

Top-Down Performance Analysis Methodology for Workflows

Ronny Tschüter, Christian Herold, Bill Williams, Maximilian Knespel, Matthias Weber

Workflows Matter

Common use cases today are not limited to single applications

- Pre- and post-processing
- Heterogeneous applications in multiple jobs
- Multi-phase calculations as separate jobs

The obvious job is not always the problem job

A single job is not always the problem

I/O captures inter job dependencies and communication

- I/O is to workflows what MPI is to individual parallel jobs

I/O: More than POSIX

New Score-P feature: I/O recording (POSIX, ISO C, HDF5, NetCDF, MPI I/O)

Instrumenting high-level I/O interfaces allows better attribution of costs

- High level interfaces may be asynchronous, distributed, filtered
- Small high-level operations may produce large actual changes, or vice versa

Approach

Display overall summary of the behavior of a workflow

- Distribution of time among constituent jobs
- Breakdown of workflow into I/O, communication, and computation components
- Dependencies among jobs

Display summary of each job's behavior

- Distribution of time among job steps
- Breakdown of each job into I/O, communication, and computation components

Display each job step (single application run)'s behavior

- Breakdown into I/O, communication, and computation components
- Access to full trace data in Vampir

Implementation

Convert OTF2 traces to high-level summary profiles

- Categorize each function as I/O, communication, or computation
- Hierarchical view of I/O handles accessed
- Summary of job properties

Generate summary of entire workflow from SLURM accounting database

- Jobs and steps involved
- Submission parameters and dependencies
- Submission, start, and end times

Visualize results

- Identify I/O dependencies
- Build timeline and dependency information
- Link profile view of each job step to detailed trace view in Vampir

Identifying I/O dependencies

Jobs reading/writing the same I/O handle

In particular, may-read after may-write

Of particular interest: identifying independent steps in the workflow

These steps can potentially be run simultaneously if the allocation is large enough

Problems: false sharing, filtering for relevance

- Files may be opened with overly permissive permissions
- /dev, /proc, /sys etc. may create false dependencies
- Scaling problems: one sample run had 28k files, of which ~500 were not in the above directories

Example Workflow: GROMACS

Well-known molecular dynamics software

Typical workflow:

- Set up simulation environment
- Add solvent medium
- Generate initial molecular model (e.g. of a protein)
- Energy minimisation
- Initial equilibration
- Actual molecular dynamics computation

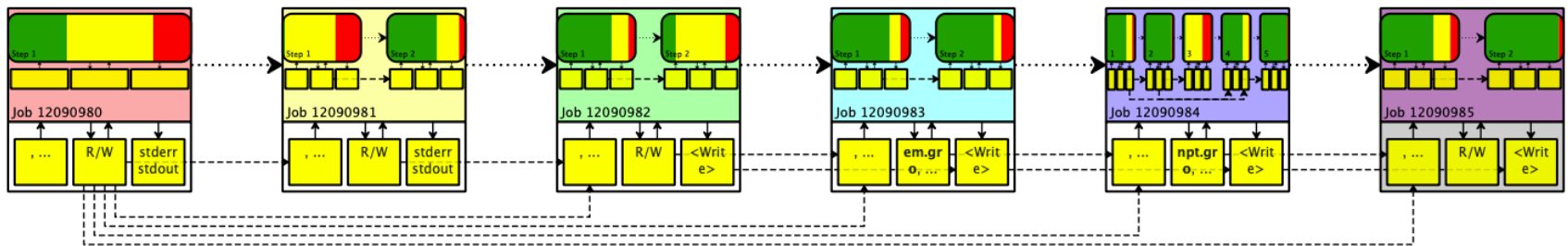
Steps communicate with each other via filesystem

Dependencies are implicit

Not preconfigured to use any off-the-shelf workflow managers

GROMACS in more depth

Dependency Graph



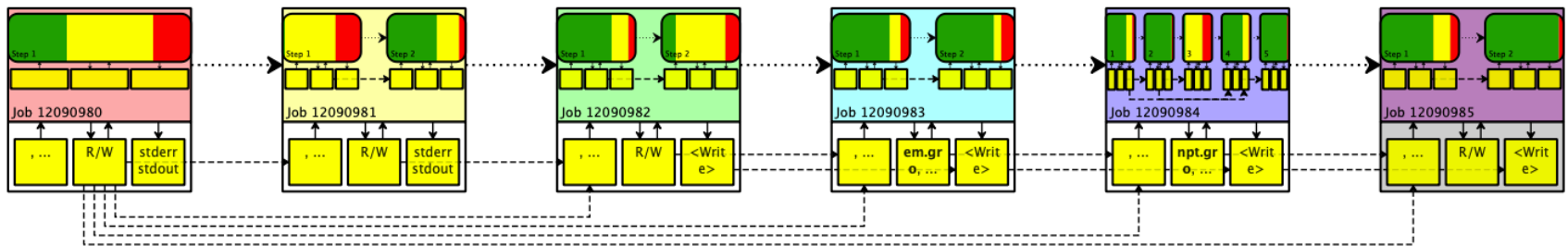
Legend



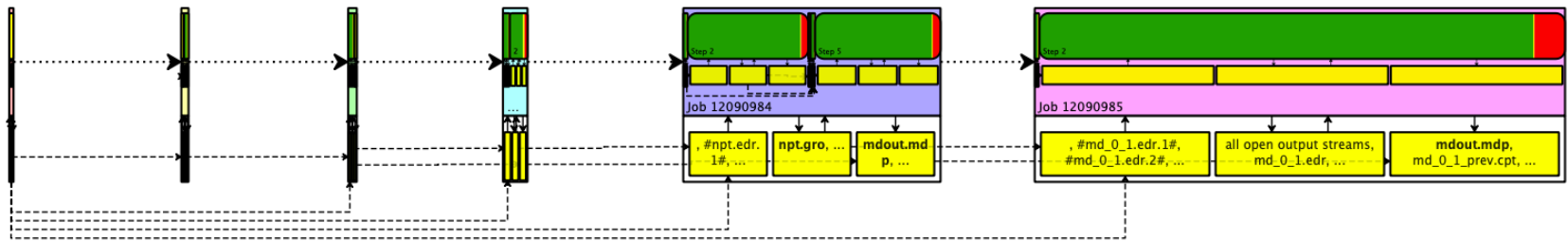
Dependency

GROMACS in more depth

Dependency Graph



Timeline

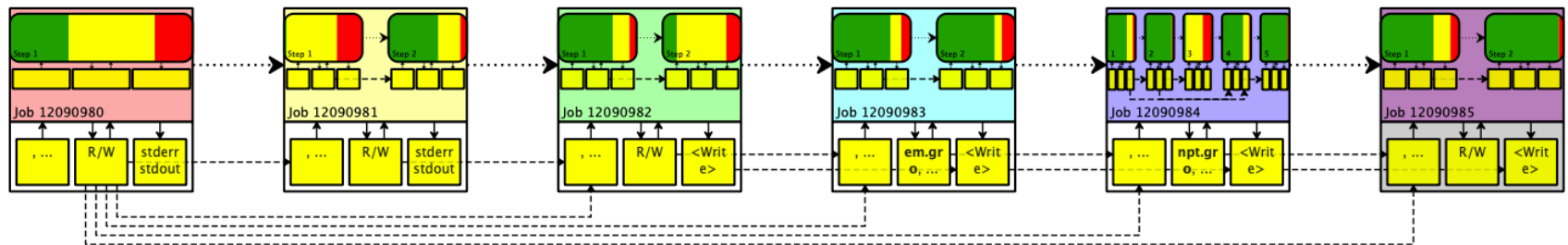


Legend

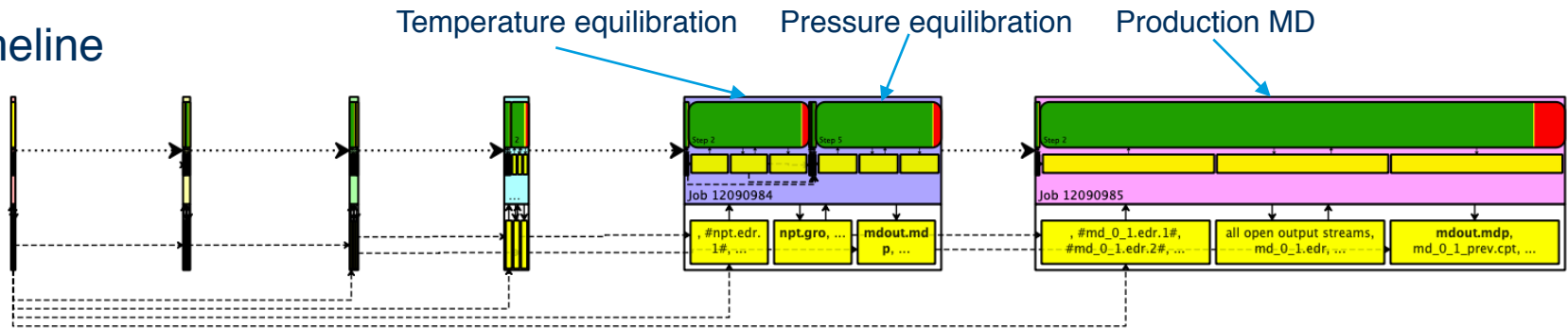


GROMACS in more depth

Dependency Graph



Timeline



Legend



GROMACS: finding load imbalance



Top: dynamic load balancing disabled, FFT code (yellow) imbalanced across ranks
Bottom: dynamic load balancing enabled, FFT code remains balanced

Result: reduces MPI share from ~10% to ~5% in the production MD step

Integration with Workflow Managers: Cromwell and GATK

Many workflows use something other than pure SLURM and shell scripts to manage dependencies

Especially true when these workflows are more complex DAGs

Sample workflow manager: Cromwell

- Supports a wide variety of back ends, including but not limited to SLURM
- Supports a flexible enough specification to allow us to insert measurement hooks
- Has off-the-shelf example workflows that provide real-world non-linear dependencies

Cromwell hooks

Backend configuration

```
slurm {
  actor-factory =
"cromwell.backend.impl.sfs.config.ConfigBackendLifecycleActorFactory"
  config {

    ""
    String scorep = ""
    ""
    script-epilogue="/usr/bin/env bash /home/wwilliam/wdl-testing/slurm-epilog.sh"

    submit = ""
      sbatch -J ${job_name} -D ${cwd} -o ${out} -e ${err} -t ${runtime_minutes} -
p ${queue} --export=ALL \
      ${"-n " + cpus} \
      --mem-per-cpu=${requested_memory_mb_per_core} \
      --wrap "/usr/bin/env bash ${scorep} /usr/bin/env bash ${script}"
    ""
    kill = "scancel ${job_id}"
    check-alive = "squeue -j ${job_id}"
    job-id-regex = "Submitted batch job (\\d+).*"
  }
}
```

Cromwell hooks

Score-P wrapper

```
#!/bin/bash

set -e
echo "Job $SLURM_JOB_ID"
export BASE_DIR=$PWD
export OUTPUT_DIR=$PWD/experiments
export SCOREP_EXPERIMENT_DIRECTORY=$OUTPUT_DIR/$SLURM_JOB_ID/$SLURM_JOB_ID
export SCOREP_ENABLE_TRACING=true
export SCOREP_ENABLE_PROFILING=false
export SCOREP_TOTAL_MEMORY=3700MB
echo "Job $SLURM_JOB_ID"
export SCOREP_FILTERING_FILE=/home/william/SimpleVariantDiscovery/test.filter

install_scorep_dir=/home/william/scorep-install-java
bin_scorep_dir=$install_scorep_dir/bin
lib_scorep_dir=$install_scorep_dir/lib
profiler=/home/william/workflow-analysis/vendor/otf2_cli_profile/build/otf-profiler
export LD_LIBRARY_PATH=$lib_scorep_dir:$LD_LIBRARY_PATH
mkdir -p $OUTPUT_DIR/$SLURM_JOB_ID
$@
pushd $SCOREP_EXPERIMENT_DIRECTORY

if [ "$SCOREP_ENABLE_TRACING" = "true" ]
then
    $profiler -i $SCOREP_EXPERIMENT_DIRECTORY/traces.otf2 --json -o $SCOREP_EXPERIMENT_DIRECTORY/result
fi
popd
```

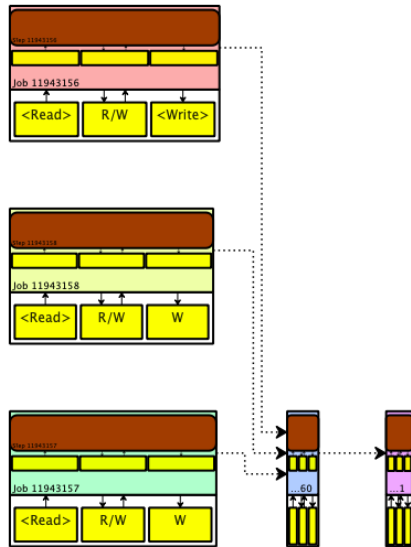
Cromwell hooks

Epilog script

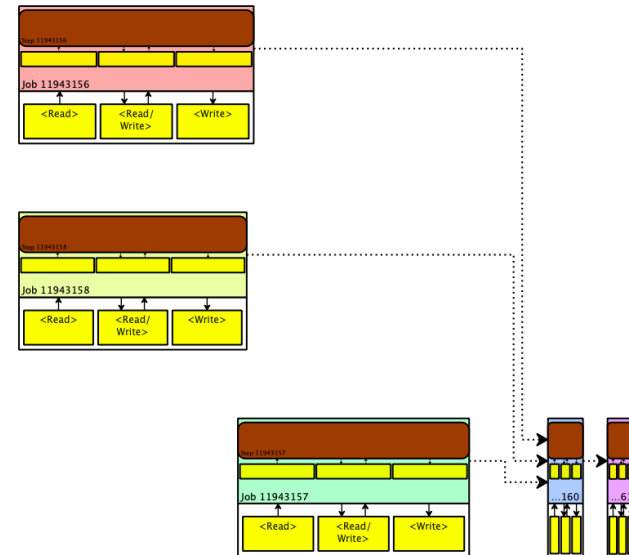
```
#!/bin/bash  
  
module load Python/3.6.6-foss-2019a  
  
cd ../experiments  
SLURM_JOB_ID=`ls`  
python /home/william/workflow-analysis/vendor/JobLog/joblog.py $SLURM_JOB_ID  
$SLURM_JOB_ID
```

GATK Sample Workflow: Joint Calling Genotypes

Duration scaled dependency graph



Timeline



Input data: one set of files per genetic sample

Process each input appropriately and independently

Merge and process results

Implementation Issues

GATK is written in Java; Score-P support is still experimental for Java tracing

As with many real-world workflows, GATK doesn't use MPI

Common parallelisation approach: more jobs in the workflow

Will need to evaluate GUI scalability

- Number of jobs
- Number of files

Future Work

Formalized metrics of workflow quality

Visualization of these quality metrics

Automate recommendations for workflow tuning

Evaluating overall workflow performance

Multiple criteria possible depending on site policy and goals:

- Goodness of fit (how efficiently does the workflow use its allocated resources?) E.g. typical shared environments with many smaller workflows sharing a large machine
- End-to-end time (if we're maximally generous with resources, how fast can we make this workflow?) E.g. weather forecasting
- Energy efficiency

Visualizing Workflow Quality

Goodness of fit: visualise allocated vs. used resources

- Scale box height
- Do we need log scaling options for time and size?
- How to distinguish large number of small jobs from single large jobs space-efficiently?
- Do we care how much of each partition we're using?

End-to-end time: identify and visualise workflow's critical path

- Based on current I/O pattern? Based on best case from trace analysis?

Important modelling question: how does average queue time scale with:

- Size of allocation
- Duration of allocation

Steps towards optimisation

Define scope of valid optimisations

- Restructuring workflow (script-level dependency changes)
- Changing filesystems for important steps
- Changing scaling of various steps
- Optimizing a single application
- Restructuring inter-app communication patterns
- Redesigning entire workflow (wholesale redesign of poorly performing apps, creating new apps/merging existing ones, redefining intermediate data requirements)

Conclusion

Implemented:

- Measurement infrastructure for HPC workflow tracing
- Initial visualiser for workflow performance
- Job dependency analysis based on SLURM information, I/O, and timing

Tested on:

- GROMACS with custom run scripts
- GATK JCG workflow under Cromwell

Visualizer available in upcoming Vampir release

Trace-to-profile tool and job log tool available on GitHub

Questions?

Contact: william.williams@mailbox.tu-dresden.de