ワークフロー言語でデータ解析ワークフローを記述する

WF再現性ベストプラクティス 2020年Q1版

大田達郎 (DBCLS)

希少疾患インフォマティクス2 @ DBCLS柏

2020-02-17

自己紹介

- 大田達郎, 特任助教 @ DBCLS
  - twitter, github: @inutano
- 遺伝学研究所 (静岡県三島市) 勤務
  - DDBJ のお隣さん
- 最近のお仕事
  - RDFによるNGSデータやサンプル情報の統合
  - Common Workflow Language を用いた解析パイプライン構築
  - 月例技術者交流会 Pitagora Meetup のホスト
    - 関連: Workflow Meetup

Agenda

1. 研究の再現性とデータ解析環境構築
2. コンテナ仮想化
    - メリット
    - エンジンの選択肢
    - コンテナ化ベストプラクティス
3. ワークフロー言語
    - 分類
    - 比較
    - 選択のポイント

1. 研究の再現性とデータ解析環境構築

こんなことありませんか

0. 色々のツールを自分のマシンにインストールしては試す
1. 本命のツールが固まったので [好きな言語] でワークフローを組む
2. 自分の使っているクラスタで分散処理できるようバッチ処理を組む
3. 大量のデータを流してはせっせと結果をこしらえる
4. 数年が経つ
5. 言語やフレームワークのバージョン、使っているマシンのOSが変わる
6. ある日突然もう一度同じ処理をする必要が発生する
7. 昔作ったワークフローを動かしてみるが動かない
8. 直しても直してもエラーが出る
9. 書き直した方が早いんじゃないかと思い始める頃には何日も経っている

あるいはこんなことがありませんか

0. 色々のツールを自分のマシンにインストールしては試す
1. 本命のツールが固まったので [好きな言語] でワークフローを組む
2. 自分の使っているクラスタで分散処理できるようバッチ処理を組む
3. 大量のデータを流してはせっせと結果をこしらえる
4. ある日 (同僚|共同研究者|論文を読んだ知らない誰か) からWFをくれと頼まれる
5. スクリプト群を渡して使い方を説明する
6. 少しすると「"....." というエラーが出て進まない」と問い合わせがくる
7. 自分のところではそんなエラーは出ない。多分これかな？と指示を出す
8. 「今度は　"....." というエラーが出て進まない」と問い合わせがくる
9. 以下 7-8 をうんざりするまで繰り返す、時間が奪われる、進捗が失われる

動かない理由

動いた時と何かが違う

- マシンスペック
- OS
- ジョブスケジューラなどのミドルウェア
- ライブラリのバージョン
- ソフトウェア (ツール) のバージョン
- ソフトウェアの実行時パラメータ
- ソフトウェア間の入出力の受け渡し
- 入力データ

対策

マシンスペック	ログに情報を残す
OS	コンテナを使ってOSから分離する
ミドルウェア	対応するWFエンジンを利用する
ライブラリのバージョン	コンテナを使って固定する
ソフトウェア (ツール) のバージョン	コンテナを使って固定する
ソフトウェアの実行時パラメータ	WF言語で記述する
ソフトウェア間の入出力の受け渡し	WF言語で記述する
入力データ	ログに情報を残す

今できる最善のこと

- ログを残す
- コンテナでソフトウェアをパッケージングする
- WF言語で入出力の依存関係を記述する

2. コンテナ仮想化

コンテナのメリット

- ツールインストールの手間から解放される
- "適切に運用すれば" ツールのバージョンを固定できる
- ツールごとにコンテナ化することでライブラリの衝突を防げる
  - さよならパッケージマネージャ
- VMと比較してイメージサイズが軽量、起動が速い
  - 可搬性が高い (気軽に別の環境でデプロイできる、クラウド向き)
  - VMよりも短いライフスパンで構築と運用ができる（使い捨て）

コンテナエンジンの選択肢

- Docker: https://www.docker.com/
  - 圧倒的シェア、デファクトだがDocker社のビジネスが不安
- Singularity: https://sylabs.io/docs/
  - HPC系では普及しつつあるがビジネスがスケールしないのではという不安
- uDocker: https://github.com/indigo-dc/udocker
  - 最後の選択肢として健在だが全体ではニッチでユーザ数が少ない不安

どのエンジンを使えばいいのか

- メジャーなエンジンは Docker image をサポートしている
- コンテナイメージは Docker で用意し、環境によってエンジンを使い分けるのが吉

```
Q1: admin がある？
    -[YES]-> Docker
    -[NO]-> Q2
Q2: admin に頼んだらソフトウェア入れてくれる？
    -[YES]-> Singularity
    -[NO]-> uDocker
```

コンテナ化ベストプラクティス

何はともあれ 公式ベストプラクティス を読むべし、その上で...

- コンテナ化の粒度: ツールごとにバラバラにする/全部入れる
- 自作スクリプトをコンテナ化するときには
- なるべく軽いイメージを
- コンテナのバージョン管理

コンテナ化の粒度

- VM的「全部入りコンテナ」は構築は楽だが維持が大変
  - 1つのツールのバージョン変更のために丸ごと変更が必要になる
  - ライブラリが衝突して env 系のツールを使わざるを得なくなる
    - pyenv とかコンテナの中で使い始めたら負け
  - 他の人が使うときに何が入っているのか把握しづらくデバッグが大変に
  - 1プロセス1コンテナが基本

自作スクリプト

- ローカルにあるスクリプトファイルを `COPY` するのはやめよう
  - Dockerfile だけあればビルドできる方がよりポータブルで管理が楽
    - 「Dockerfile と同じ場所にあるスクリプトを名前で指定」という仕様が危険
    - ここで意図しない挙動が起きるとデバッグ時に気付けない
  - GitHub でスクリプトをきちんとバージョン管理、タグを打っておく
  - GitHub で Release を作って `ADD` で取ってくるのがよい (by @suecharo)
- スクリプト間で利用するライブラリの衝突の心配がなければマイスクリプト全部入りユーティリティコンテナを作っても
  - 理想はバラバラのスクリプトよりは1つのコマンドラインツール、複数のサブコマンドとして実装したい

なるべく軽いイメージを

- サイズが大きいと取り回しが大変
  - テストや開発時にビルドを繰り返す場合に負担
- サイズの小さいベースイメージを使う e.g. debian, alpine
- 不要なライブラリは入れない、ビルド時にのみ必要なライブラリは構築後に削る
  - パッケージマネージャの機能を使う
    - alpine の `apk add --virtual` とか (参考)
  - マルチステージビルドを活用する (参考)

コンテナのバージョン管理

- 手動はしんどいので自動化しましょう
  - Dockerfile を書いて GitHub に置く
    - タグとリリースを適切に付与してバージョン管理する
  - Docker Hub もしくは Quay を使って Automated build でイメージを作る
    - イメージタグをGitHubのタグに連携してつける
      - `latest` はデバッグで死ぬので**絶対に**使ってはいけない

シェアウェアの闇

- 有償、ユーザ登録が必要、再配布が禁止されているツールは上の方法が使えない
  - Dockerfile だけを公開しておき、イメージは手元でビルドしてもらう
    - ツールはユーザにダウンロードしてもらい、コンテナは各自でビルド
    - ツールのバージョンがコントロールできないなど問題が多い
  - ビルドしたイメージをパーソナルコミュニケーションのレベルで配布する
    - 再配布禁止に引っかかるのでツールによっては違反
  - いっそコンテナ化しない
    - インストールの手間がそこまででもなければ、コンテナ化しなくていいかも
- 同等の機能を持つ他の選択肢があればこのようなツールは出来るだけ避ける
  - もしそういう開発をやっている人間がいたら使いにくいからやめろと説得する
  - MIT や Apache 2.0 などの緩いOSSを勧めよう

番外: それはコンテナにしなくてもいいのでは？

- 絶対に自分の環境でしか動かない（ハードコードしている）ツールやスクリプト
  - まずは違う環境で動くようにコーディングするところから...
- 事実上特定の環境でしか動かせないソフトウェア
  - 数十TBのDB/referenceを持っておかなければいけないなど
  - コンテナ化よりも、その環境を SaaS 化してAPIで叩けるようにするべきかも
- 入力データが変わるたびに試行錯誤をするようなプロセス
  - 統計解析や可視化などは、Notebook の方が向いてるかも

3. ワークフロー言語

ワークフロー言語 戦国時代

- Existing Workflow systems
  - 256 workflow systems 🙈

BOSC 2019 で一定のユーザが確認できたもの

BOSC: Bioinformatics Open Source Conference https://www.open-bio.org/events/bosc/

- Galaxy workflow specification (-2007)
- Workflow description language (WDL) (2012-)
- snakemake (2012-)
- nextflow (2013-)
- common workflow language (CWL) (2014-)

ワークフロー言語の分類

- シンタックスによる分類
  - Domain Specific Language (DSL) 型
  - データ型 (マークアップ型)
- 実行エンジンとの結合度合いによる分類
  - エンジンが選択可能
  - エンジンが選択不可 (密結合)
- 開発元による分類
  - 個人
  - 研究所単位でのバックアップ
  - コミュニティによる運営

Pros (and Cons)

- DSL vs Data
  - DSL: 一度覚えたら書きやすい, 柔軟性が高い
  - Data: パースしやすいので変換が容易, 文法がプレーン
- 実行エンジン
  - 固定: 開発リソースが分散しないので多機能、高機能に
  - 複数: 自分の環境やニーズに合ったエンジンを選択できる
- 開発元
  - 組織: 組織の信頼性や安定性が開発の品質に繋がりやすい
  - 個人: 身軽だが継続性にリスクがある
  - OSS: 組織と個人の中間, 品質や継続性はコミュニティのメンバーに依存する

前出の言語の比較

Name	Syntax	Engine	Dev
Galaxy	Data	Galaxy	JHU/OSS
WDL	DSL	Cromwell	Broad
snakemake	DSL	Snakemake	individual
nextflow	DSL	nextflow	individual
CWL	Data	multiple implementations	OSS

nextflow を選ぶ理由

- 柔軟な記法
  - コマンドラインを埋め込むような感覚で書ける
- パワフルなエンジン
  - 大概のジョブスケジューラに対応、クラウドにも対応
- コミュニティの勢い
  - nf.core などのリソースも多い
- 少人数の同じくらいのスキルのチームで生産性を上げるなら現状はnextflowがベスト
  - BroadのGATK WFがWDLなのでこれらを再利用できるWDLもGood

CWL を選ぶ理由 (あるいは敢えてnextflowで書かない理由)

- 思想的な好み
  - 「何を実行するか」 と「どう実行するか」を完全に分離できる
  - CWLは「何を」だけ書く、「どう」は良くも悪くもエンジン任せ
  - コアな部分をCWLで書いて残りはシェルやnextflowということもできる
- DSLの弱点は文法の学習コスト
  - 他人がちょっとだけ変更したいような場合にnextflowを勉強しろと言えるのか
  - CWL は少しだけマシ
- nextflow はこの先100年持つのか？
  - CWLなら少なくともパースは簡単、別の言語への移植は容易
- 全てにおいて完璧で永遠に使える言語は（当然ながら）ない
  - 将来別のトレンドが来ることを予期した上での選択
  - 言語間の交換フォーマットとしてのCWLの可能性

Common Workflow Language (CWL)

- BOSC生まれ、GitHub育ちの完全OSSプロジェクト
- 規約 (specification) と標準実装 (reference implementation)
- YAMLベースで input/command/output を記述する
- Bioinfo に限らない, 物理学分野などでも使っている人がいる
- 日本にも5人のコミッタがいる

CWL で何ができるか

- Command Line Tool のパッケージング
- Workflow のパッケージング
- 可視化
- エディタのサポート
- 複数のワークフローエンジンで実行が可能
  - cwltool, arvados, toil, CWL-airflow, REANA, Cromwell, CWLEXEC
  - その他 Galaxy, Taberna などでもサポートのための開発中

Command Line Tool の定義ファイル `fastq-dump.cwl`

```yaml
cwlVersion: v1.0
class: CommandLineTool
hints:
  DockerRequirement:
    dockerPull: quay.io/inutano/sra-toolkit:v2.9.0

baseCommand: [fastq-dump]
inputs:
  sraFiles:
    type: File[]
    inputBinding:
      position: 1
  split_files:
    type: boolean?
    default: true
    inputBinding:
      prefix: --split-files
outputs:
  fastqFiles:
    type: File[]
    outputBinding:
      glob: "*fastq*"
```

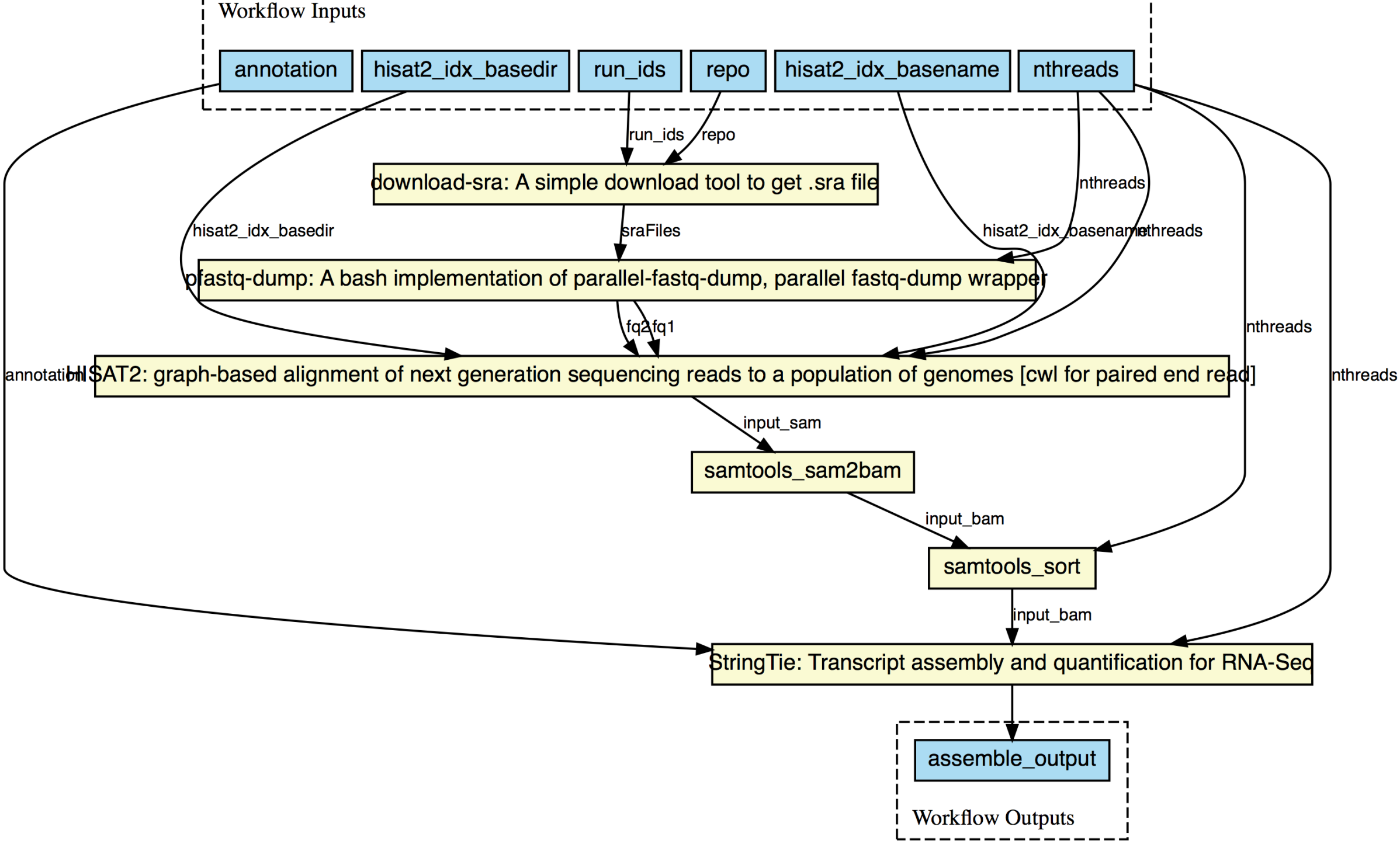
Workflow の定義ファイル `fastqc_wf.cwl`

```
cwlVersion: v1.0
class: Workflow

inputs:
  sra_files: File[]
outputs:
  fastqc_result:
    type: File[]
    outputSource: fastqc/fastqc_result
steps:
  pfastq_dump:
    run: pfastq-dump.cwl
    in:
      sraFiles: sra_files
    out:
      [fastqFiles]
  fastqc:
    run: fastqc.cwl
    in:
      seqfile: pfastq_dump/fastqFiles
    out:
      [fastqc_result]
```

view.commonwl.org

GitHub 上の CWL をレンダリング

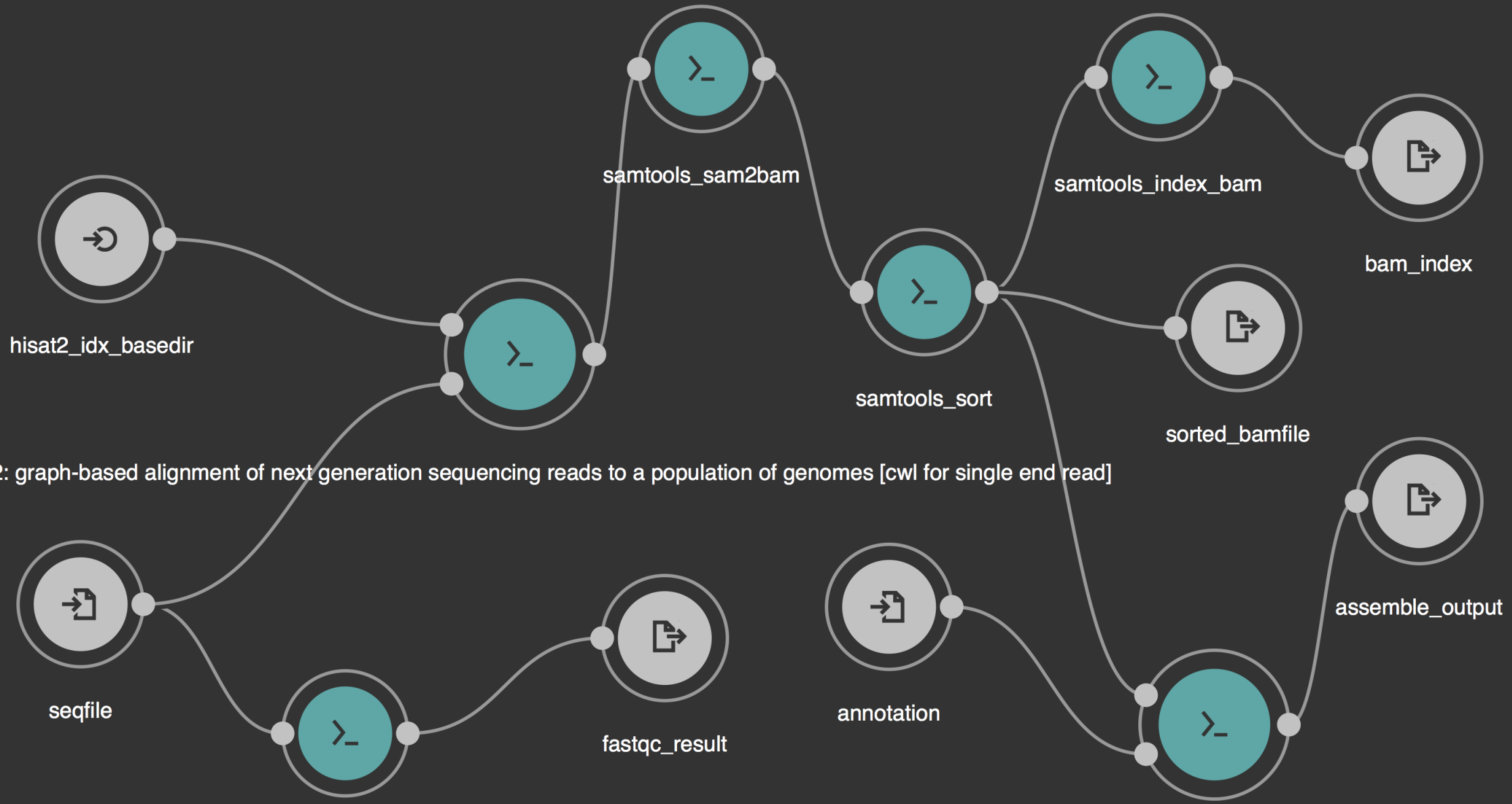

Workflow Inputs
annotation
hisat2_idx_basedir
run_ids
repo
hisat2_idx_basename
nthreads
run_ids
repo
download-sra: A simple download tool to get .sra file
nthreads
hisat2_idx_basedir
sraFiles
hisat2_idx_basename
nthreads
pfastq-dump: A bash implementation of parallel-fastq-dump, parallel fastq-dump wrapper
fq2fq1
nthreads
annotation
HISAT2: graph-based alignment of next generation sequencing reads to a population of genomes [cwl for paired end read]
nthreads
input_sam
samtools_sam2bam
input_bam
samtools_sort
input_bam
StringTie: Transcript assembly and quantification for RNA-Seq
assemble_output
Workflow Outputs

エディタサポート

- Rabix Composer
- Atom
- Vim
- Emacs
- VScode
- IntelliJ
- gedit
- Sublime Text

Rabix Composer

GUI でのCWLの編集と可視化をサポート


samtools_sam2bam
samtools_index_bam
bam_index
hisat2_idx_basedir
samtools_sort
sorted_bamfile
HISAT2: graph-based alignment of next generation sequencing reads to a population of genomes [cwl for single end read]
assemble_output
seqfile
annotation
fastqc_result
FastQC: A quality control tool for high throughput sequence data
StringTie: Transcript assembly and quantification for RNA-Seq

標準実装 cwltool で CWL workflow を実行する

```
$ cwltool fastqc_wf.cwl --sra_files SRR000001.sra
---
$ cat job_conf.yml
sra_files:
  - SRR000001.sra
  - SRR000002.sra
  - SRR000003.sra
$ cwltool fastqc_wf.cwl job_conf.yml
```

Implementations

Name	Platform
cwltool	Linux, OS X, Windows, local execution only
Arvados	AWS, GCP, Azure, Slurm
Toil	AWS, Azure, GCP, Grid Engine, OpenStack, Slurm, etc.
CWL-Airflow	Linux, OS X
REANA	Kubernetes, CERN OpenStack (OpenStack Magnum)
Cromwell	Google, HTCondor, Local, LSF, PBS/Torque, SGE, Slurm, TES
CWLEXEC	IBM Spectrum LSF 10.1.0.3+

詳細なログを残すことの重要性

- 残すべき情報
  - コンテナ、WF言語で分離したもの「以外」
    - e.g. マシン、OS、ミドルウェア、入力データ
- WFエンジンのログ情報を利用
  - `cwltool --debug`
  - ResearchObject というハードコアな手段もある
    - `cwltool --provenance`
- マシンスペックと消費リソースを記録するユーティリティツール `cwl-metrics`
  - github.com/inutano/cwl-metrics

Summary

- コンテナ化とワークフロー言語による記述で環境構築の再現性は高まる
- 条件や環境によって選択肢は異なる, 使わないという選択肢も
- 万能なものはない、コストとPros/Consを比較して合ったものを選ぶことが重要

番外編: Notebook

- **Notebookは最高**
- 前述の通り試行錯誤はWF言語ではなくnotebookでやるべき
  - R, python, Julia による統計解析と可視化は notebook 以外あり得ない
- notebook である程度固まってからWF言語に落とし込むというフローがベスト
- NGSの文脈で言うと以下のような役割分担
  - 配列データ - (WF言語) -> vcfなどのテーブルデータ - (notebook) -> 統計値や図
- Notebook もコンテナで: hub.docker.com/u/jupyter

追記: 当日の質疑の一部

- 再配布不可のツールの現状のコンテナ化ベストプラクティスは
  - ツールは各自でインストールしてもらい、インストールされているツールが正しいバージョンかどうかを確かめるステップをWFに入れるのがよい
- blastのdbファイルなどはどのように扱うのがよいか？
  - コンテナには含めずオブジェクトストレージなどに置いておきそれをワークフローの入力として取りに行くのがよいが、サイズやデータの消費期限次第。
- DSL vs Data, 入出力のハンドリングが柔軟なのは？
  - 大量の入出力の処理などはDSLの方が得意。Dataでも書けるが冗長になりがち
- KNIMEのような統合環境はないか？
  - 強いて言えば Galaxy か。Rabix Composer も近い
  - 配列解析はツールのバリエーションが多い、ライフサイクルが比較的短いため CLI が好まれがちだと思われる