SEA JUBE Benchmarking Workshop

DR. MAX HOLICKI | JÜLICH SUPERCOMPUTING CENTRE

JÜLICH
Forschungszentrum

THE PEOPLE BEHIND JSC SEA BENCHMARKING?

- Sebastian Lührs (Developer of JUBE)

- Andreas Smolenko (DEEP-SEA T1.2 Lead, IO-SEA)
- Max Holicki (DEEP-SEA, IO-SEA T1.2 Lead, Red-SEA)
- Jan-Oliver Mirus (DEEP-SEA, IO-SEA)
- Yannik Müller (DEEP-, IO- & RED-SEA)

- Tom Ridley
- Filipe Guimarães

SCHEDULE

08:50 Video Conference Opens

09:00 Introduction

09:15 Introduction to JUBE by Sebastian Lührs

10:15 Workshop Part 1 (installation + Hands on web tutorial)

12:00 Lunch

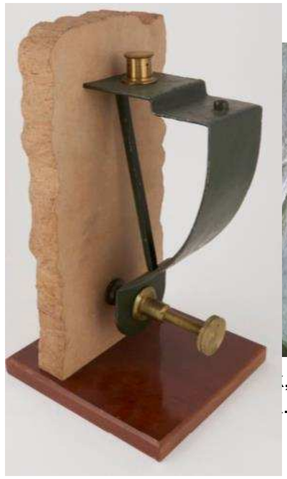
13:00 SEA Benchmarking + Discussion

14:00 Workshop Part 2 (application integration)

16:00 Wrap-Up

16:30 Video Conference Closes

WHAT IS A BENCHMARK?



https://collection.sciencemuseumgroup.org.uk/objects/co52873/angle-iron-to-support-levelling-staff-at-bench-mar-bracket-structural-element-bench-mark

, CC BY-SA 4.0,
.org/w/index.php?curid=57

Origin from surveying (1884): Bench + Mark

1. Bench: An angle iron (used to support a leveling staff)
2. Mark: A marking identifying a location

Combined these define a point of reference or a point of comparison.

Nowadays benchmarking no longer refers to the process of establishing a benchmark, but to the comparing to a benchmark.

https://www.zenithsurvey.co.uk/uploaded/thumbnails/db_file_img_4188_1024x1024.jpg

WHAT IS A BENCHMARK IN THE COMPUTER SCIENCES?

Benchmarking in the Computer Sciences is the act of comparing the execution of a program based on metrics, like runtime.

Often this is done to ensure that there are no regressions in performance, which makes it akin to regression testing, but not 100% the same.

WHAT DO WE WANT TO ACHIEVE IN IO-SEA?

Improve IO

WHAT DOES THIS MEAN FOR BENCHMARKING IN IO-SEA?

Network Bandwidth

IO-Time

IO-Latency

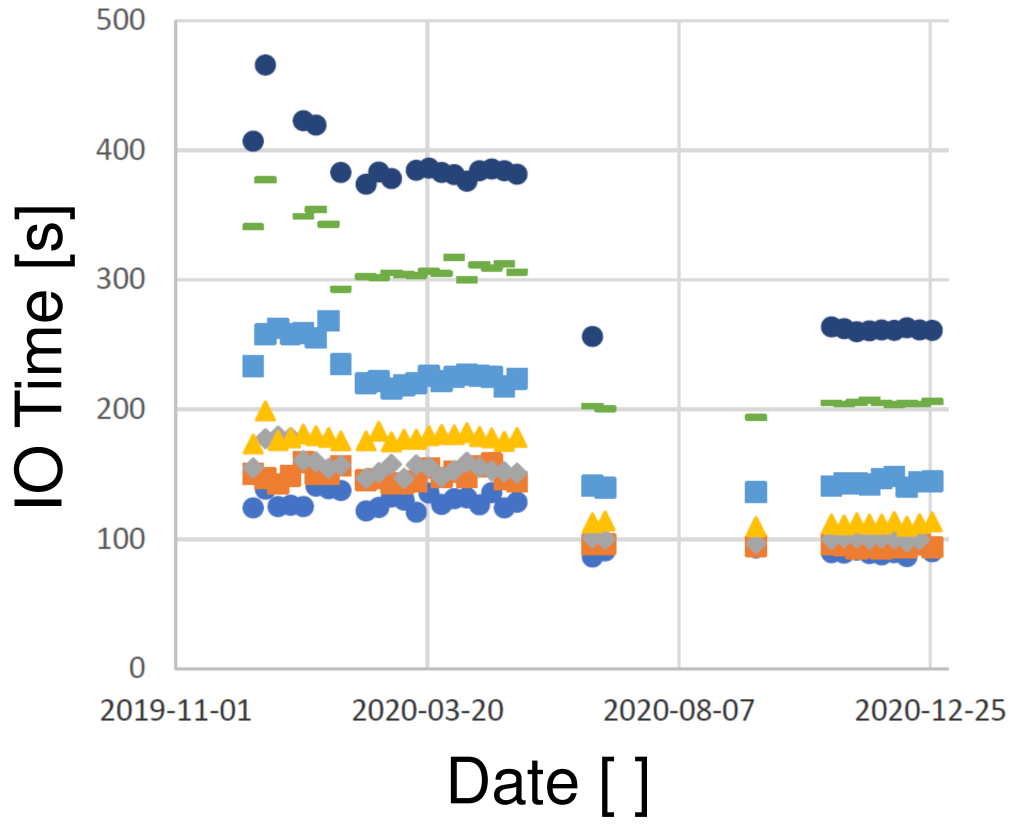
Benchmark IO

IO-Bandwidth

Runtime

Network Latency

WHAT IS OUR END GOAL?



This graph is purely illustrative!

WHY BENCHMARK IN SEA PROJECTS?

1. **Accountability**: At the end of the day we need to demonstrate to our sponsors that the money invested into us has led to something and that this is also an agreed possible outcome. *Trust is good, control is better!*
2. **Staying-on-Track/Regression-Testing**: Consistent benchmarking will allow us to stay on track and give feedback to developers if their codes/systems are improving. *In a sense we do regression testing for you.*

JUBE TUTORIAL

- Online at: https://apps.fz-juelich.de/jsc/jube/jube2/docu/tutorial.html

- On JSC systems:
  1. Load JUBE: *module load JUBE/2.4.1*
  2. Copy Examples: *cp $EBROOTJUBE/examples <PATH>*

- From TAR file:
  1. Decompress archive
  2. Navigate to examples folder
- Reservation: workshop_hpc on JUSUF

SEA JUBE Benchmarking

DR. MAX HOLICKI | JÜLICH SUPERCOMPUTING CENTRE

THE PLAN I

- Benchmark systems using use cases and synthetic benchmarks.
- Why synthetic benchmarks?
  - We need to understand performance change in use cases.
    - Are they due to use-case–code changes?
    - Are they due to software changes?
    - Are they due to hardware changes?

THE PLAN II

- Weekly or Biweekly Schedule.
- Automated Benchmarking via bots, cron jobs and JUBE.
- Aggregate results on GitLab.
    - One repository per software, one branch for each system.
- Archive important benchmark results on local machines.

- **We need you to keep a log of system/software changes!**

THE CRON JOB

(Bi-)Weekly Schedule

JUBE

1. Build
2. Benchmark
3. Extract Results

GitLab

1. Checkout Branch
2. Push Results

Tar

1. Compress Benchmark Results

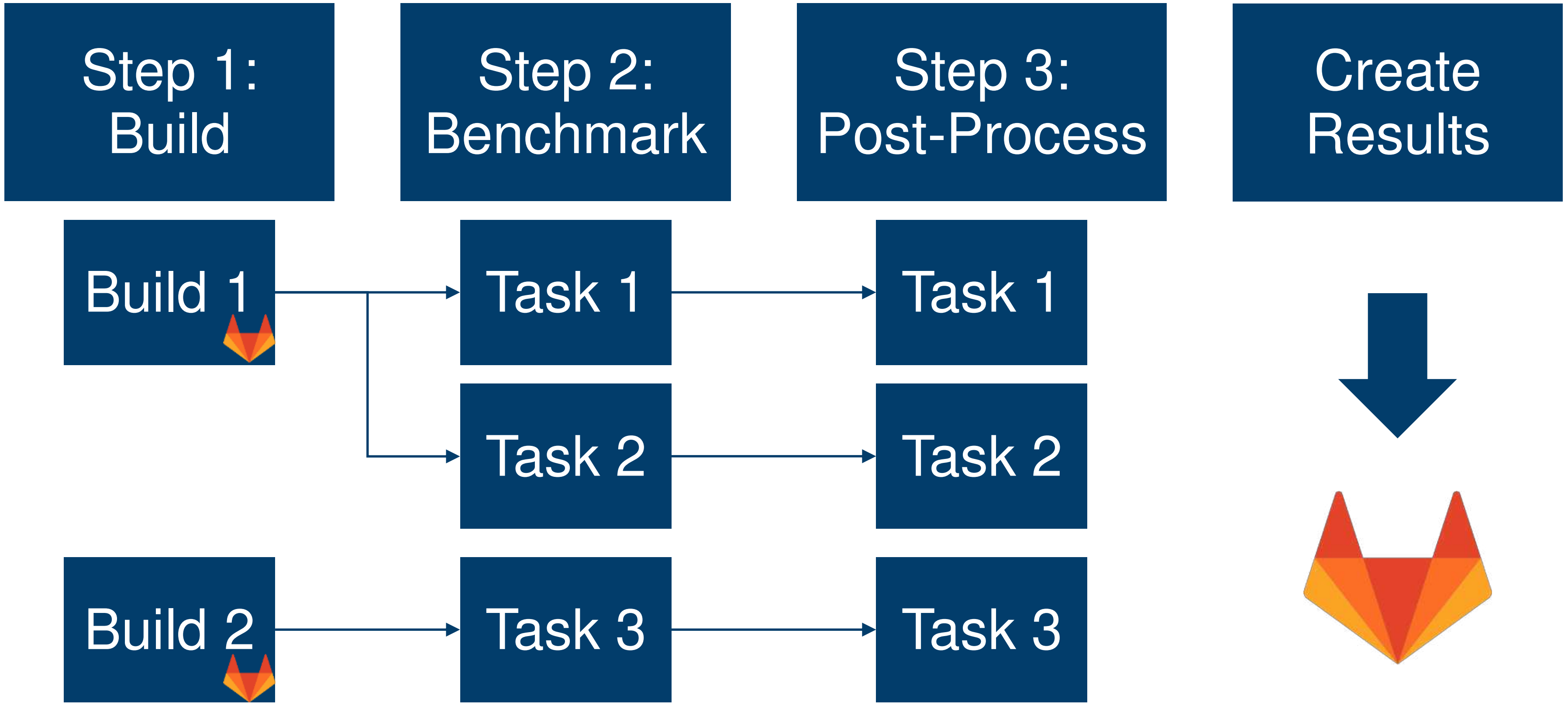
Von GitLab B.V. - gitlab.com/gitlab-org/gitlab-ce/blob/master/app/assets/images/logo wordmark.svg, MIT, https://commons.wikimedia.org/w/index.php?curid=74127414

Von rar-labs - http://www.seeklogo.com/files/W/WinRAR-vector-logo-9A6D30BEBC-seeklogo.com.zip / farben von Datei:WinRAR-Logo.png, Logo, https://de.wikipedia.org/w/index.php?curid=6203168

THE BENCHMARKS

- Each Benchmark will consist of at least two steps:
  - Step 1: Build (compile, download files & executable, etc.). We always want to be using the freshest build possible.
  - Step 2: Run benchmark.
  - Step 3: (Optional) post-process results.
- After benchmark completion result tables are generated.
  - 1 master table is appended to.
  - 1 separate table named after the day is also generated.
- These tables are then pushed to GitLab.

THE BENCHMARKS



GITLAB

- One Group per project (DEEP- and IO-SEA groups present).
- One repository per use-case software or synthetic benchmark.
  - One branch per system (not an issue for DEEP).
  - Access to use-case repositories restricted to benchmarking team and use-case developers.
  - Everyone has access to synthetic benchmarks, master branch edits restricted to benchmarking team.

BOTS

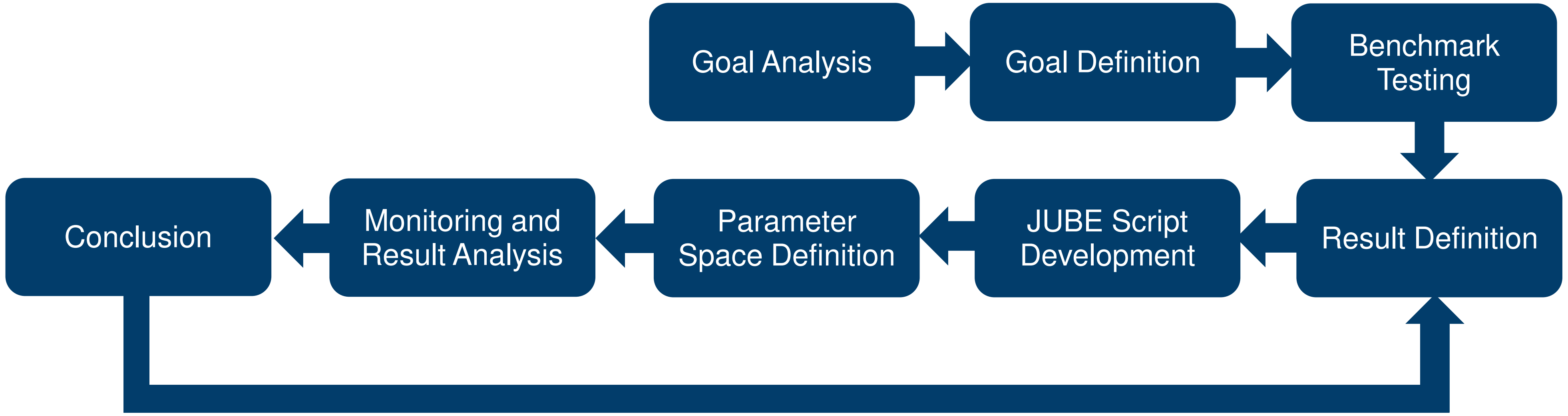
- Benchmarks will be executed on a schedule by bots.
  - Weekly or bi-weekly (TBD)
- Result post processing by the bot is possible.
- Aggregated results will automatically be uploaded to GitLab.
- All pertinent benchmark data is stored locally on the system.
  - These data can be archived. They just need to be accessible for inspection at a later date.
  - The data can be deleted at project end.

BOTS & GITLAB

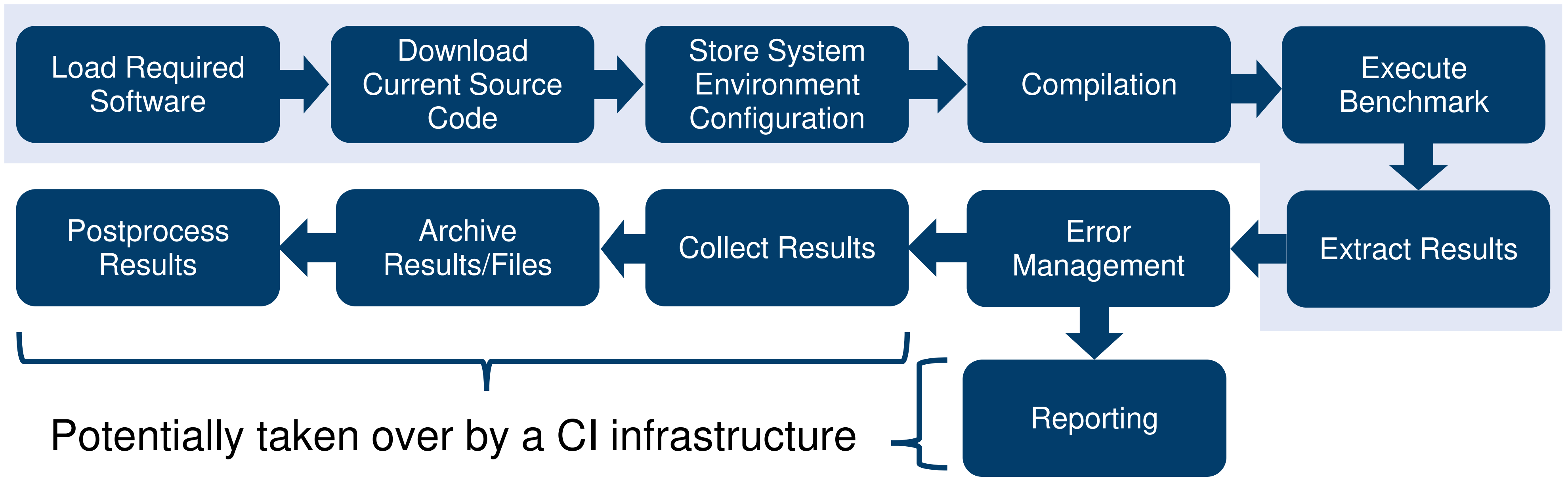
- The Bots will have GitLab accounts.
- This is mostly so that we can distinguish between user-generated benchmark data, which you are more than welcome to generate, and those created on a schedule.
- GitLab access will be provided via tokens.

```
# 0. Parameters
username  ="deep_bench";           #The bot that will be reporting the benchmark results.
usetoken  ="????????????????????"; #The access token of the bot.
usermail  ="sc@fz-juelich.de";     #The bot does not have an email address so we use JSC support.
repository="gitlab.version.fz-juelich.de/holicki1/benchmarking-workshop-linktest.git";

# 1. Switch to deep_bench user for reporting
git config user.name  ${username}; #Switch user
git config user.email ${usermail}; #Switch email
# 2. Change remote url to use access token
git remote set-url origin https://${username}:${usetoken}@${repository};
# 3. Create system repository
git checkout -b ${SYSTEMNAME};
```

SAMPLE BENCHMARK INTEGRATION WORKFLOW

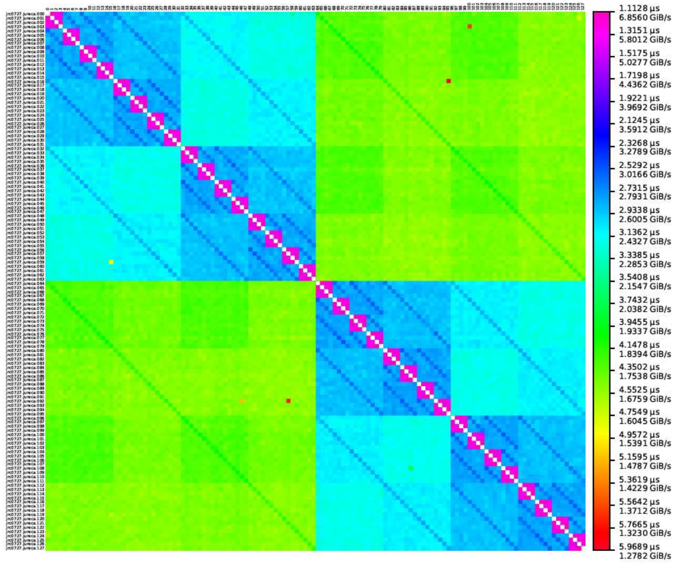


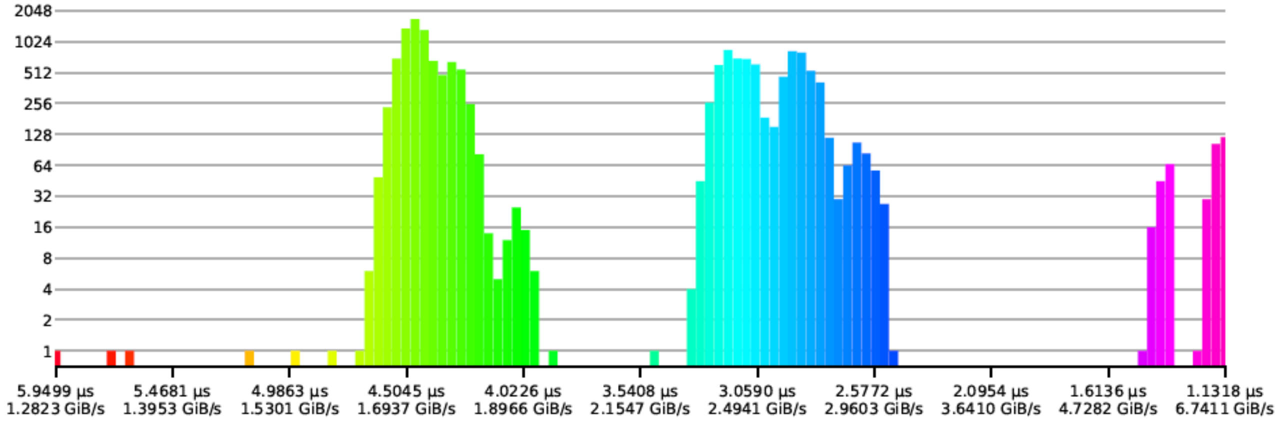
SAMPLE BENCHMARK EXECUTION WORKFLOW



MAKING EFFECTIVE BENCHMARKS

- KISS (Keep It Simple Stupid)
- Only add as much complexity as necessary
- Benchmarks should be short and to the point
- Time things as precisely as possible
- Output intelligently to log files
- Make benchmark quantities easily findable
- Implement proper error checking at the end
- Have Fun!



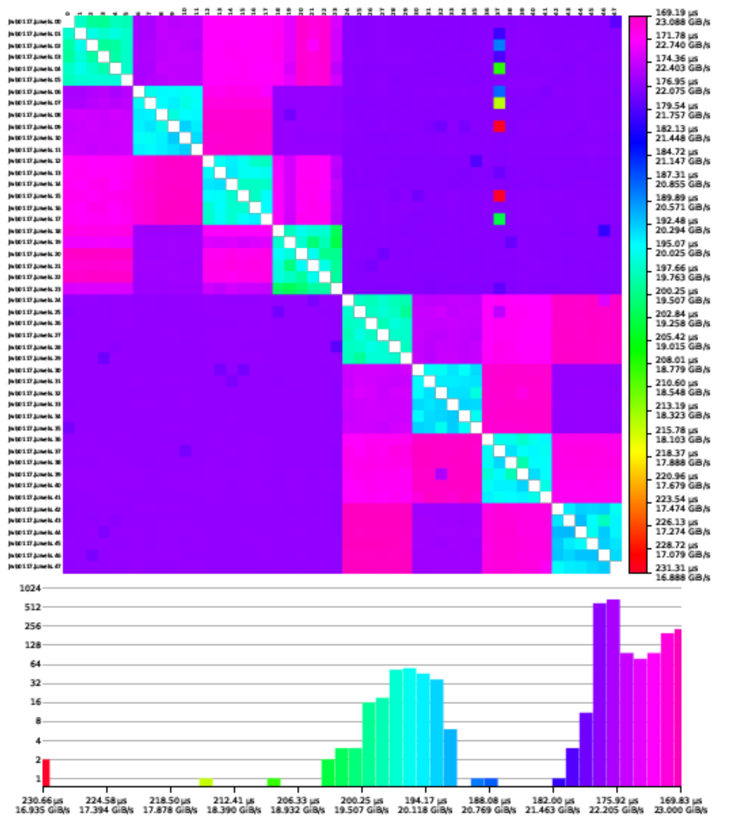


LINKTEST EXAMPLE

A basic example JUBE benchmark using linktest

LINKTEST

- Peer-to-peer message-passing timing benchmark
- Supports: MPI, IB verbs, PSM2, UCP, TCP &
  NVLink via CUDA calls
- Supports (non-)blocking send and recv calls
  - Does not support collective calls aside from MPI all-to-all
- Testing by default in parallel
- Uses either CPU (default) or GPU RAM
- Can perform bidirectional tests
- Can perform bisection tests

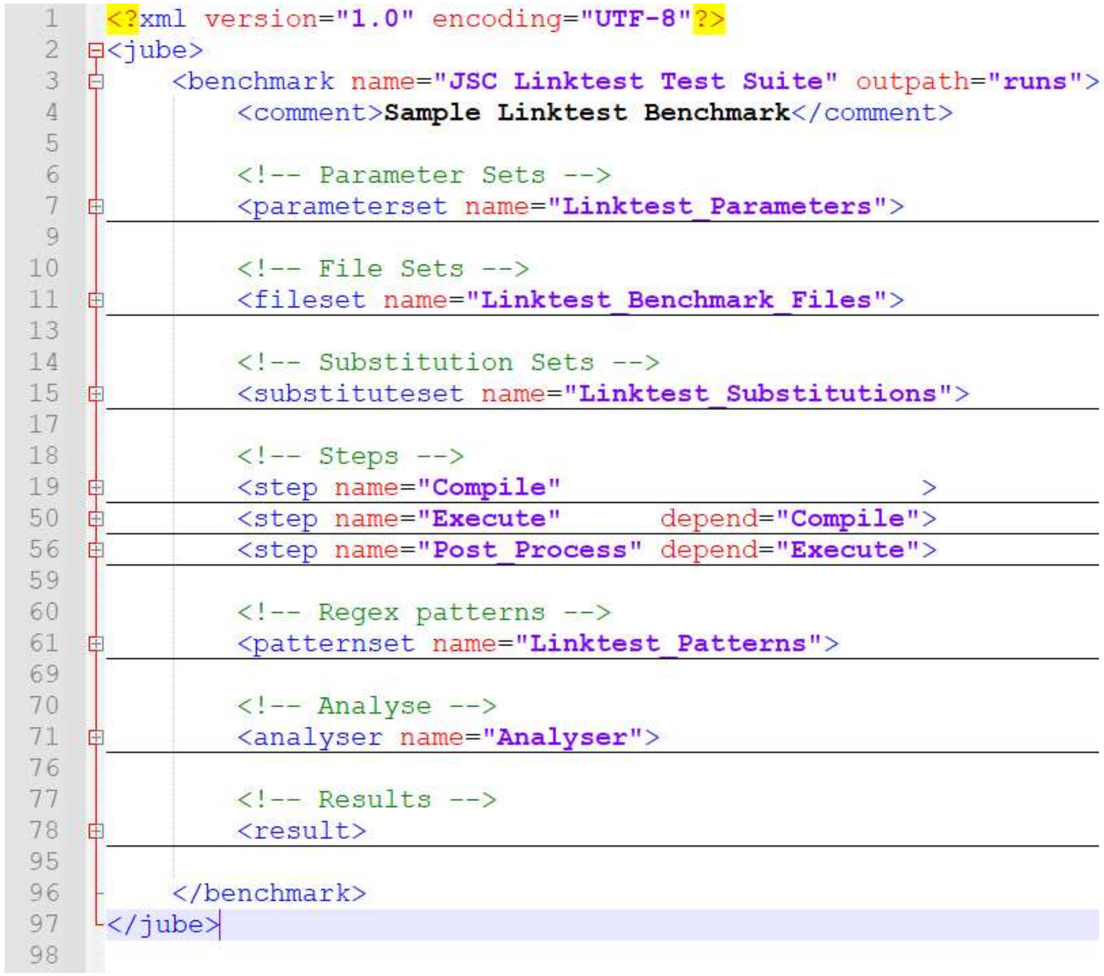


Communication:	mpi	Number of Tasks:	48	Number of Hosts:	1
Number of Messages:	100	Message Length:	4,194,304 B (4.0000 MiB)		
# Warm-Up Messages:	5	# Serial Retests:	0	Domain:	
Execution Order:	Serial	All-to-All:	No	Mixed Ranks:	No
Bidirectional:	No	Bisection:	No	Use GPUs:	No
Min Value:	169.19 µs 23.088 GiB/s	Average Value:	173.77 µs 22.479 GiB/s	Max Value:	231.31 µs 16.888 GiB/s

Linktest Version: 2.1.7 Report generated by the FZJ Linktest Python Report Tool 1.0.0, Forschungszentrum Jülich GmbH

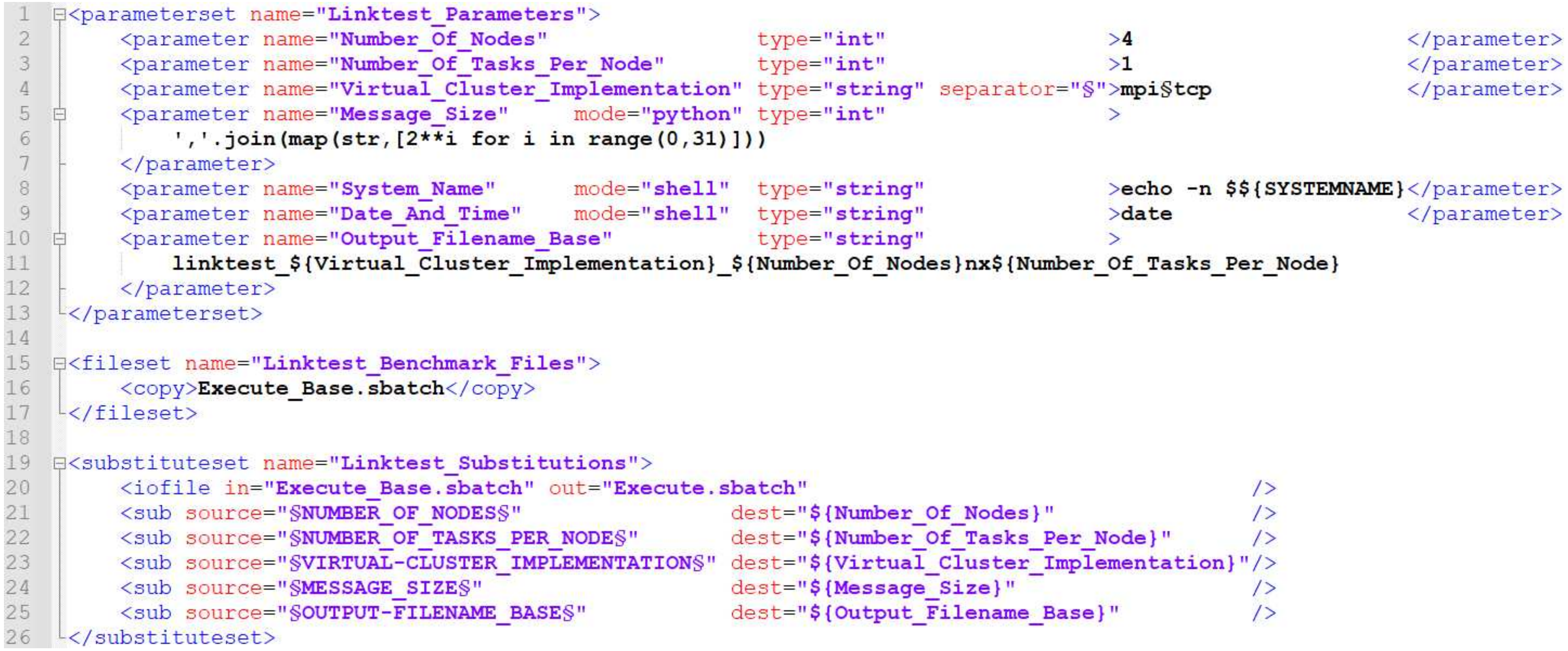
WHAT DOES A SAMPLE JUBE SCRIPT LOOK LIKE?

This is XML! YAML also works.



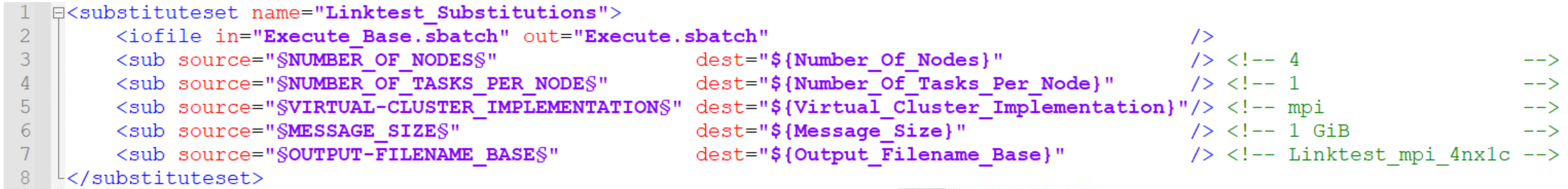
```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
    <benchmark name="JSC Linktest Test Suite" outpath="runs">
        <comment>Sample Linktest Benchmark</comment>

        <!-- Parameter Sets -->
        <parameterset name="Linktest_Parameters">

        <!-- File Sets -->
        <fileset name="Linktest_Benchmark_Files">

        <!-- Substitution Sets -->
        <substituteset name="Linktest_Substitutions">

        <!-- Steps -->
        <step name="Compile"                     >
        <step name="Execute"      depend="Compile">
        <step name="Post_Process" depend="Execute">

        <!-- Regex patterns -->
        <patternset name="Linktest_Patterns">

        <!-- Analyse -->
        <analyser name="Analyser">

        <!-- Results -->
        <result>

    </benchmark>
</jube>
```

WHAT JUBE SETS LOOK LIKE

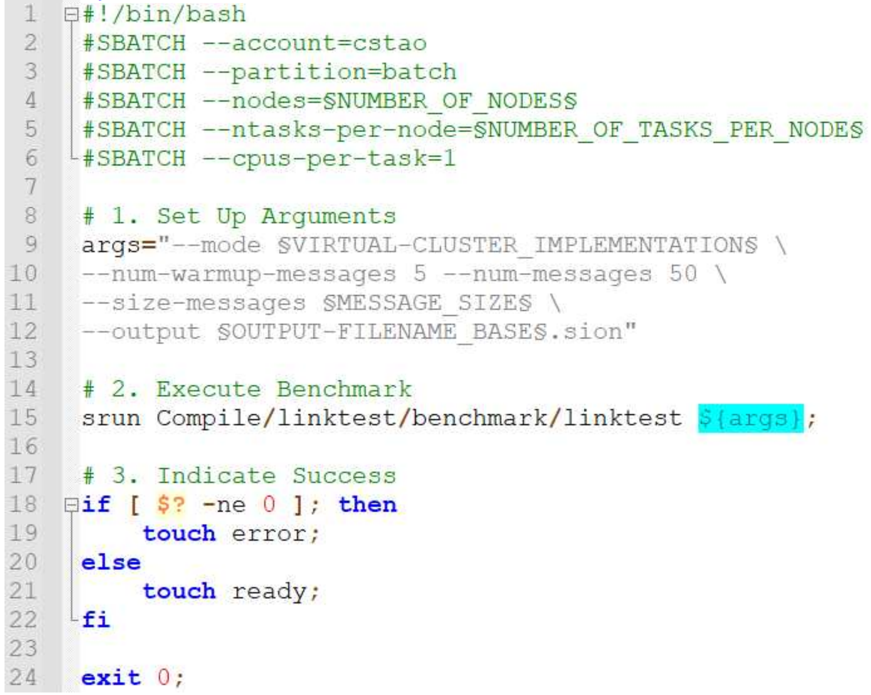


```
<parameterset name="Linktest_Parameters">
    <parameter name="Number_Of_Nodes"                type="int"                 >4                     </parameter>
    <parameter name="Number_Of_Tasks_Per_Node"       type="int"                 >1                     </parameter>
    <parameter name="Virtual_Cluster_Implementation" type="string" separator="§">mpi§tcp               </parameter>
    <parameter name="Message_Size"     mode="python" type="int"                 >
        ','.join(map(str,[2**i for i in range(0,31)]))
    </parameter>
    <parameter name="System_Name"      mode="shell"  type="string"              >echo -n $${SYSTEMNAME}</parameter>
    <parameter name="Date_And_Time"    mode="shell"  type="string"              >date                  </parameter>
    <parameter name="Output_Filename_Base"           type="string"              >
        linktest_${Virtual_Cluster_Implementation}_${Number_Of_Nodes}nx${Number_Of_Tasks_Per_Node}
    </parameter>
</parameterset>

<fileset name="Linktest_Benchmark_Files">
    <copy>Execute_Base.sbatch</copy>
</fileset>

<substituteset name="Linktest_Substitutions">
    <iofile in="Execute_Base.sbatch" out="Execute.sbatch"                                  />
    <sub source="§NUMBER_OF_NODES§"                dest="${Number_Of_Nodes}"               />
    <sub source="§NUMBER_OF_TASKS_PER_NODE§"       dest="${Number_Of_Tasks_Per_Node}"      />
    <sub source="§VIRTUAL-CLUSTER_IMPLEMENTATION§" dest="${Virtual_Cluster_Implementation}"/>
    <sub source="§MESSAGE_SIZE§"                   dest="${Message_Size}"                  />
    <sub source="§OUTPUT-FILENAME_BASE§"           dest="${Output_Filename_Base}"          />
</substituteset>
```

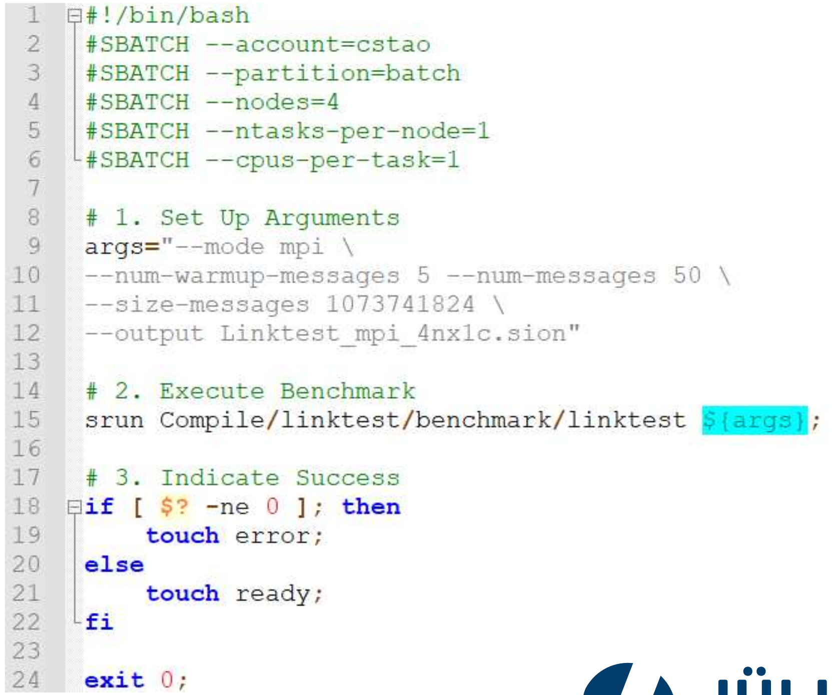
HOW SUBSTITUTIONS WORK



```xml
<substituteset name="Linktest_Substitutions">
    <iofile in="Execute_Base.sbatch" out="Execute.sbatch"                                  />
    <sub source="$NUMBER_OF_NODES$"                dest="${Number_Of_Nodes}"               /> <!-- 4                  -->
    <sub source="$NUMBER_OF_TASKS_PER_NODE$"       dest="${Number_Of_Tasks_Per_Node}"      /> <!-- 1                  -->
    <sub source="$VIRTUAL-CLUSTER_IMPLEMENTATION$" dest="${Virtual_Cluster_Implementation}"/> <!-- mpi                -->
    <sub source="$MESSAGE_SIZE$"                   dest="${Message_Size}"                  /> <!-- 1 GiB              -->
    <sub source="$OUTPUT-FILENAME_BASE$"           dest="${Output_Filename_Base}"          /> <!-- Linktest_mpi_4nx1c -->
</substituteset>
```

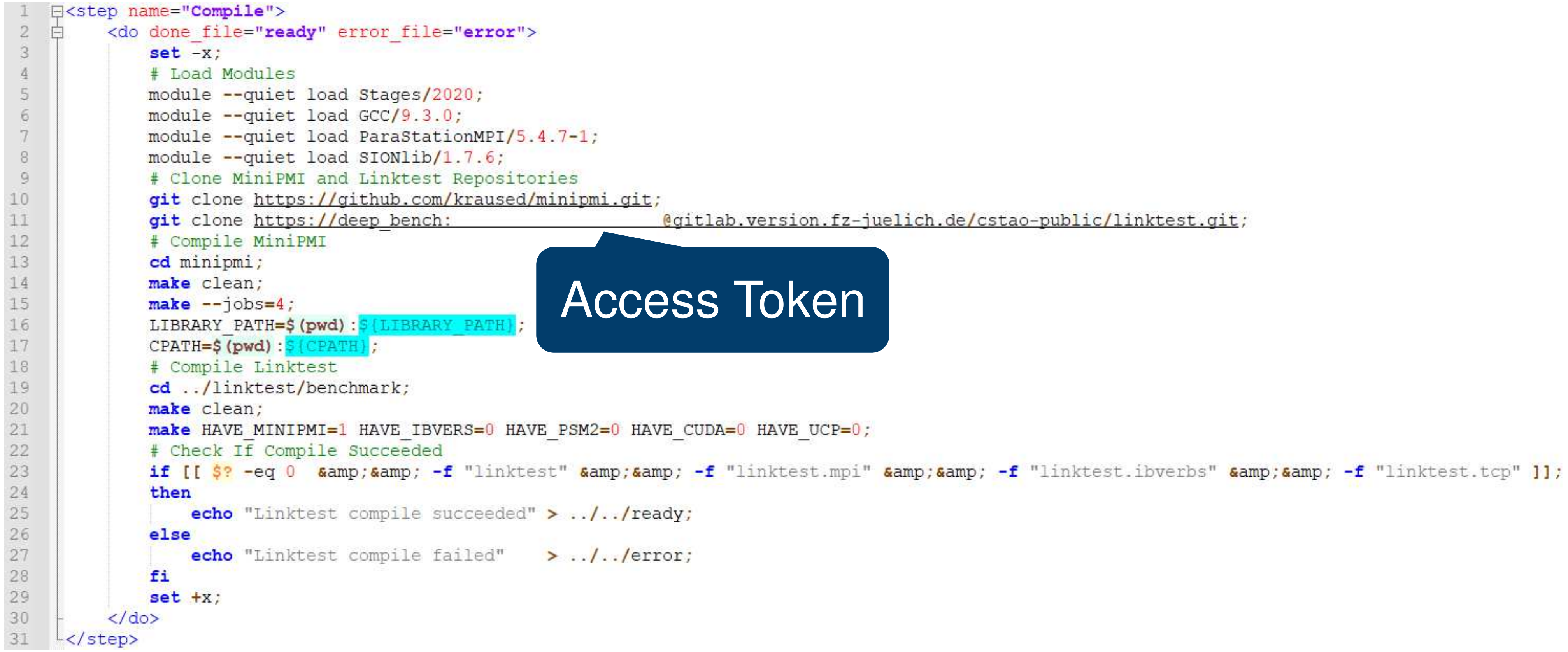


```bash
#!/bin/bash
#SBATCH --account=cstao
#SBATCH --partition=batch
#SBATCH --nodes=$NUMBER_OF_NODES$
#SBATCH --ntasks-per-node=$NUMBER_OF_TASKS_PER_NODE$
#SBATCH --cpus-per-task=1

# 1. Set Up Arguments
args="--mode $VIRTUAL-CLUSTER_IMPLEMENTATION$ \
--num-warmup-messages 5 --num-messages 50 \
--size-messages $MESSAGE_SIZE$ \
--output $OUTPUT-FILENAME_BASE$.sion"

# 2. Execute Benchmark
srun Compile/linktest/benchmark/linktest ${args};

# 3. Indicate Success
if [ $? -ne 0 ]; then
    touch error;
else
    touch ready;
fi

exit 0;
```



```bash
#!/bin/bash
#SBATCH --account=cstao
#SBATCH --partition=batch
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1

# 1. Set Up Arguments
args="--mode mpi \
--num-warmup-messages 5 --num-messages 50 \
--size-messages 1073741824 \
--output Linktest_mpi_4nx1c.sion"

# 2. Execute Benchmark
srun Compile/linktest/benchmark/linktest ${args};

# 3. Indicate Success
if [ $? -ne 0 ]; then
    touch error;
else
    touch ready;
fi

exit 0;
```

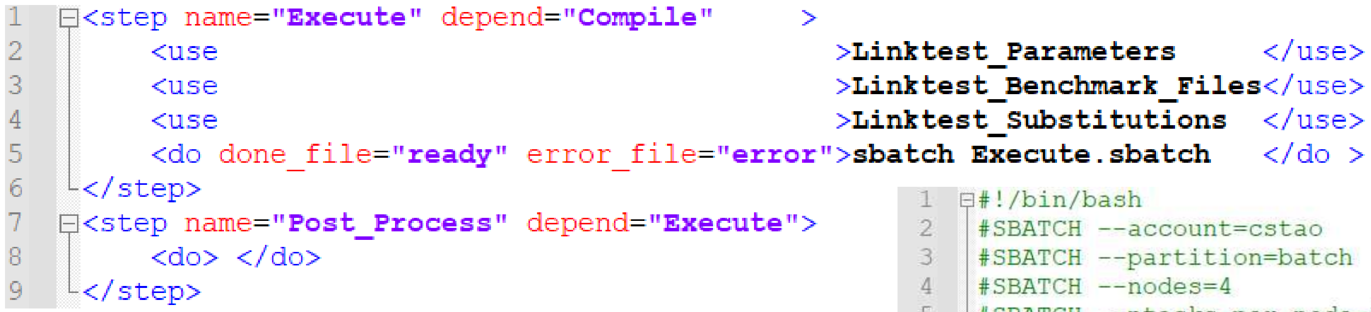
WHAT THE COMPILE STEP MIGHT LOOK LIKE



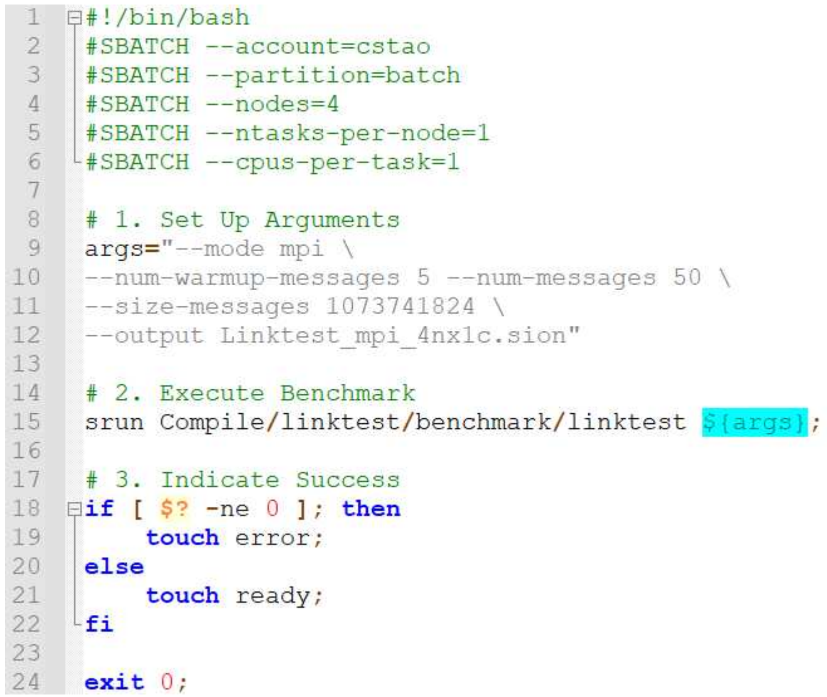
```
<step name="Compile">
    <do done_file="ready" error_file="error">
        set -x;
        # Load Modules
        module --quiet load Stages/2020;
        module --quiet load GCC/9.3.0;
        module --quiet load ParaStationMPI/5.4.7-1;
        module --quiet load SIONlib/1.7.6;
        # Clone MiniPMI and Linktest Repositories
        git clone https://github.com/kraused/minipmi.git;
        git clone https://deep_bench:                    @gitlab.version.fz-juelich.de/cstao-public/linktest.git;
        # Compile MiniPMI
        cd minipmi;
        make clean;
        make --jobs=4;
        LIBRARY_PATH=$(pwd):${LIBRARY_PATH};
        CPATH=$(pwd):${CPATH};
        # Compile Linktest
        cd ../linktest/benchmark;
        make clean;
        make HAVE_MINIPMI=1 HAVE_IBVERS=0 HAVE_PSM2=0 HAVE_CUDA=0 HAVE_UCP=0;
        # Check If Compile Succeeded
        if [[ $? -eq 0  &amp;&amp; -f "linktest" &amp;&amp; -f "linktest.mpi" &amp;&amp; -f "linktest.ibverbs" &amp;&amp; -f "linktest.tcp" ]];
        then
            echo "Linktest compile succeeded" > ../../ready;
        else
            echo "Linktest compile failed"    > ../../error;
        fi
        set +x;
    </do>
</step>
```

Access Token

WHAT THE OTHER STEPS LOOK LIKE



```xml
<step name="Execute" depend="Compile"     >
    <use                                  >Linktest_Parameters     </use>
    <use                                  >Linktest_Benchmark_Files</use>
    <use                                  >Linktest_Substitutions  </use>
    <do done_file="ready" error_file="error">sbatch Execute.sbatch   </do >
</step>
<step name="Post_Process" depend="Execute">
    <do> </do>
</step>
```



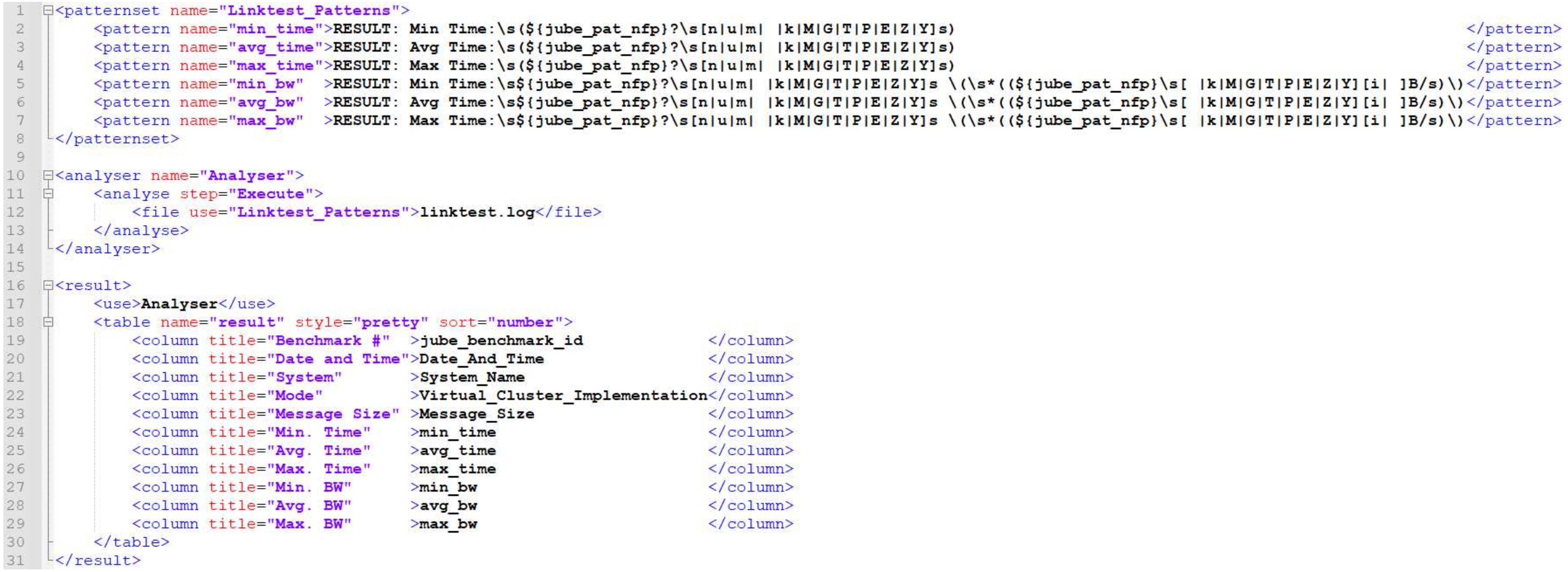
```bash
#!/bin/bash
#SBATCH --account=cstao
#SBATCH --partition=batch
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1

# 1. Set Up Arguments
args="--mode mpi \
--num-warmup-messages 5 --num-messages 50 \
--size-messages 1073741824 \
--output Linktest_mpi_4nx1c.sion"

# 2. Execute Benchmark
srun Compile/linktest/benchmark/linktest ${args};

# 3. Indicate Success
if [ $? -ne 0 ]; then
    touch error;
else
    touch ready;
fi

exit 0;
```

You can submit batch jobs, just remember to create a file to indicate completion/error. JUBE tests for this to check if it should continue running the benchmark.

The post-process step in this case is not required.

HOW RESULTS WORK

https://regex101.com/

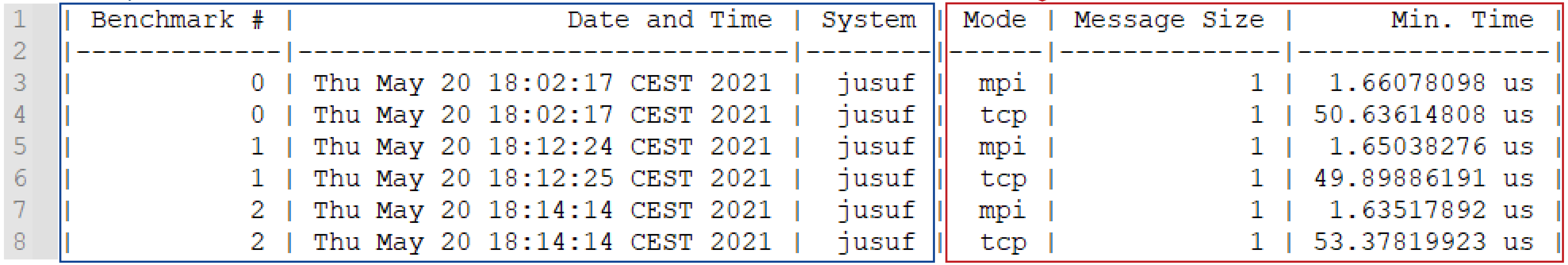


```
<patternset name="Linktest_Patterns">
    <pattern name="min_time">RESULT: Min Time:\s(${jube_pat_nfp}?\s[n|u|m| |k|M|G|T|P|E|Z|Y]s)                                                      </pattern>
    <pattern name="avg_time">RESULT: Avg Time:\s(${jube_pat_nfp}?\s[n|u|m| |k|M|G|T|P|E|Z|Y]s)                                                      </pattern>
    <pattern name="max_time">RESULT: Max Time:\s(${jube_pat_nfp}?\s[n|u|m| |k|M|G|T|P|E|Z|Y]s)                                                      </pattern>
    <pattern name="min_bw"  >RESULT: Min Time:\s${jube_pat_nfp}?\s[n|u|m| |k|M|G|T|P|E|Z|Y]s \(\s*((${jube_pat_nfp}\s[ |k|M|G|T|P|E|Z|Y][i| ]B/s)\)</pattern>
    <pattern name="avg_bw"  >RESULT: Avg Time:\s${jube_pat_nfp}?\s[n|u|m| |k|M|G|T|P|E|Z|Y]s \(\s*((${jube_pat_nfp}\s[ |k|M|G|T|P|E|Z|Y][i| ]B/s)\)</pattern>
    <pattern name="max_bw"  >RESULT: Max Time:\s${jube_pat_nfp}?\s[n|u|m| |k|M|G|T|P|E|Z|Y]s \(\s*((${jube_pat_nfp}\s[ |k|M|G|T|P|E|Z|Y][i| ]B/s)\)</pattern>
</patternset>

<analyser name="Analyser">
    <analyse step="Execute">
        <file use="Linktest_Patterns">linktest.log</file>
    </analyse>
</analyser>

<result>
    <use>Analyser</use>
    <table name="result" style="pretty" sort="number">
        <column title="Benchmark #"  >jube_benchmark_id             </column>
        <column title="Date and Time">Date_And_Time                 </column>
        <column title="System"       >System_Name                   </column>
        <column title="Mode"         >Virtual_Cluster_Implementation</column>
        <column title="Message Size" >Message_Size                  </column>
        <column title="Min. Time"    >min_time                      </column>
        <column title="Avg. Time"    >avg_time                      </column>
        <column title="Max. Time"    >max_time                      </column>
        <column title="Min. BW"      >min_bw                        </column>
        <column title="Avg. BW"      >avg_bw                        </column>
        <column title="Max. BW"      >max_bw                        </column>
    </table>
</result>
```

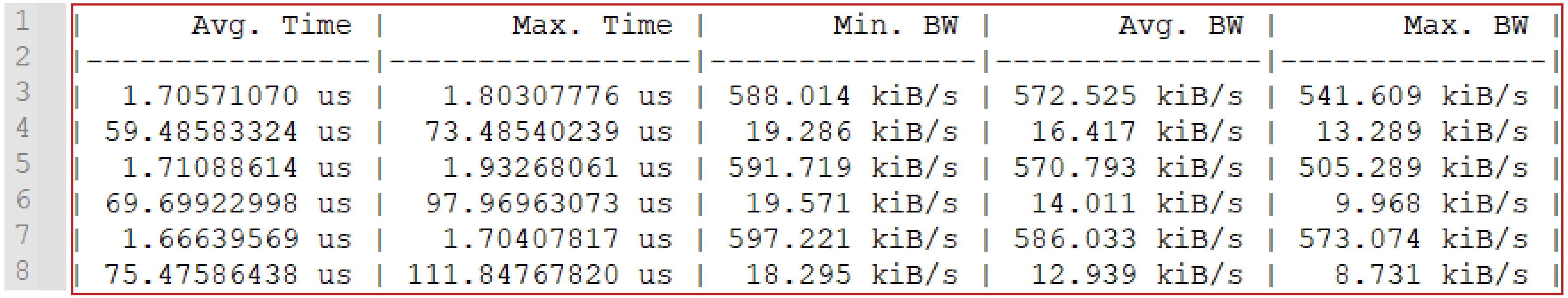
HOW RESULTS WORK

Meta Data

Benchmark Data



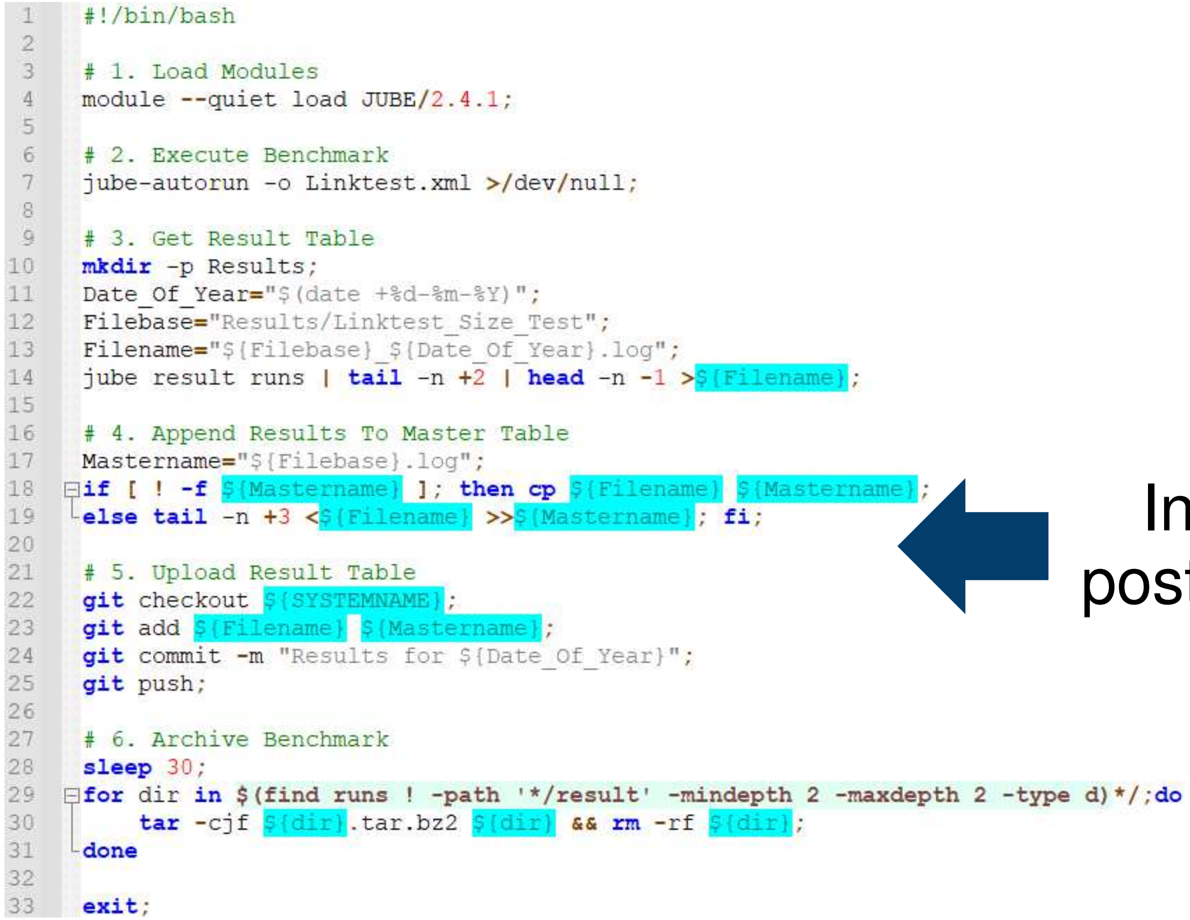
```
| Benchmark # |                  Date and Time | System | Mode | Message Size |        Min. Time |
|-------------|--------------------------------|--------|------|--------------|------------------|
|           0 | Thu May 20 18:02:17 CEST 2021 |  jusuf |  mpi |            1 |  1.66078098 us |
|           0 | Thu May 20 18:02:17 CEST 2021 |  jusuf |  tcp |            1 | 50.63614808 us |
|           1 | Thu May 20 18:12:24 CEST 2021 |  jusuf |  mpi |            1 |  1.65038276 us |
|           1 | Thu May 20 18:12:25 CEST 2021 |  jusuf |  tcp |            1 | 49.89886191 us |
|           2 | Thu May 20 18:14:14 CEST 2021 |  jusuf |  mpi |            1 |  1.63517892 us |
|           2 | Thu May 20 18:14:14 CEST 2021 |  jusuf |  tcp |            1 | 53.37819923 us |
```



```
|       Avg. Time |       Max. Time |       Min. BW |       Avg. BW |       Max. BW |
|-----------------|-----------------|---------------|---------------|---------------|
|   1.70571070 us |   1.80307776 us | 588.014 kiB/s | 572.525 kiB/s | 541.609 kiB/s |
|  59.48583324 us |  73.48540239 us |  19.286 kiB/s |  16.417 kiB/s |  13.289 kiB/s |
|   1.71088614 us |   1.93268061 us | 591.719 kiB/s | 570.793 kiB/s | 505.289 kiB/s |
|  69.69922998 us |  97.96963073 us |  19.571 kiB/s |  14.011 kiB/s |   9.968 kiB/s |
|   1.66639569 us |   1.70407817 us | 597.221 kiB/s | 586.033 kiB/s | 573.074 kiB/s |
|  75.47586438 us | 111.84767820 us |  18.295 kiB/s |  12.939 kiB/s |   8.731 kiB/s |
```

JÜLICH Forschungszentrum

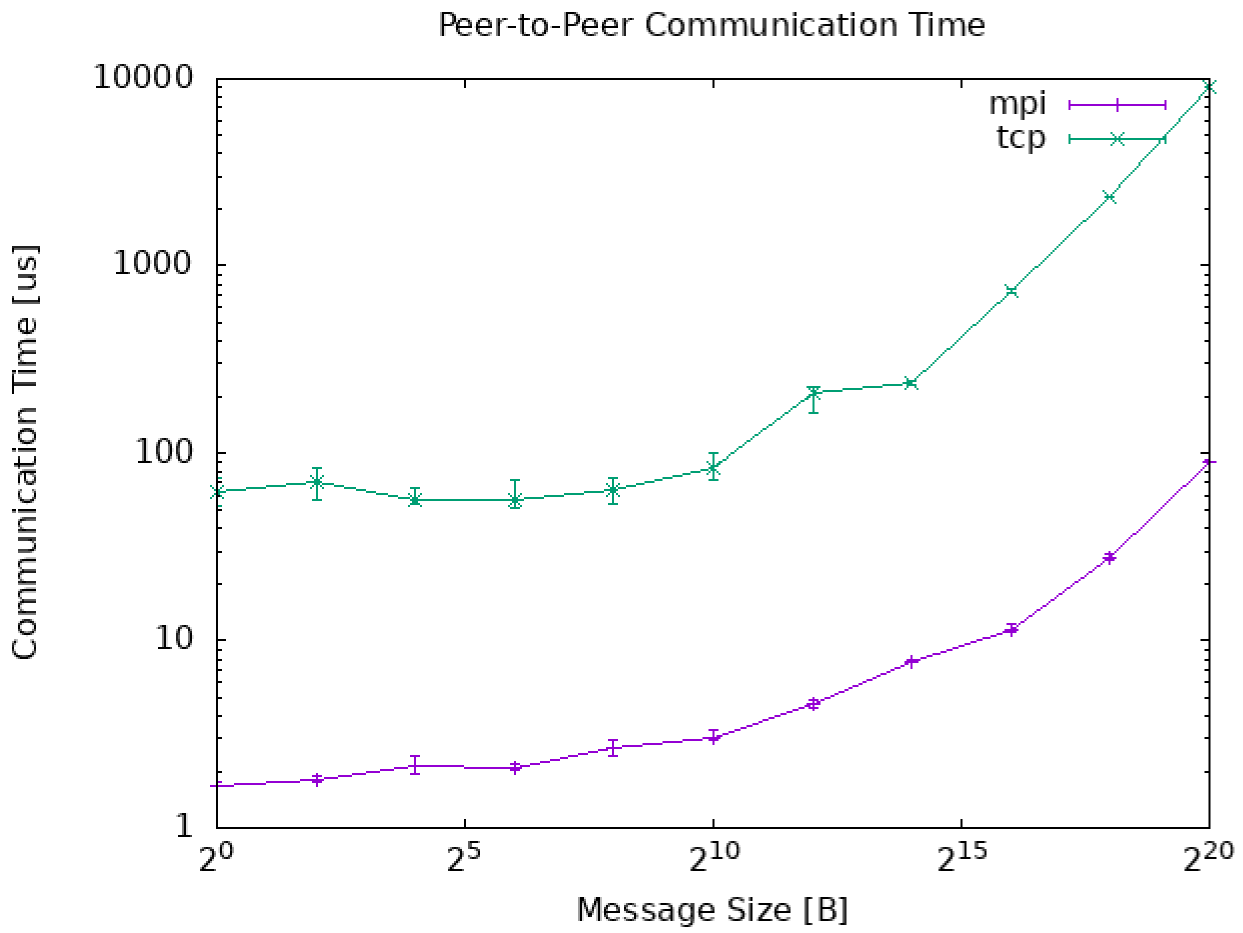
THE JUBE CRON JOB



```bash
#!/bin/bash

# 1. Load Modules
module --quiet load JUBE/2.4.1;

# 2. Execute Benchmark
jube-autorun -o Linktest.xml >/dev/null;

# 3. Get Result Table
mkdir -p Results;
Date_Of_Year="$(date +%d-%m-%Y)";
Filebase="Results/Linktest_Size_Test";
Filename="${Filebase}_${Date_Of_Year}.log";
jube result runs | tail -n +2 | head -n -1 >${Filename};

# 4. Append Results To Master Table
Mastername="${Filebase}.log";
if [ ! -f ${Mastername} ]; then cp ${Filename} ${Mastername};
else tail -n +3 <${Filename} >>${Mastername}; fi;

# 5. Upload Result Table
git checkout ${SYSTEMNAME};
git add ${Filename} ${Mastername};
git commit -m "Results for ${Date_Of_Year}";
git push;

# 6. Archive Benchmark
sleep 30;
for dir in $(find runs ! -path '*/result' -mindepth 2 -maxdepth 2 -type d)*/;do
    tar -cjf ${dir}.tar.bz2 ${dir} && rm -rf ${dir};
done

exit;
```

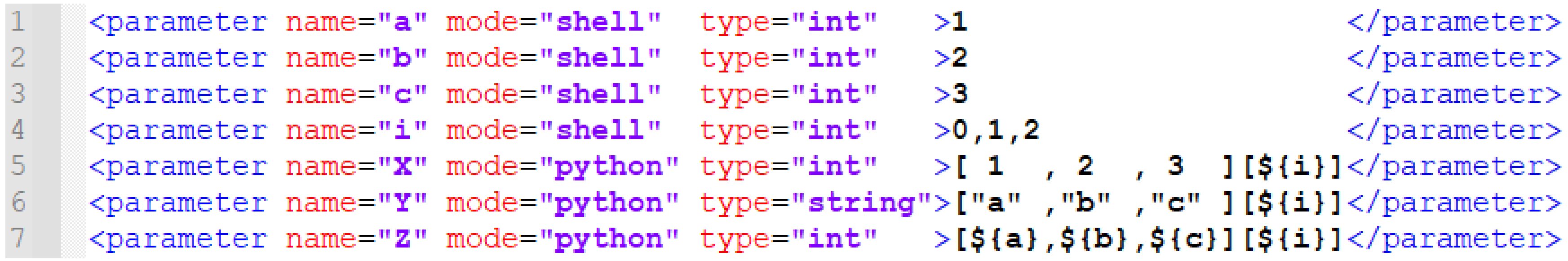
Insert result-table post-processing here.

POST-PROCESSING EXAMPLE



ONE LAST TIP

You can use parameters to select other parameters!



```
<parameter name="a" mode="shell"  type="int"   >1                     </parameter>
<parameter name="b" mode="shell"  type="int"   >2                     </parameter>
<parameter name="c" mode="shell"  type="int"   >3                     </parameter>
<parameter name="i" mode="shell"  type="int"   >0,1,2                 </parameter>
<parameter name="X" mode="python" type="int"   >[ 1  , 2  , 3  ][${i}]</parameter>
<parameter name="Y" mode="python" type="string">["a" ,"b" ,"c" ][${i}]</parameter>
<parameter name="Z" mode="python" type="int"   >[${a},${b},${c}][${i}]</parameter>
```

Thank You! Questions?!

DR. MAX HOLICKI | JÜLICH SUPERCOMPUTING CENTRE

Live Demonstration

DR. MAX HOLICKI | JÜLICH SUPERCOMPUTING CENTRE

EFFECTIVE JUBE BENCHMARK CREATION

- KISS (Keep It Simple Stupid)
- Use precompiled binaries
- Skip compile step for now (replace it with a copy)
- Only include relevant parameters at the start
- Use Python for parameters
- Use JUBE debug mode for dry runs 1st
- To set up the regex patterns use update (saves rerunning)
- Use a regex pattern builder
- Do not forget the ready/error files
- Have a look at the JUBE glossary https://apps.fz-juelich.de/jsc/jube/jube2/docu/glossar.html

Have Fun!