




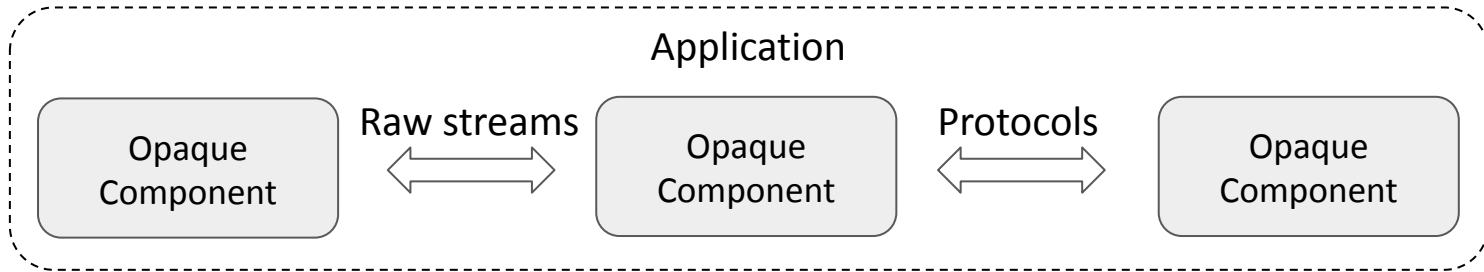
Fractal: Fault-Tolerant Shell-Script Distribution

Zhicheng Huang*, Ramiz Dundar*, **Yizheng Xie**, Konstantinos Kallas^{UCLA}, Nikos Vasilakis

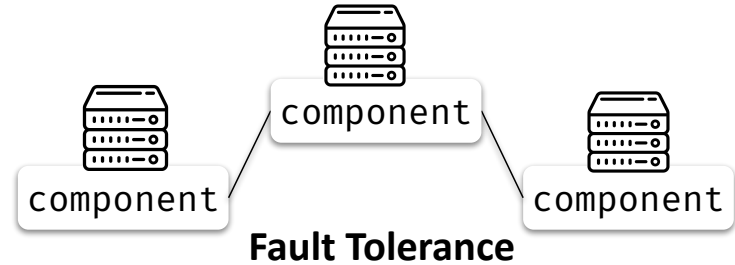
 github.com/binpash/fractal



Opaque Software Components are Pervasive



... Shell Programs



Shell: A Concrete and Pervasive Example

Scripts for Bioinformatics

A collection of re-usable bash/C++ scripts for processing

Anuradha Wickramarachchi Follow 3 min read · M

101

In this article, we will present you with a set of handy scripts you can use to pre-process your sequencing data.

PublMed

Nucleic Acids Res. 2021 Nov 18;49(20)

TERA-Seq: true end-to-end RNA molecules for transcriptomics

Fadia Ibrahim^{1,2}, Jan Oppett¹, Manolis Kellis¹

Written by: Marcin Buczkowski

linuxbash

Linux Bash

March 2025 • Artificial Intelligence

Automating statistical analysis with Bash


Replace MCP With CLI , The Best AI Agent Interface Already Exists

The Most Powerful Tool Protocol For AI Agents Has Been On Every Unix System Since 1971


COBUS GREYLING
FEB 13, 2026

2 1 Share


AI Agent Integration



(MCP)
Model Context Protocol



VS



Command-Line Interface (CLI)

+more!



... Shell Programs

Existing Shell Programs Distribution Systems Lack Fault Tolerance

Extending Unix Pipelines to DAGs

Diomidis Spinellis, *Senior Member, IEEE*, and Marios Fragkoulis

PASH: Light-touch Data-Parallel Shell Processing

Nikos Vasilakis* Konstantinos Kallas* Konstantinos Mamouras

Executing Shell Scripts in the Wrong Order, Correctly

Georgios Liargkovas
gliargovas@aueb.gr

Konstantinos Kallas
kallas@seas.upenn.edu

Parallelizing the Execution of Sequential Scripts

Zhao Zhang Daniel S. Katz Timothy G. Armstrong

Support for graphs of processes in a command interpreter

[Chris McDonald](#), [Trevor I. Dix](#)

Optimizing Shell Scripting Languages

Emery D. Berger

Sun Grid Engine: Towards Creating a Compute Power Grid

POSH: A Data-Aware Shell

Deepti Raghavan Sadjad Fouladi Philip Levis Matei

GNU Parallel: The Command-Line Power Tool

OLE TANGE

SLURM: Simple Linux Utility for Resource Management

DiSh: Dynamic Shell-Script Distribution

Tammam Mustafa
MIT

Konstantinos Kallas
University of Pennsylvania

The Server

Aurèle Mahéo
Télécom SudParis
Palaiseau, france

aurele.maheo@telecom-sudparis.eu

Pierre Sutra
Télécom SudParis
Palaiseau, france

pierre.sutra@telecom-sudparis.eu

Aristan Tarrant
RedHat
London, United Kingdom
ttarrant@redhat.com

Not Fault-Tolerant!

An Example Pipeline

```
#!/bin/bash
```

```
bots='Googlebot|Bingbot|Baiduspider'
```

```
for log in $(hdfs dfs -ls -C $in); do
```

```
  # Identify bot IPs by visit frequency
```

```
  hdfs dfs -cat $log |
```

```
  grep -E $bots |
```

filtering

```
  cut -d" " -f1 |
```

extracting

```
  sort |
```

calculating

```
  uniq -c |
```

```
  sort -rn >> ${log}.out
```

sorting

```
  # ...
```

```
done
```

Rewriting the Pipeline Using Fault-Tolerant Systems

```
#!/bin/bash
```

```
bots='Googlebot|Bingbot|Baiduspider'
```

```
for log in $(hdfs dfs -ls -C $in); do
```

```
# Identify bot IPs by visit frequency
```

```
hdfs dfs -cat $log |
```

```
grep -E $bots |
```

filtering

```
cut -d" " -f1 |
```

extracting

```
sort |
```

calculating

```
uniq -c |
```


calculating

```
sort -rn >> ${log}.out
```

sorting

```
# ...
```

```
done
```

```
sc.textFile(log)
  .filter(lambda l: p.search(l)...)
  .map(lambda l: l.split(" ")[0])
  .map(lambda key: (key, 1))
  .reduceByKey(lambda a, b: a + b)
  .sortBy(lambda kv: kv[1]...) 
```

```
# mapper.py
```

```
for l in stdin:
```

```
if bots.search(l)...
```

```
# reducer.py
```

```
for l in sys.stdin:
```

```
k, v = l.split(...)
```



Manual Rewriting is Inefficient and Error-Prone

```
7  v pure_func() {
8      tempfiles=$(mktemp)
9
10     tee $tempfile | cut -d "\"" -f3 | cut -d ' ' -f2 | sort | uniq -c | sort -rn
11     # awk alternative, too slow
12     awk '{print $9}' $tempfile | sort | uniq -c | sort -rn
13     # find broken links broken links
14     awk '{ $9 ~ /404/ }' $tempfile | awk '{print $7}' | sort | uniq -c | sort -rn
15     # for 502 (bad-gateway) we can run following command:
16     awk '{ $9 ~ /502/ }' $tempfile | awk '{print $7}' | sort | uniq -c | sort -r
17     # Who are requesting broken links (or URLs resulting in 502)
18     awk -F" " '{ $2 ~ "/wp-admin/install.php" }' $tempfile | awk '{print $1}' | sort | uniq -c | sort -r
19     # 404 for php files -mostly hacking attempts
20     awk '{ $9 ~ /404/ }' $tempfile | awk -F" " '{ $2 ~ "^GET.*.php" }' | awk '{print $7}' | sort | uniq -c | sort -r | head -n 20
21     #####
22     # Most requested URLs #####
23     awk -F" " '{print $2}' $tempfile | awk '{print $2}' | sort | uniq -c | sort -r
24     # Most requested URLs containing XYZ
25     awk -F" " '{ $2 ~ "ref" }' $tempfile | awk '{print $2}' | sort | uniq -c | sort -r
26
27     rm $tempfile
28 }
29 export -f pure_func
fi

710 if test "$NMDS5" = y; then
711     if test "$SQUIET" = "n"; then
712         echo "Skipping md5sum at user request"
713     fi
714 else
715     # Try to locate a MD5 binary
716     OLD_PATH=$PATH
717     PATH=${GUESS_MD5_PATH:-"$OLD_PATH:/bin:/usr/bin:/sbin:/usr/local/sbin:/usr/local/bin:/opt/openssl/bin"}
718     MDS_ARG=""
719     MDS_PATH=$(exec <&- 2>&-; which md5sum || command -v md5sum || type md5sum)
720     test -x "$MDS_PATH" || MDS_PATH=$(exec <&- 2>&-; which md5 || command -v md5 || type md5)
721     test -x "$MDS_PATH" || MDS_PATH=$(exec <&- 2>&-; which digest || command -v digest || type digest)
722
723     PATH=$OLD_PATH
724     if test -x "$MDS_PATH"; then
725         if test `basename "$MDS_PATH" `x = digestx; then
726             MDS_ARG="-a md5"
727
728             md5sum=$(eval "$MDS_PATH $MDS_ARG" < "$tempfile" | cut -b-32)
729             if test "$SQUIET" = "n"; then
730                 echo "MDS: $md5sum"
731             fi
732         else
733             if test "$SQUIET" = "n"; then
734                 echo "MDS: none, MDS command not found"
735             fi
736         fi
737     fi
738 fi
```

Expressing only a subset of the computation

h just the unique classes in y
fficients
lticlass \$penalty 1
lticlass \$penalty 2
lticlass \$penalty 3
lticlass \$penalty 4
lticlass \$penalty 5
lticlass \$penalty 6
lticlass \$penalty 7

```
60 $PYTHON $SCRIPTS/zip_coef.py $MODEL
61 $PYTHON $SCRIPTS/adjust_coef.py $MODEL $X $multiclass $n_classes $RESULT/trained_model.obj
```

Error-prone!

E.g., env variables, conditionals...

sc.textFile(log)

.filter(lambda l: p.search(l)...)

.map(lambda l: l.split(" ")[0])

.map(lambda key: (key, 1))

.reduceByKey(lambda a, b: a + b)

.sortBy(lambda kv: kv[1]...)



mapper.py

reducer.py

for l in stdin:

for l in sys.stdin:

if bots.search(l)... k, v = l.split(...)

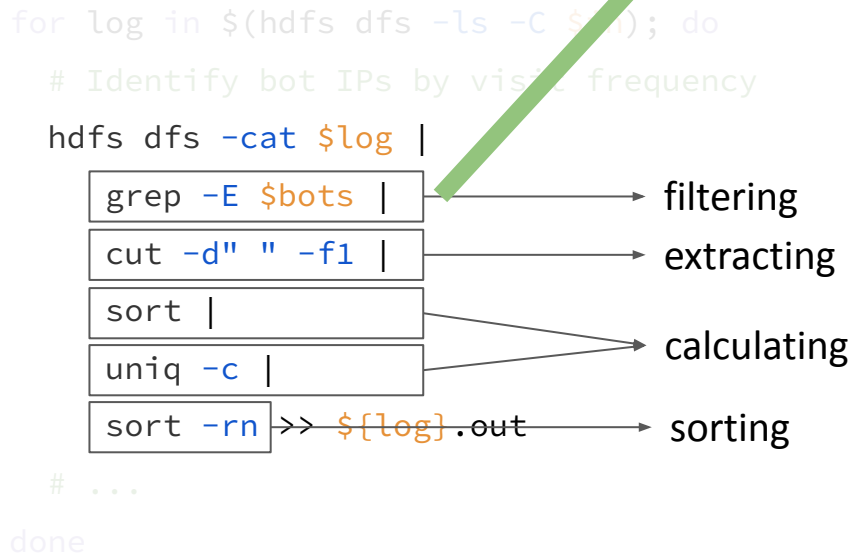


Distributing the Example Pipeline

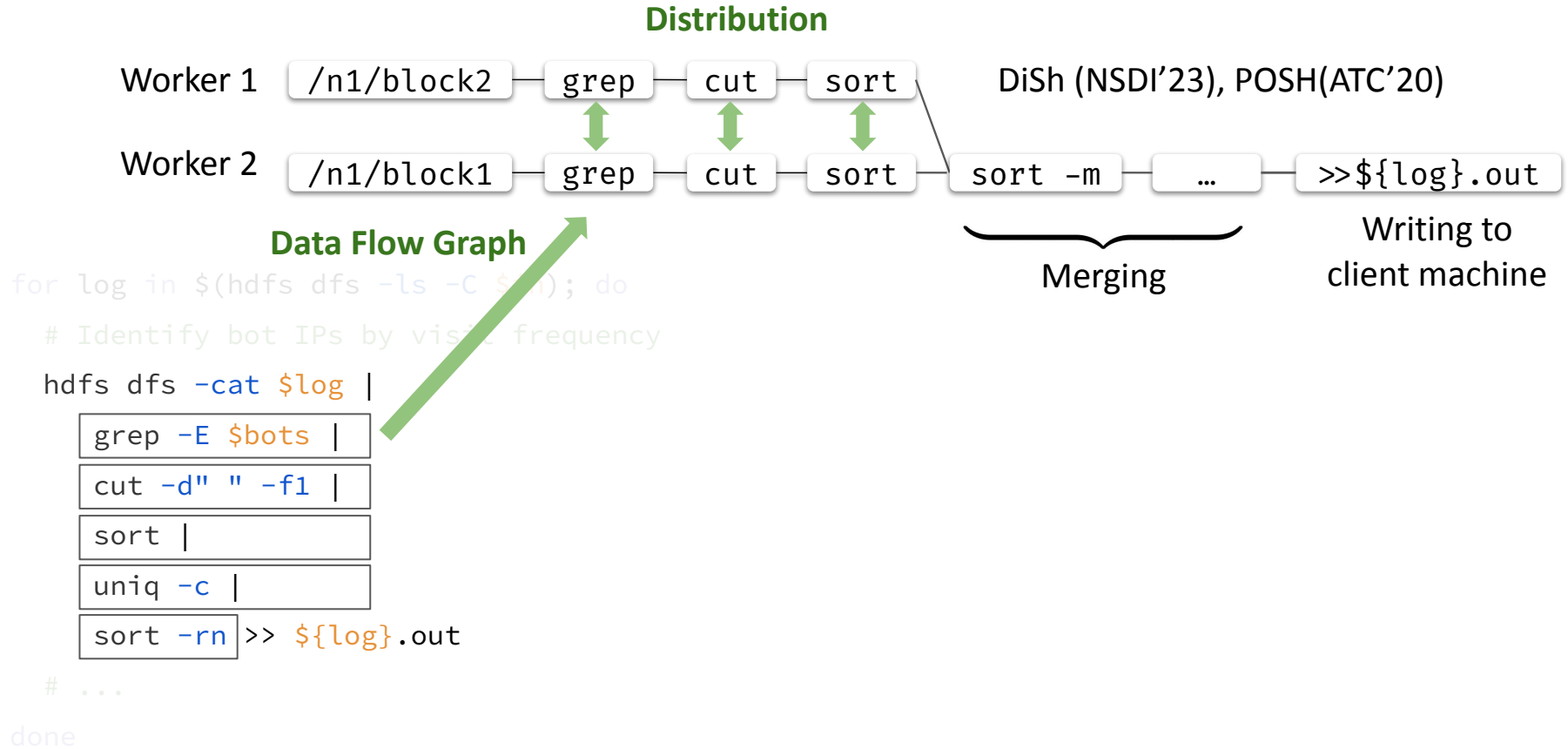
DiSh (NSDI'23), POSH(ATC'20)



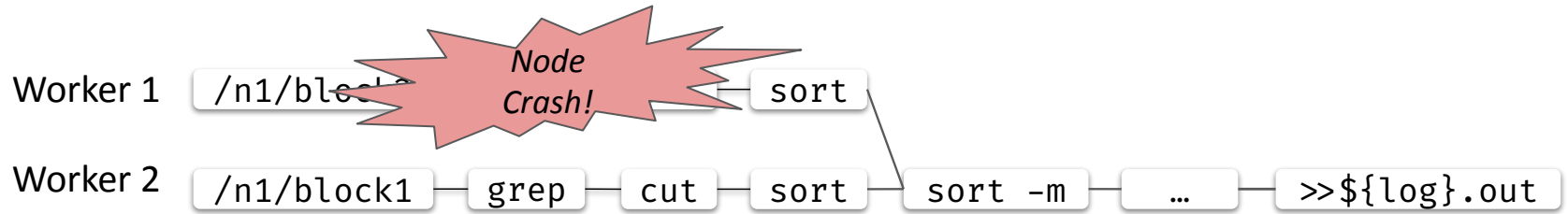
Data Flow Graph



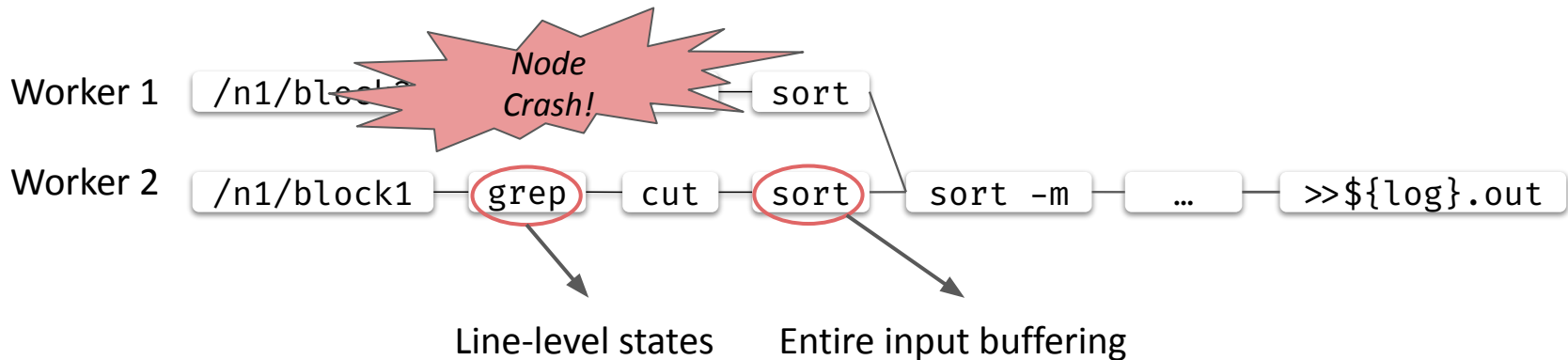
Distributing the Example Pipeline



Tolerating Faults in Distribution is Challenging



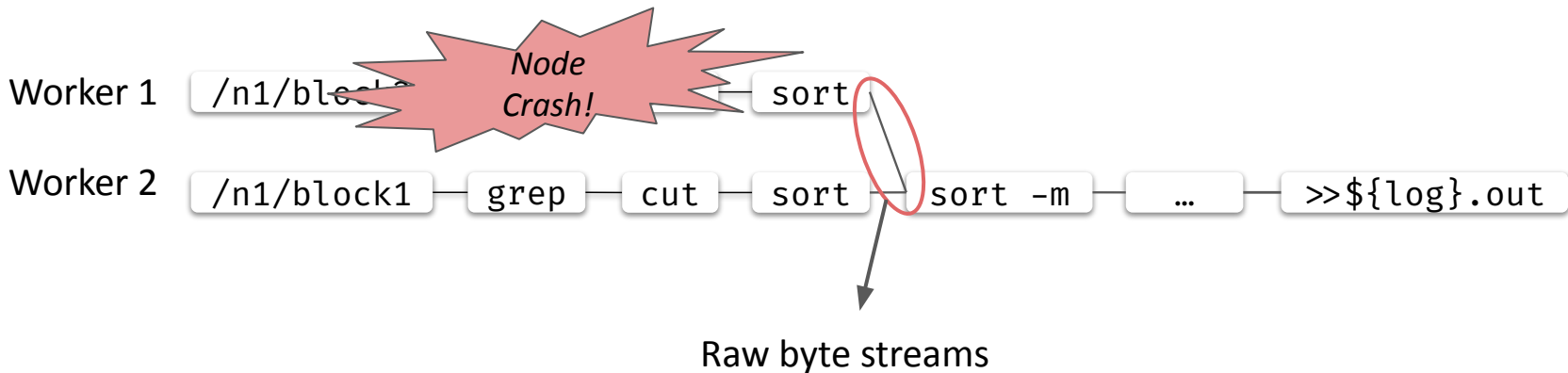
Checkpointing Opaque States is Hard



Difficult to checkpoint and recover from all states correctly

Idea 1: Checkpointing? E.g., Flink (VLDB'17), Naiad (SOSP'13)

No Barrier, No Stages, No Structured Data



Idea 2: Barrier-based? E.g., MapReduce, Hadoop Streaming

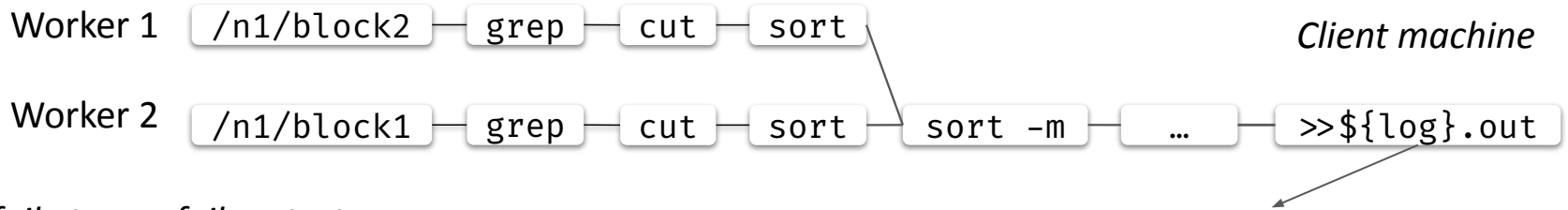
Idea 3: Lineage-based? E.g., Spark (NSDI'12), Ray (OSDI'18)

Requirement: either

- (1) barrier or stage, e.g., mapper, reducer
- (2) structured task or data, e.g., objects

***Fractal** is the first system to provide fault tolerance for opaque components in shell-program distribution.*

Fault Model and Goal



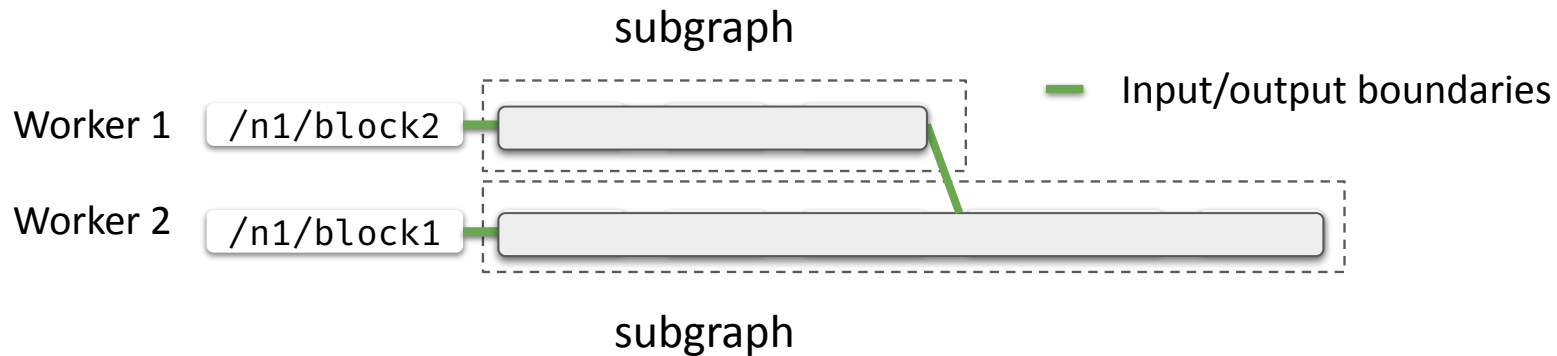
*Target: fail-stop or fail-restart
crashes of **worker nodes***

*Side-effectful, non-idempotent (e.g.,
interacting with client file system)...*

Distributed workload: deterministic and idempotent computation

Goal: the exact same behaviors (e.g., exit codes) and output (e.g., no duplicates or loss)

Key Insight: Opaque Subgraph Granularity

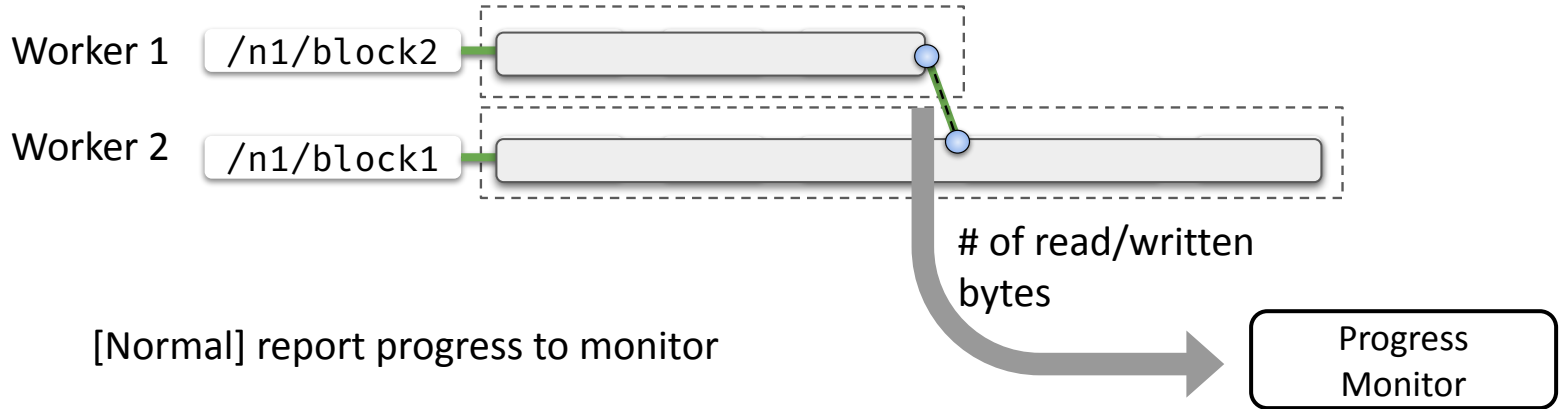


Fractal only tracks input and output for each *subgraph*.

Bolt-on Progress Tracking via Raw Bytes



Remote pipes: drop-in replacement for Unix pipes



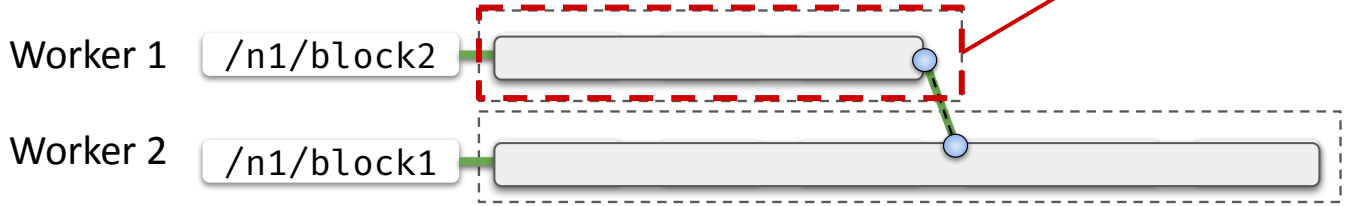
[Normal] report progress to monitor

[Recovery] detect and discard duplicates

Fast Recovery via Dynamic Output Persistence



Persistence? Disk capacity? Input size?



Recovering from Faults with Different Strategies

Regular subgraph: operates on individual data blocks



Merger subgraph: merges multiple results from regular subgraphs

Input for recovering regular faults: the same input data blocks

Input for recovering merger faults: persisted results or re-execution of upstream subgraphs

Evaluation

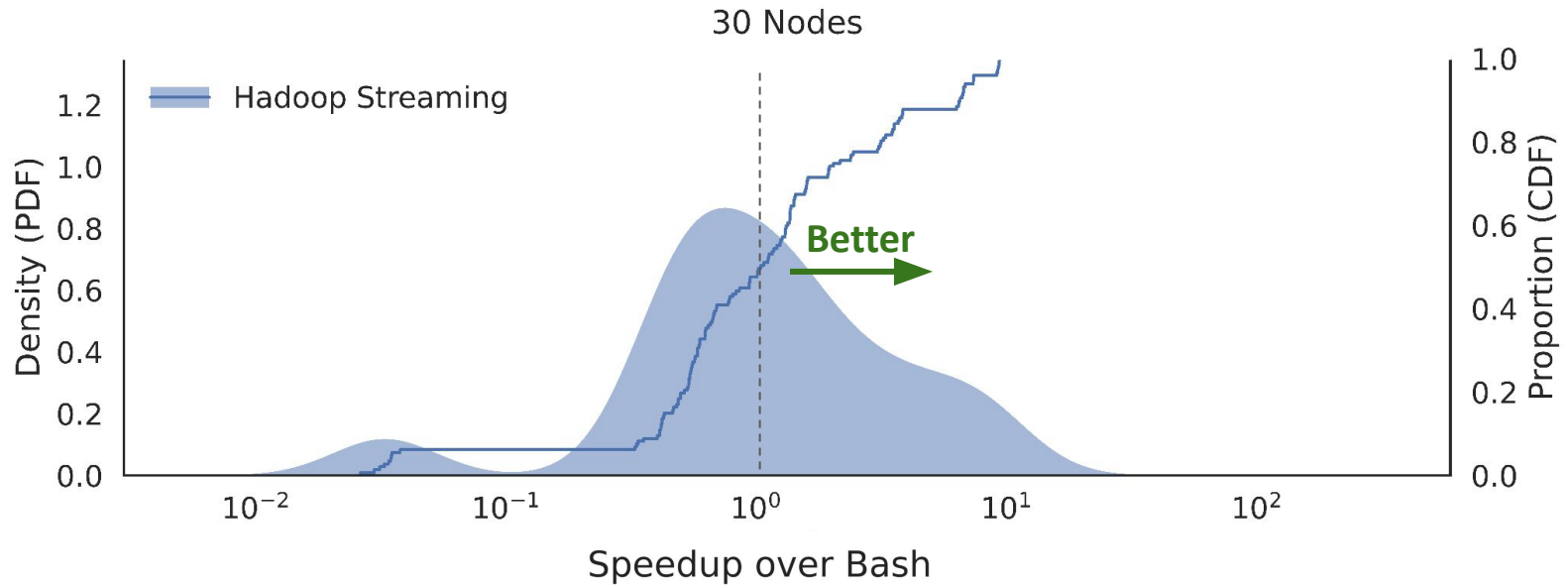
How fast is Fractal under fault-free execution and fault recovery?

Evaluation

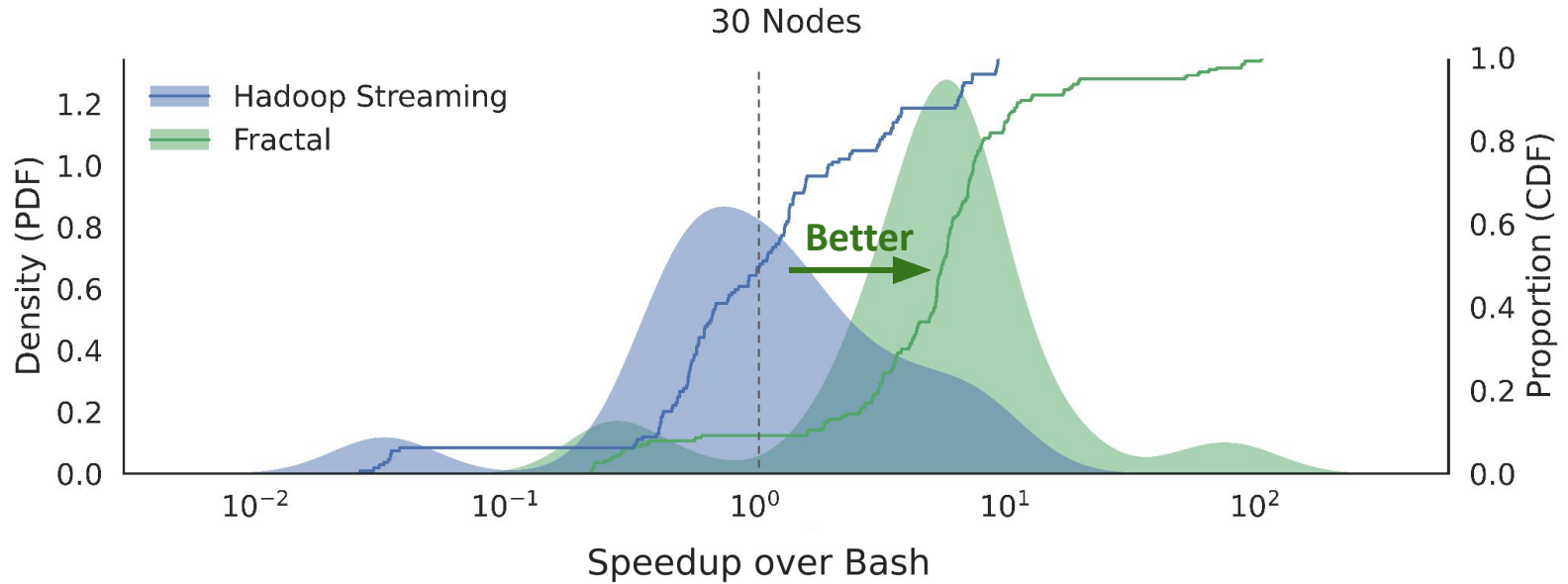
- Benchmarks
 - 5 Benchmarks from the Koala benchmark suite^[1]
 - Totaling 77 scripts and up to 33.4 GB input.
- Baselines
 - Bash: the standard shell interpreter
 - DiSh (NSDI'23): a state-of-the-art shell distribution system
 - Apache Hadoop Streaming (AHS): a MapReduce system that offers fault-tolerant distributed execution of arbitrary binaries
- *frac*: fault injection subsystem

[1] Evangelos Lamprou, Ethan Williams, Georgios Kaoukis, Zhuoxuan Zhang, Michael Greenberg, Konstantinos Kallas, Lukas Lazarek, and Nikos Vasilakis. "The Koala benchmarks for the shell: characterization and implications." USENIX ATC. 2025.

Evaluation — Fault-free Performance

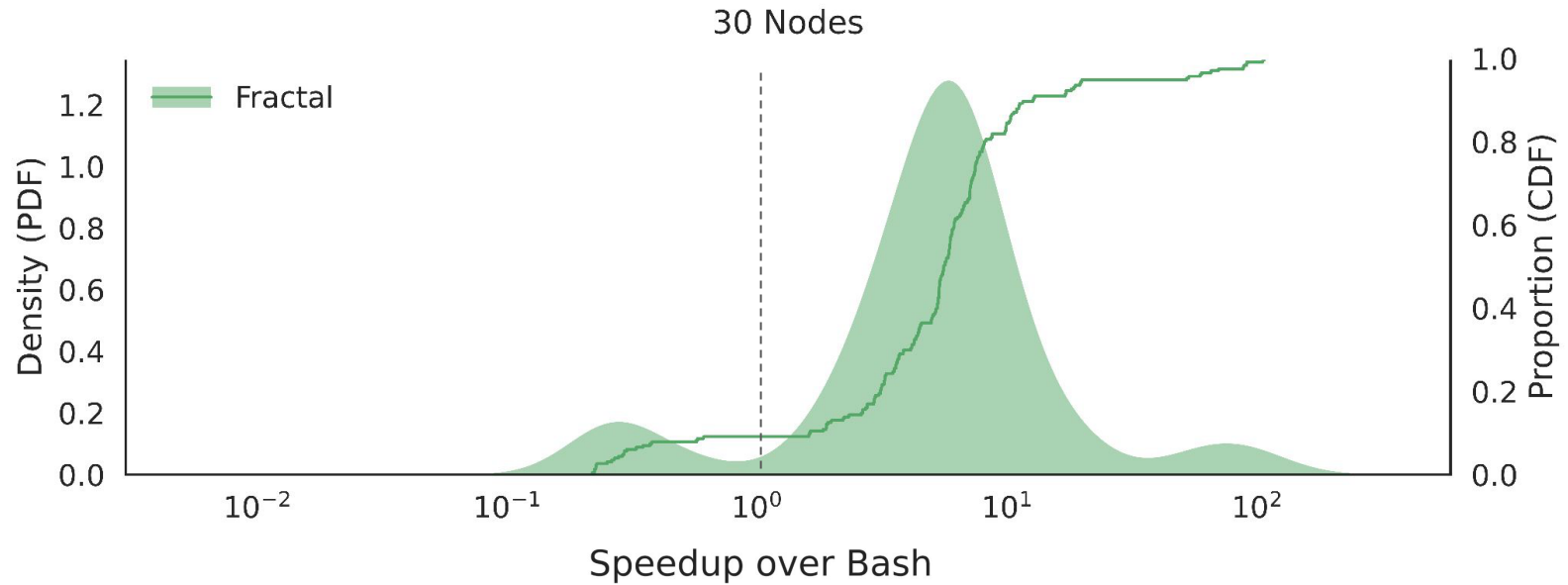


Evaluation — Fault-free Performance

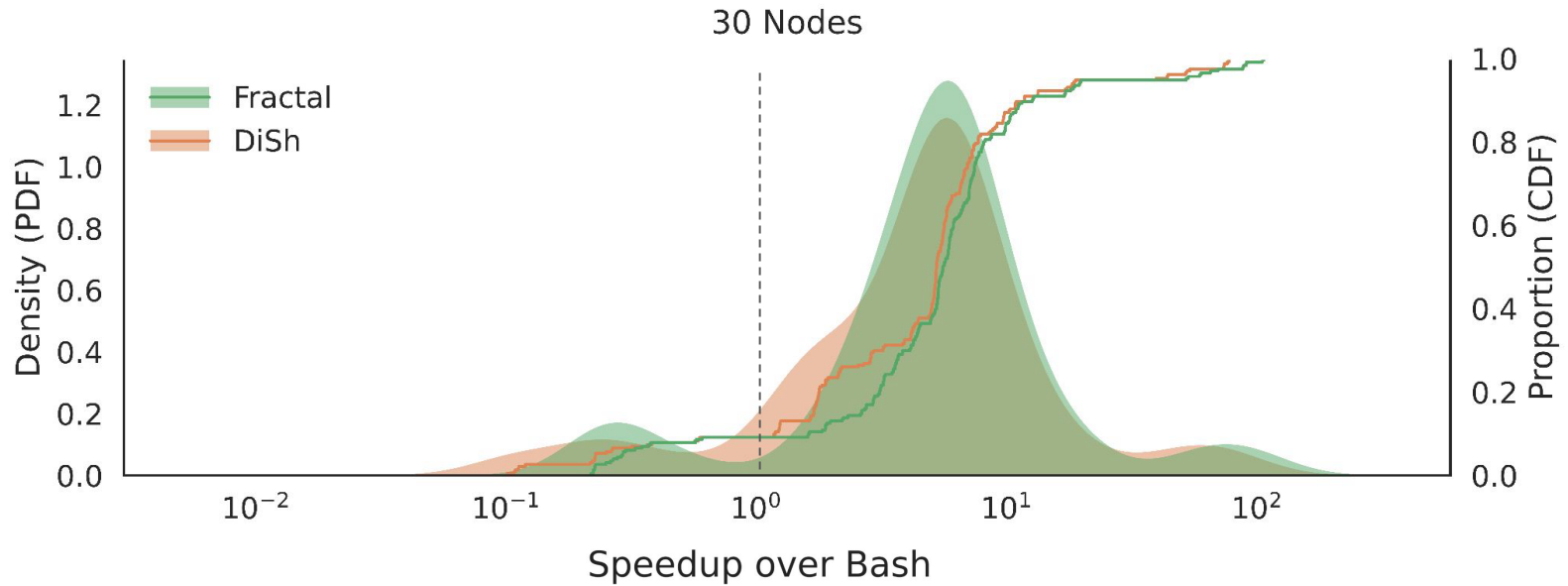


Fractal: max — **107.84x** Hadoop Streaming: max — 9.48x

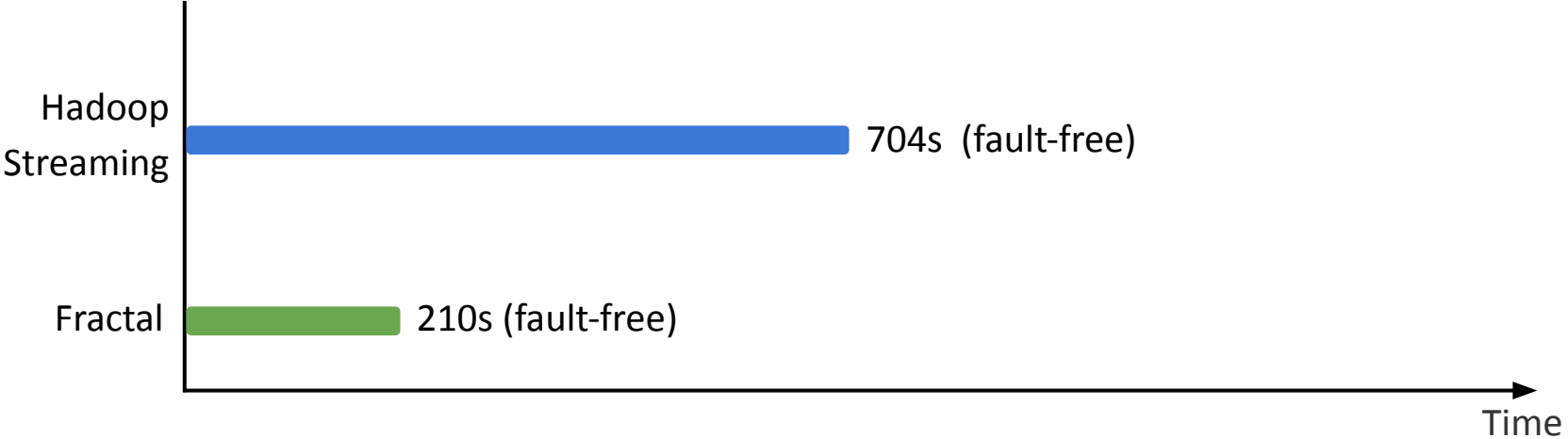
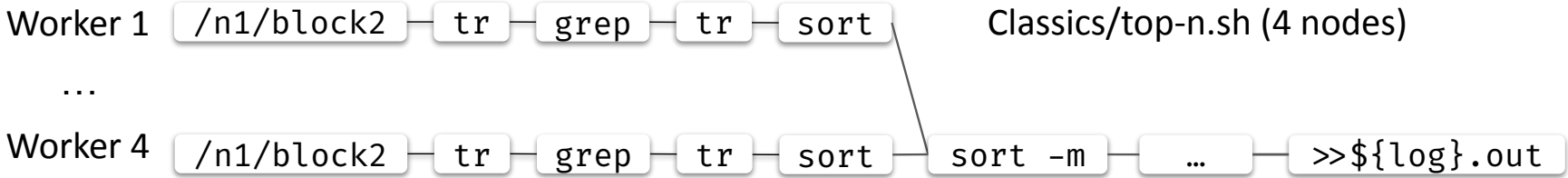
Evaluation — Fault-free Performance



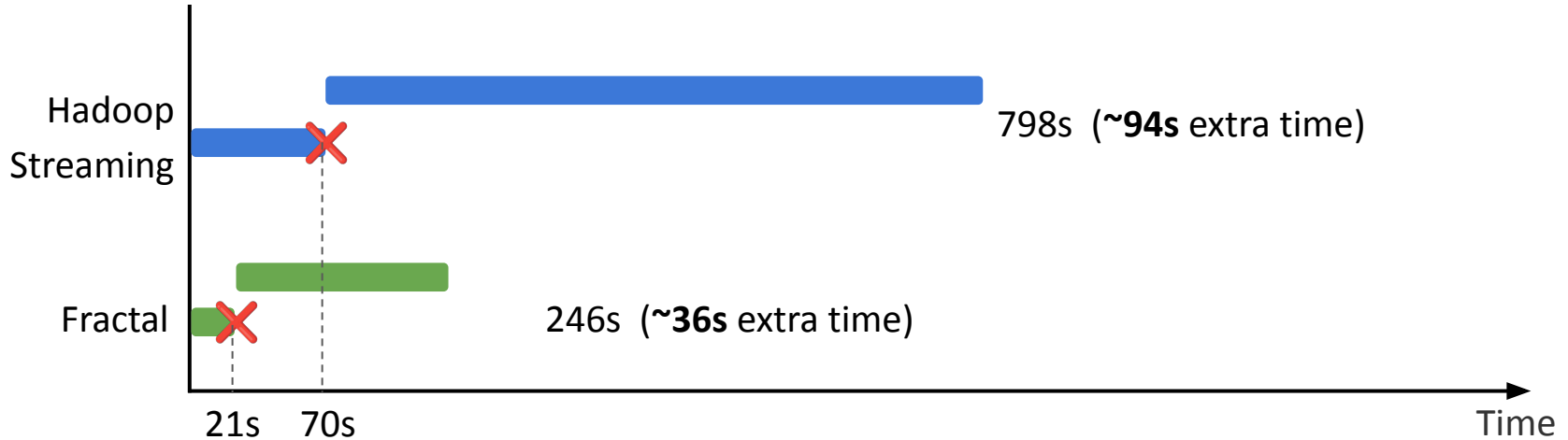
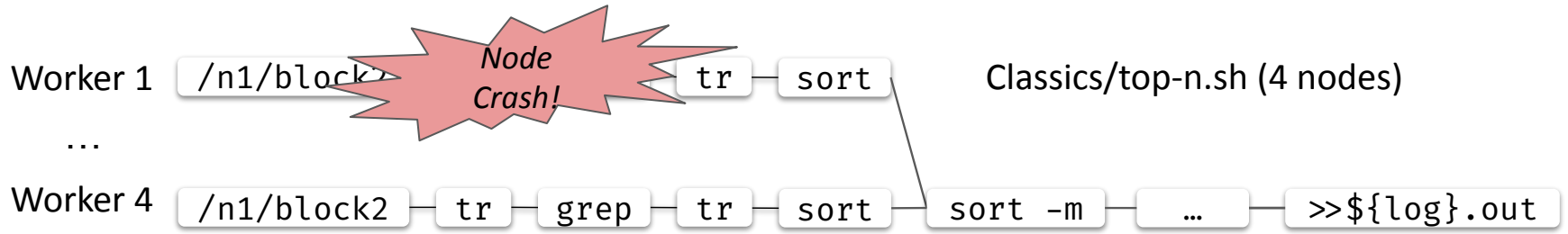
Evaluation — Fault-free Performance



Evaluation — Fault Recovery Performance

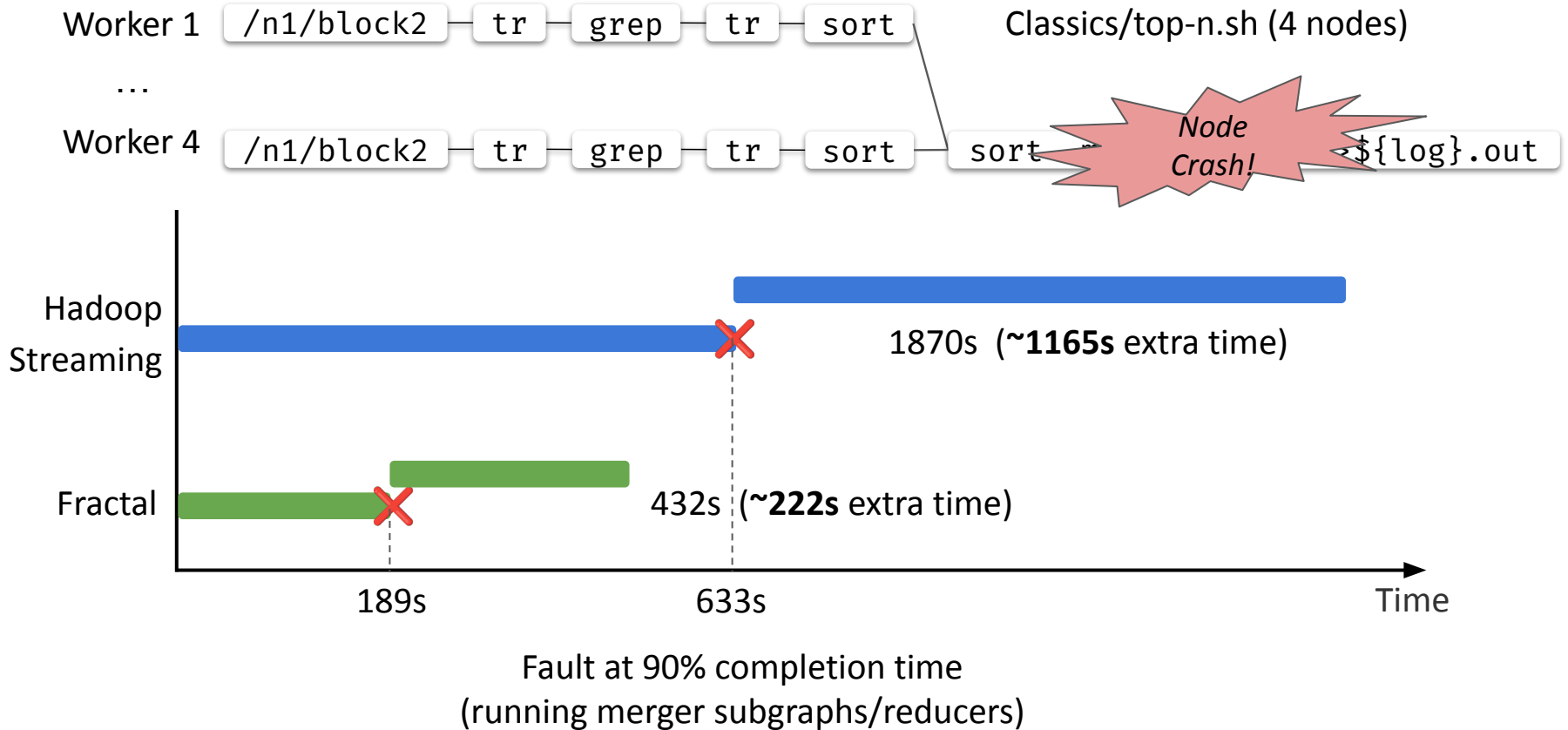


Evaluation — Fault Recovery Performance

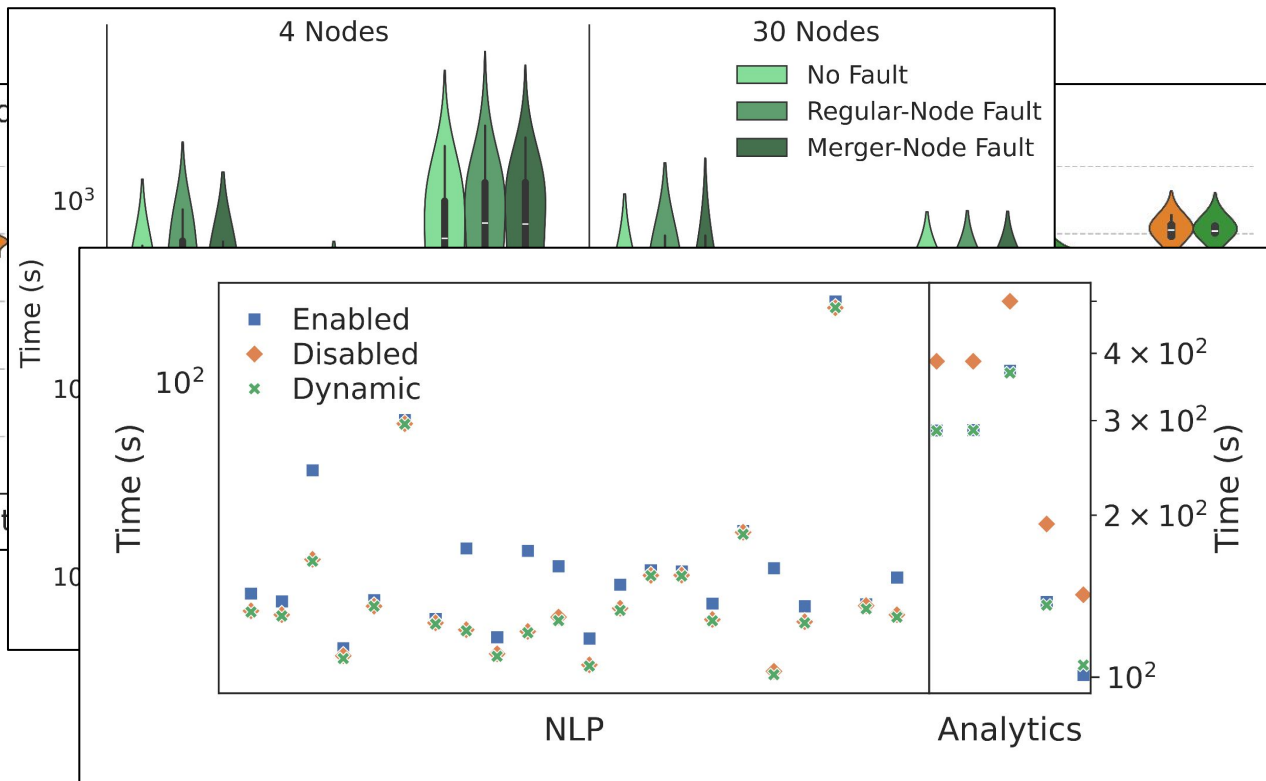
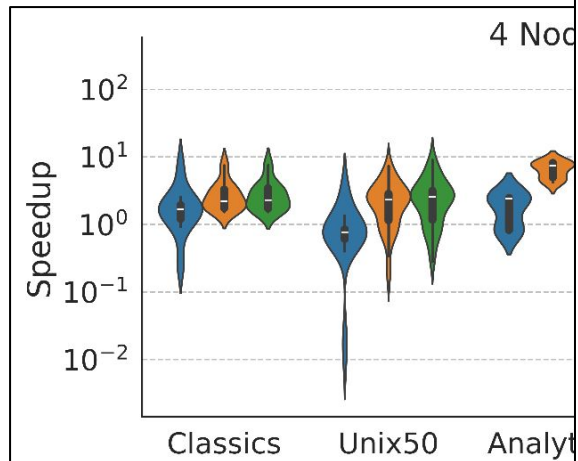


Fault at 10% completion time
(running regular subgraphs/mappers)

Evaluation — Fault Recovery Performance



More in the paper



- Fractal design
 - Runtime optimizations
- Evaluation
 - Fine-grained performance and recovery analysis
 - ...and more experiments!



Fractal: Fault-Tolerant Shell-Script Distribution

It offers:

- An execution engine to distribute unmodified Shell scripts with fault tolerance.
- Performance optimizations to improve efficiency at runtime.
- A fault injection subsystem to enable large-scale characterization of fault recovery.

Zhicheng Huang*, Ramiz Dundar*, **Yizheng Xie**,
Konstantinos Kallas^{UCLA}, Nikos Vasilakis



 github.com/binpash/fractal

