 実行可能な計算またはプログラム
 - 送受信するファイル
 - この情報を API がワークフローから遠隔計算機へ、遠隔計算機からワークフローへと受け渡す。
 - * 遠隔計算機へはコマンドとパラメータ
 - * ワークフローへは計算結果などのファイル
- MInt システム側
 - 外部計算資源有効利用用の計算ノードを設置してある。(以下専用計算機または専用ノードとする)
 - 外部計算資源有効利用モジュールを作成
 - このモジュールは専用計算機を指定して計算を行うよう実装する。
 - ポーリング用 API を実行する。MInt システムへ到達可能ならどこでもよい。
 - この API プログラムはモジュールごとに専用の設定を必要とする。
 - このモジュールはこの API とだけ通信する。
- MInt システム外（外部計算資源側）
 - NIMS 所外にあって、https で本 API へ到達可能なネットワーク設定の場所にあること。
 - 本 API と計算を行うためのポーリングプログラムのサンプルを python で用意した。
 - * ほとんどの場合このサンプルプログラムで事足りる。
 - 用意する計算機は Linux が望ましいが、サンプルを利用する場合 python が実行可能な PC なら何でもよい。
 - 必要な資材を取得、展開。
 - 資材をローカライズ（プログラム等を環境に合わせて編集）

2.2.5 用意されているサンプルワークフロー

下記イメージの動作検証用サンプルワークフローを用意してある。

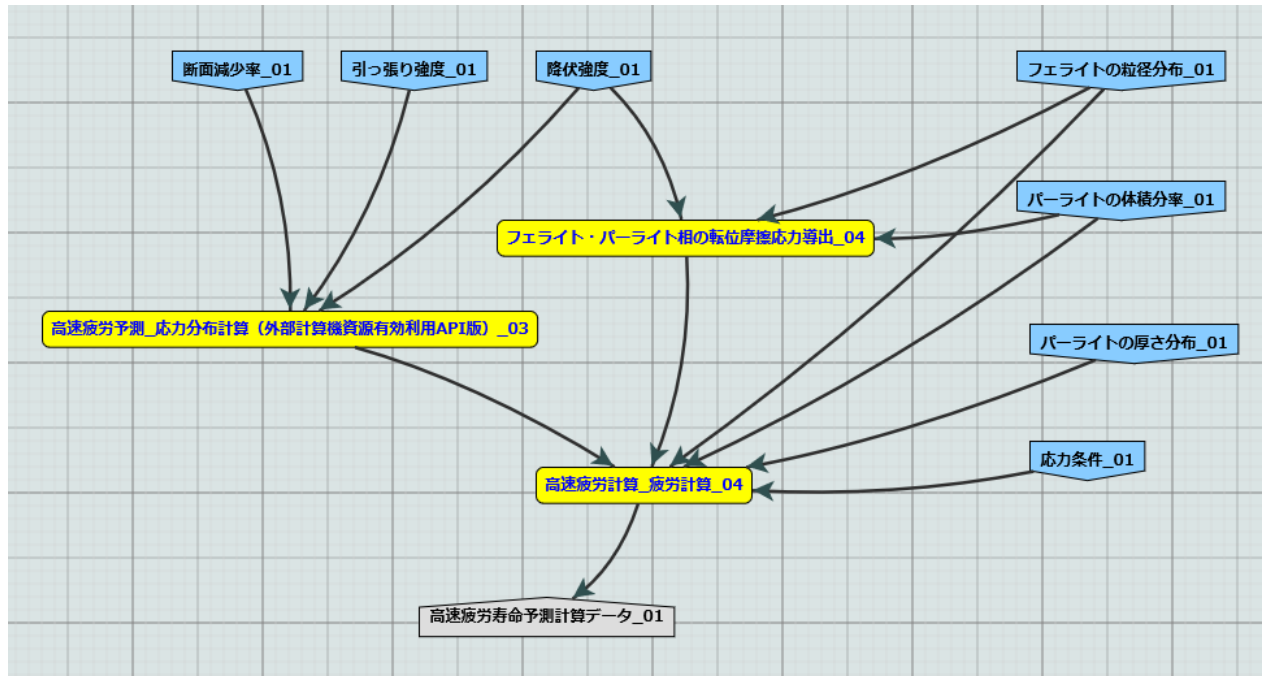


図 2.5 検証用ワークフロー

※赤枠の部分が外部計算機資源を利用するモジュールである。

2.2.6 MInt システムでのディレクトリ

MInt システム側のディレクトリ構造は以下のようになっている。

- 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。

2.2.7 外部計算資源でのディレクトリ

外部計算資源のディレクトリ構造は以下のようになっている。インストール方法については後述する。

- 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>
```


2.2.8 外部計算機で MInt システムから実行されるプログラム

ワークフローの該当モジュールから API 経由で外部計算機のコマンドが実行されるまでの流れを下記に示す。

図 2.6 ポーリング方式でのコマンドの流れ

本システムでは、MInt システムの API に設定したプログラムを外部計算機での実行に使用する。サンプルワークフローでは、「execute_remote-side_program_api.sh」となっている。外部計算機側ではインストール後にこのファイル（インストール直後は、execute_remote_program_api.sample.sh という名前）を必要に応じて編集して使用する。

2.2.9 MInt システムで送受信されるデータ

MInt システムで送受信されるデータは、MInt システム側の API と通信するモジュールの実行ファイルであらかじめ決め置く。API にはその情報によって外部計算資源とデータのやりとりをする。この情報に必要なファイルのみ設定することで、それ以外のファイルの存在を MInt システム側で感知できず、したがって不要なファイルのやりとりは発生せず、秘匿データなどの保護が可能となる。

第 3 章

使用方法

インストールおよびプログラムの準備など説明する。SSH 方式、API 方式のそれぞれの準備から実行までを記述する。

本システムの利用者は MInt システムのアカウントは既に発行済であるものとし、その手順は記載しない。また git コマンドなどの利用方法はシステム管理者などに問い合わせることとし、ここではそれらのインストール、詳細な使用方法は言及しない。

手順は以下のようになっている。

- 事前に決めておくこと
- 事前準備
- MInt システム側の準備
- 外部計算機側の準備
- ワークフローの準備

3.1 事前決定事項

事前に決定しておく項目は以下の通り。

- `misrc_remote_workflow` リポジトリの展開場所
 - クライアント側のプログラム実行場所として使用する。
 - 実行プログラム用のテンプレートなどが入っているのでこれを利用する。
- `misrc_distributed_computing_assist_api` リポジトリの展開
 - API 方式の場合に必要
 - `debug/remote-side/mi-system-reote.py` がポーリングプログラムで、これを実行しておく。

- 実行するプログラム
 - 外部計算資源側で実行するプログラム及び必要なパラメータの調査。
 - MInt システムから最初に呼び出されるスクリプトを決める
- SSH の場合
 - MInt 側からクライアント計算機への ssh ログインのための情報
 - 鍵暗号化方式によるパスワードなし、パスフレーズなし接続が望ましい。
- API の場合
 - API 方式の場合は不特定多数の利用者と API プログラムを共有するので、設定事項を MInt システム側に事前設定しておく。

3.1.1 API 方式の場合の設定事項

API 方式では、ssh とはまた違う認証情報が必要なため、それらを記述する。以下の情報は外部計算機側でポーリングプログラムを実行する際に必要である。

- API トークン
 - 本方式では MInt システムの API 認証システムを使用しているので、そのトークンが必要となる。NIMS 側に問い合わせ取得しておく。
- ホスト情報
 - MInt システム側で API 問い合わせに対する個別の識別を行うためにサイト情報（文字列として区別できれば何でもよい）が必要である。
- MInt システムの URL
 - MInt システムの URL（エンドポイントは不要）が必要である。NIMS 側に問い合わせしておく。

3.2 SSH 方式

SSH 方式での準備を決定事項にしたがって実施する。

3.2.1 外部計算機資源側

1. misrc_remote_workflow リポジトリを以下の手順で作成しておく。

```
$ git clone https://gitlab.mintsys.jp/midev/misrc_remote_workflow
$ cd misrc_remote_workflow
$ ls
README.md  documents  inventories  misrc_remote_workflow.json  modulesxml  sample_
↳data  scripts
$ cd scripts
$ ls
abacus                                execute_remote_command.sample.sh  ↳
↳kousoku_abacus_ssh.sh
create_inputdata.py                  input_data                          ↳
↳kousoku_abacus_ssh_version2.py
execute_remote-side_program_api.sample.sh  kousoku_abacus_api_version2.py  ↳
↳kousoku_abacus_ssh_version2.sh
execute_remote-side_program_ssh.sample.sh  kousoku_abacus_api_version2.sh  ↳
↳remote-side_scripts
execute_remote_command.sample.py          kousoku_abacus_http.py
```

2. 外部計算機資源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 システム側から外部計算機資源側へ ssh ログインして最初に実行されるプログラム名は「execute_remote-side_program_ssh.sh」です。このため「execute_remote-side_program_ssh.sample.sh」を「execute_remote-side_program_ssh.sh」にコピーするか、「execute_remote-side_program_ssh.sh」を独自に作成し、2. などの実行および必要な手順をスクリプト化しておきます。

3.2.2 MInt システム側

1. ワークフローを作成する場合に「misrc_remote_workflow/scripts/execute_remote_command.sample.sh」を必要な名称に変更し、内容を参考にして ssh 経由実行が可能なように編集し、ワークフローから実行させる。
2. 1. を実行可能な通常どおりのワークフローを作成する。作成方法に差は無い。

3.3 API 方式

3.3.1 外部計算機資源側

1. `misrc_distributed_computing_assist_api` リポジトリを以下の手順で作成しておく。

```
$ git clone https://gitlab.mintsys.jp/midev/misrc_distributed_computing_assist_api
$ 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. `my-system-remote.py` を実行しておく。

```
$ python mi-system-remote.py rme-u-tokyo https://nims.mintsys.jp <API token>
```

3.3.2 MInt システム側

1. `misrc_distributed_computing_assist_api` リポジトリを展開。
2. `mi_dicomapi.py` が本体であるが、まだ動作させてなければ、`mi_distributed_computing_assist.ini` に外部計算機資源側の設定を実施する。動作させていたら、設定の再読み込みを実施する。

```
$ python
>>> import requests
>>> session = requests.Session()
>>> ret = session.post("https://nims.mintsys.jp/reload-ini")
>>>
```

3. まだ動作していなかったら、動作させて待ち受け状態にしておく。

```
$ python mi_dicomapi.py
```

3.4 ワークフローについて

外部計算機資源利用を行うワークフローの作成の仕方を記述する。

3.4.1 共通事項

SSH 方式と API 方式の両方に共通する事項である。

- 予測モジュール
 - 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。
 - 外部計算機資源側で別途 Torque などのバッチジョブシステムに依存する。

3.4.2 SSH 方式

予測モジュールの実行プログラムから `misrc_remote_workflow/scripts/execute_remote_command.sample.sh` またはこのファイルを専用に別名コピー編集したものを必要なパラメータとともに実行するように構成する。

3.4.3 API 方式

予測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。

3.4.4 サンプル

`misrc_remote_workflow` リポジトリにある、`sample_data` ディレクトリにテストで使ったワークフロー実行用のサンプルファイルが用意されている。これを利用してワークフローおよび外部計算機側の動作の実行テストが可能である。

また、`misrc_remote_workflow/scripts` にこの時の予測モジュール実行プログラムがある。これを参考に別な予測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。

- `kousoku_abaqus_api_version2.py` : API 方式の予測モジュール実行スクリプト
- `kousoku_abaqus_ssh_version2.py` : SSH 方式の予測モジュール実行スクリプト

以上

図目次

2.1	SSH 実行のイメージ	3
2.2	動作検証用のワークフロー	5
2.3	SSH 接続経由によるコマンド実行の流れ	6
2.4	ポーリングシステムの流れ	10
2.5	検証用ワークフロー	12
2.6	ポーリング方式でのコマンドの流れ	14

表目次