Comparative Analysis of Genomic Sequencing Workflow Management Systems

L. S. Mainzer[5,7], A. Ahmed[11,12], S. Baheti[3], M. A. Bockol[4], P. Burra[7], R. Campbell[9], T. M. Drucker[4], F. M. Fadlelmola[11], S. N. Hart[2], J. Heldenbrand[7], M. Hernaez[5], M. E. Hudson[5,6], R. K. Iyer[8], M. T. Kalmbach[4], K. I. Kendig[7], E. W. Klee[2], C. Liu[7], N. R. Mattson[4], O. Milenkovic[5,8], C. A. Ross[4], S. Sinha[5,9], R. Venkatakrishnan[7], M. R. Weber[6,7], E. D. Wieben[1], M. Wiepert[4], D. E. Wildman[5,10]

Introduction

As genomic sequencing becomes common in academic, clinical and commercial settings, workflow management systems are being developed to manage the large volume of data and the complexity of analyses. Here, we compare three popular workflow management systems for large-scale genomic sequencing analyses: Cromwell/WDL [1], Nextflow [2], and Swift/T [3,4], on the example of the GATK Best Practices for variant calling. Though all three serve the same general purpose, their inbuilt functionalities lend them to different usages. We present a qualitative comparison of the three and a delineation of key comparison metrics, to aid users in selecting the best workflow management system for their high-performance computational needs.

Supporting code can be found at:
https://github.com/ncsa/MayomicsVC
https://github.com/ncsa/Genomics_MGC_VariantCalling_Nextflow
https://github.com/ncsa/Swift-T-Variant-Calling

Comparison aspects

User interface: the means by which the user interacts with the software. Possible options include command-line interface (CLI), read-eval-print-loop (REPL), and integrated development environment (IDE) .

Containerization support: methods to virtualize an OS to run on a host without separate virtual machines.

Checkpointing: ability to save workflow state periodically, allowing for rerun from it upon failure.

Caching: ability to store frequently used data in memory to reduce data retrieval time.

Portability: usability of software in a variety of different operating environments.

Distributed execution engine: makes the computer cluster look like a single machine. Circumvents the use of task scheduler and resource manager.

Modularity: program implemented as a library of modules, allowing for design flexibility and maintainability.

Error handling strategy: functionalities to address and resolve errors that arise during program execution

Parallelization: methods to distribute data among multiple compute nodes, allowing many instances of the same function to run at the same time.

SPARK support: GATK is moving from being deployed on the grid, to cloud-based analytics computation using MapReduce in SPARK. Thus SPARK support will be required of future variant calling workflows.

Workflow Management Systems

Cromwell/WDL: intended to serve as a bridge between complex domain-specific languages and simple scripts. WDL=Workflow Definition Language; Cromwell is the execution engine for WDL workflows. Emphasis is placed on user-friendly coding suitable for non-programmers.

Nextflow: based on common programming languages Groovy and Ruby. It is incredibly user-friendly with inbuilt functionalities for error handling and metadata compilation.

Swift/T: intended for computation on a massive scale. Swift is a powerful C-like language. Turbine is the execution engine for Swift workflows. Though Swift/T contains many unique features like load-balancing, the programming is not intuitive and may be overwhelming to novice programmers.

Swift/T logging, user error notifications:

```
file alignBams[ ] =
   alignRun(sampleLines, variables, failureLog) =>
   logging(variables[“TMPDIR”], timingLog, “alignlogs”);

assert(
         size(alignBams) != 0,
         “FAILURE: The aligned bam array was empty:
          none of the samples finished properly”
);
```

Nextflow error handling commands:

- **terminate:** terminates execution as soon as error emerges, kills pending processes (default condition)
- **finish:** orderly shutdown of workflow; waits for completion of any submitted processes
- **ignore:** ignores execution errors from processes, sends message to user that event has occurred
- **retry:** re-submit/re-execute process that returned an error condition. Can specify maxErrors and maxRetries (these are disabled as a default)

Cromwell/WDL error handling:

```
runtime {
 continueOnReturnCode: {true|false|array-of-integers}
 failOnStderr: {true|false}
}
```

Swift/T implicit parallelization:

Statements are evaluated in parallel unless prohibited by a data dependency or resource constraints, without the developer needing to explicitly code parallelism or synchronization. Swift/T will automatically wait on a process to finish if the next step depends on its output. When a stage must wait on another, yet a direct data dependency does not exist, the wait can be forced:

```
mkdir(LogDir) =>
mkdir(AlignDir) =>
void mkdirSignal = mkdir(tmpLogDir);

wait (mkdirSignal) {
  alignedsam = alignReads(vars, sampleName, reads, rgheader);
}
```

Cromwell/WDL scatter task for parallel read mapping:

```
import “BWAMemSamtoolView.wdl” as BWASAMTOOLVIEW

Workflow CallReadMappingTask {
  # define inputs

  scatter(sample in inputsamples) {
     call BWASAMTOOLSORT.ReadMappingTask {
       input : sampleName = sample[0]
}}}
```

Nextflow data-level parallelization via “channels”:

```
inputFiles = Channel
    .fromPath(params.inputFiles)
    .splitText()
    .splitCsv(sep: "\t")
```

Swift/T modularity via “workers”:

Individual Swift functions are chained together by the primary workflow script, as subroutines in most computer languages.

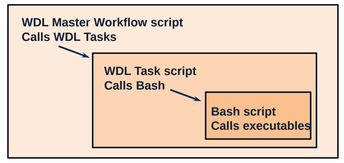
```
@dispatch=WORKER
app (file output, file outLog) bwa_mem (string bwaexe, string read1,
  string read2, string INDEX, string bwamemparams[], int PBSCORES,
  string rgheader)
{
  bwaexe "mem" "-M" bwamemparams "-t" PBSCORES "-R" rgheader
    INDEX read1 read2 @stdout=output @stderr=outLog;
}

import bioapps.align_dedup;
```

Comparison Aspect	Cromwell/ WDL	Nextflow	Swift/T
User interface	CLI	CLI, REPL, IDE	CLI
Containerization support	Docker	Docker, Singularity	None
Checkpointing & caching	Yes	Yes	No
Portability	LSF, HTCondor, Google JES	LSF, NQSII, HTCondor, Kubernetes, Ignite, DNAnexus	Cray aprun
Distributed execution engine	Spark	Apache Ignite/ MPI	MPI-based
Modularity	Yes	Yes	Yes
Retry on error	No	Yes	Yes, if failed QC
Error handling strategy	Continue	Continue, retry, terminate, organized finish	Continue upon failing quality control
User notifications	Easy Bash addition	Built-in	Easily implemented
Parallelization	Scatter-gather	Implicit within channels	Implicit & complete
Documentation & community	Extensive, supported by Broad Institute	Extensive, with online forums	Extensive documentation & tutorials
Ease of use	Easy, but requires Bash knowledge	Easy	Difficult, but with many unique features
Tracing & visualization	No	Yes	Some
SPARK support	Yes	No	?

Modularity in WDL and Nextflow:

✓ Bash script for each analysis step
✓ WDL or nf task for each analysis step, calls the Bash script
✓ Unit workflow to test each WDL task or nf script
✓ Workflow of tasks for the entire Design Block

WDL “Task” == Nextflow “Process”



Nextflow has built-in functionality to create execution, trace, and timeline reports, and vidualize DAGs. Execution reports consist of a workflow summary, a resource usage graph, and a list of tasks alongside their respective runtime metadata. The DAG visualization will create a direct acyclic graph of the workflow, with processes illustrated as nodes.

Tasks in Nextflow:

```
outputDir = file(params.folder)

process alignReads {
//perform bwa mem alignment on sample reads

  input:
    first_read+fastq=file(params.LeftReads)
    second_read_fastq=file(params.RightReads)
    fasta_ref = file(params.fasta_ref)
    // Etc …

  output:
    file 'set5Aligned.bam' into alignedFiles

  script:
    template 'bwaMemSamtools.sh'
}
alignedFiles.subscribe{it.copyTo(outputDir)}
```

Tasks in WDL:

```
task alignmentTask {
  File Ref                # Reference Genome
  File InputRead1         # Input Read File
  String InputRead2       # Input Read File
  # Etc …

  command {
    /bin/bash ${AlignmentScript}
    -L ${SentieonLicense} -P ${PairedEnd}
    -g ${Group} -I ${InputRead1} -r ${InputRead2}
    -s ${SampleName} -p ${Platform} -G ${Ref}
    -S ${Sentieon} -t ${Threads} ${DebugMode}
  }

  output {
    File AlignedSortedBam ="${SampleName}.aligned.sorted.bam"
    File AlignedSortedBamIdx = "${SampleName}.aligned.sorted.bam.bai"
}}
```

Master wflow in Nextflow:

```
outputFile = file(params.folder)
process bwaMem {
    """
    nextflow run ${params.bwaMem}
    """
}
process Novosort {
    """
    nextflow run ${params.run_novosort}
    """
}
```

Master workflow in WDL:

```
import “some/path/alignment.wdl” as ALIGNMENT
import “ some/path /dedup.wdl” as DEDUP

workflow CallBlock1Tasks {
  call ALIGNMENT.alignmentTask as align

  call DEDUP.dedupTask as dedup {
    input:
      InputAlignedSortedBam  = align.AlignedSortedBam,
      InputAlignedSortedBamIdx = align.AlignedSortedBamidx
}}
```

Conclusions

Swift/T is a powerful language that gives utmost flexibility and freedom in developing workflows. With its ability to rapidly perform thousands of small processes, it is ideal for exascale analyses. However, the learning curve may be steep and debugging difficult.

Nextflow is intuitive, mature and provides all features necessary for robust code development and maintenance for the Clinic: transparent inclusion of subprocesses, progress tracking, loggery. Unfortunately, it does not yet provide an option for deployment on Spark.

Cromwell/WDL is extremely similar to Nextflow in spirit, syntax and structure, but lacks many useful features and can be verbose. Using JSON as config files adds chores and complexity. Built-in Spark functionality will enable seamless deployment of GATK4.

Acknowledgements

This work was a product of the Mayo Clinic and Illinois Strategic Alliance for Technology-Based Healthcare. Major funding was provided by the Mayo Clinic Center for Individualized Medicine and the Todd and Karen Wanek Program for Hypoplastic Left Heart Syndrome. We thank the Interdisciplinary Health Sciences Institute, UIUC Institute for Genomic Biology and the National Center for Supercomputing Applications for their generous support and access to resources. We particularly acknowledge the support of Keith Stewart, M.B., Ch.B., Mayo Clinic/Illinois Grand Challenge Sponsor and Director of the Mayo Clinic Center for Individualized Medicine. Special gratitude to Amy Weckle for managing the project. Finally we are grateful for the support of H3ABioNet, funded by the National Institutes of Health Common Fund under grant number U41HG006941.

ORGANIZATIONS
Mayo Clinic, Rochester, Minnesota, USA
1. Department of Biochemistry and Molecular Biology
2. Department of Health Sciences Research
3. Department of Research Services
4. Department of IT Executive Administration

University of Illinois at Urbana-Champaign, Urbana, IL, USA
5. Carl R. Woese Institute for Genomic Biology
6. Department of Crop Sciences
7. National Center for Supercomputing Applications
8. Department of Electrical and Computer Engineering
9. Department of Computer Science
10. Department of Molecular and Integrative Physiology

University of Khartoum, Khartoum
11. Center for Bioinformatics and Systems Biology, Faculty of Science
12. Department of Electrical and Electronic Engineering, Faculty of Engineering

REFERENCES

1. WDL User Guide. Broad Institute. https://software.broadinstitute.org/wdl/documentation/
2. Di Tommaso P., Chatzou M., Floden EW. et al. Nextflow enables reproducible computational workflows. *Nature Biotechnology* **35**, 316–319(2017). doi:10.1038/nbt.3820
3. Wilde M, Hategan M, Wozniak JM, Clifford B, Katz DS, Foster I. Swift: A language for distributed parallel scripting. Parallel Computing. 2011;37(9):633–652. doi:10.1016/j.parco.2011.05.005.
4. Wozniak JM, Armstrong T, Maheshwari K, Lusk E, Katz D, Wilde M, et al. Turbine: A distributed-memory dataflow engine for extreme-scale many-task applications. Proceedings of 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies (SWEET'12); 2012.Available from: http://dl.acm.org/citation.cfm?id=2443421.