

DATE
17/06/2024

EP-SFT Group Meeting

RESEARCH OF THE SCHEDULING MECHANISMS IN JULIA DAGGER LIBRARY

Summary

PRESENTED BY

Oleksandr Shchur
(Ukrainian Catholic University,
Ukrainian Remote Student Program)

SUPERVISOR

Benedikt Hegner

CO-SUPERVISOR

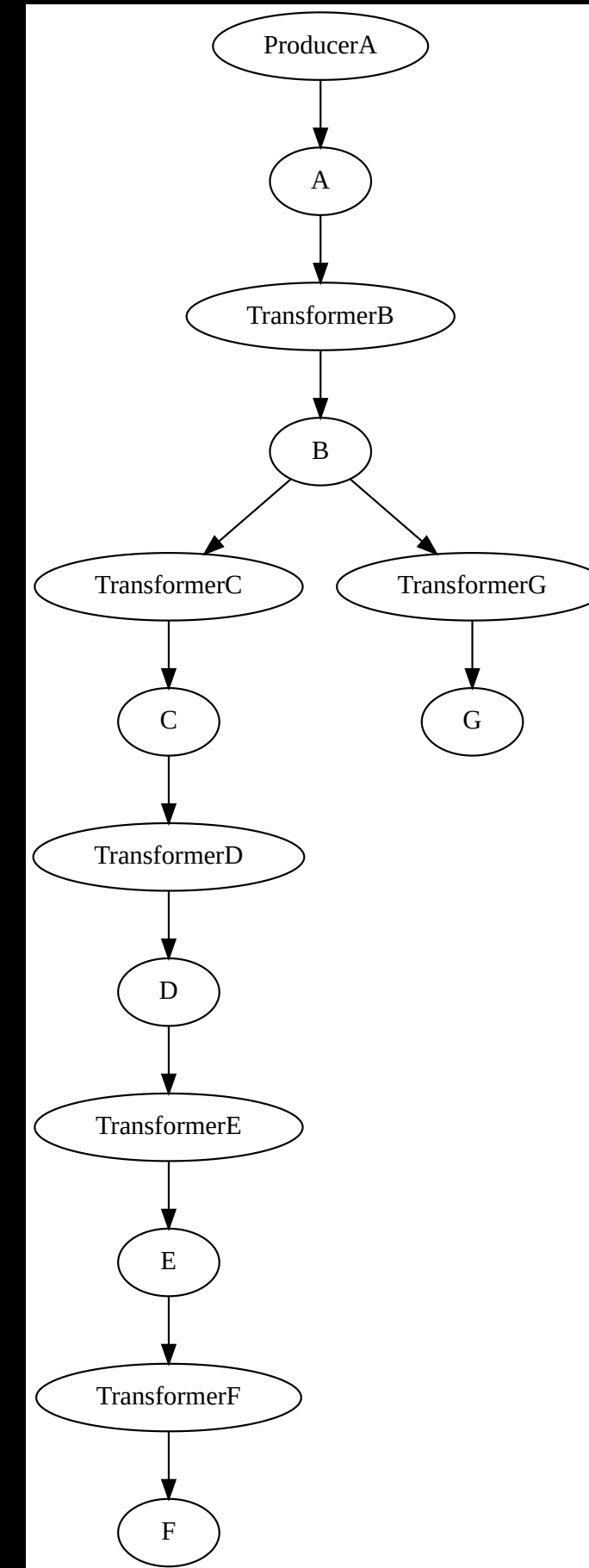
Mateusz Jakub Fila



Introduction

What is this project about?

The workloads of modern data processing frameworks used by LHC experiments can be very often described as directed acyclic graphs of tasks.



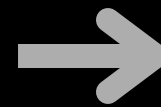
Objectives

- Parse a directed acyclic graph (DAG) representing the tasks workflow
- Schedule the tasks parallelly with **Dagger**
- Get scheduler events and metrics logged
- Display the logging data using meaningful representations

Parsing DAG

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <graphml xmlns="http://graphml.graphdrawing.org/xmlns" xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
3 <key id="key0" for="node" attr.name="class" attr.type="string" />
4 <key id="key1" for="node" attr.name="node_id" attr.type="string" />
5 <key id="key2" for="node" attr.name="type" attr.type="string" />
6 <graph id="G" edgedefault="directed" parse.nodeids="free" parse.edgeids="canonical" parse.order="no
7 <node id="n0">
8   <data key="key0">MicroProducer</data>
9   <data key="key1">ProducerA</data>
10  <data key="key2">GaudiAlgorithm</data>
11 </node>
12 <node id="n1">
13   <data key="key0"></data>
14   <data key="key1">A</data>
15   <data key="key2">DataObject</data>
16 </node>
17 <node id="n2">
18   <data key="key0">MicroTransformer</data>
19   <data key="key1">TransformerB</data>
20   <data key="key2">GaudiAlgorithm</data>
21 </node>
22 <node id="n3">
23   <data key="key0"></data>
24   <data key="key1">B</data>
25   <data key="key2">DataObject</data>
26 </node>
27 <node id="n4">
```

.graphml file examples

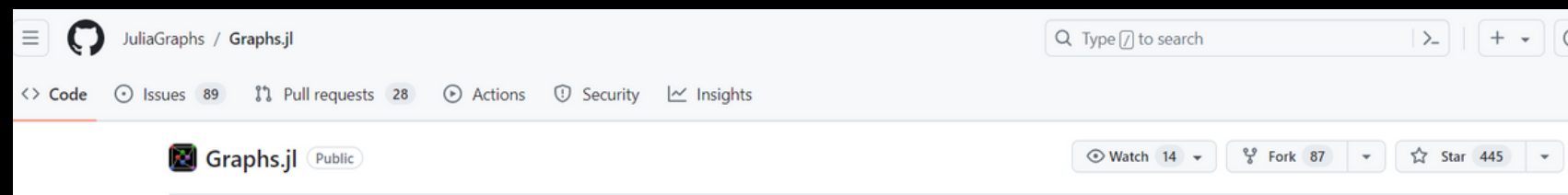


```
1 mutable struct MetaDiGraph{T<:Integer,U<:Real} <: AbstractMetaGraph{T}
2   graph::SimpleDiGraph{T}
3   vprops::Dict{T,PropDict}
4   eprops::Dict{SimpleEdge{T},PropDict}
5   gprops::PropDict
6   weightfield::Symbol
7   defaultweight::U
8   metaindex::MetaDict
9   indices::Set{Symbol}
10 end
```

MetaDiGraph structure

Parsing DAG: pitfalls

Popular Julia package for graphs processing: **JuliaGraphs** organization packages:



Related packages

It is an explicit design decision that any data not required for graph manipulation (attributes and other information, for example) is expected to be stored outside of the graph structure itself.

Additional functionality like advanced IO and file formats, weighted graphs, property graphs, and optimization-related functions can be found in the packages of the [JuliaGraphs organization](#).

Packages

Algorithms and graph types

The central package of the ecosystem is [Graphs.jl](#). It contains a standard graph interface and some basic types for unweighted graphs, as well as a set of combinatorial algorithms like shortest paths.

Many of the other packages we list rely only on this interface, so as to be compatible with arbitrary graph types.

More graph types

- [SimpleWeightedGraphs.jl](#): Simple graphs with weighted edges.
- [MetaGraphs.jl](#): Graphs with metadata on the vertices and edges.
- [MetaGraphsNext.jl](#): A more efficient but less flexible alternative to [MetaGraphs.jl](#).
- [StaticGraphs.jl](#): Memory-efficient immutable graphs.
- [MultilayerGraphs.jl](#): Graphs with multiple layers.

Parsing DAG: metagraphs

Both **MetaGraphs.jl** and **MetaGraphsNext.jl** do not support loading graphs from **.graphml**

The screenshot shows the documentation for `MetaGraphsNext.jl` under the "File storage" section. It includes a search bar, a sidebar with navigation links (Home, Tutorial, Basics, Graphs.jl interface, File storage, MGFormat, DOTFormat, Type stability, Benchmark, API reference), and a main content area. The "File storage" section contains a code block with the following content:

```
using Graphs
using MetaGraphsNext
```

The "MGFormat" section explains that `MetaGraphsNext.jl` overloads `Graphs.savegraph` to write graphs in a custom format called `MGFormat`, which is based on `JLD2`. It is not very readable but gives the right result when loaded back. A code block shows an example of saving and loading a graph:

```
example = MetaGraph(Graph(), Symbol);

example2 = mktemp() do file, io
    savegraph(file, example)
    loadgraph(file, "something", MGFormat())
end

example2 == example
```

The result of the execution is shown as `true`.

The "DOTFormat" section mentions that `MetaGraphsNext.jl` also supports the more standard `DOT` encoding, which is used as follows.

The screenshot shows the documentation for `MetaGraphs` under the "Overview" section. It includes a search bar, a sidebar with navigation links (Overview, Getting started, Example use, MetaGraphs Functions, License Information), and a main content area. The "Overview" section contains the following text:

MetaGraphs

docs stable docs dev CI passing codecov unknown

A flexible package for graphs with arbitrary metadata.

For a more performant option, check out [MetaGraphsNext.jl](#)

Getting started

Installation is straightforward: from the Julia `pkg` prompt,

```
pkg> add MetaGraphs
```

```
2
3     struct MGFormat <: AbstractGraphFormat end
4     struct DOTFormat <: AbstractGraphFormat end
5
```

Parsing DAG: GraphIO

JuliaGraphs also mentions:

Interfaces and visualization

Input / Output

- [GraphIO.jl](#): Read graphs from files and write them to files in various formats.
- [SNAPDatasets.jl](#): Extract graphs from the SNAP Datasets collection.

But it does not parse metadata...


The screenshot shows the GitHub repository for GraphIO.jl. The repository name is 'GraphIO.jl' and it is public. It has 5 watchers, 26 forks, and 61 stars. The README section is visible, showing the title 'GraphIO' and a table of supported graph formats. The table has columns for Format, Read, Write, Multiple Graphs, Format Name, and Comment. The 'GraphML' row is highlighted with a red underline.

GraphIO provides support to [Graphs.jl](#) for reading/writing graphs in various formats.


Currently, the following functionality is provided:


Format	Read	Write	Multiple Graphs	Format Name	Comment
EdgeList	✓	✓		EdgeListFormat	a simple list of sources and dests separated by whitespace and/or comma, one pair per line.
GML	✓	✓	✓	GMLFormat	
Graph6	✓	✓	✓	Graph6Format	
GraphML	✓	✓	✓	GraphMLFormat	
Pajek NET	✓	✓		NETFormat	
GEXF		✓		GEXFFormat	
DOT	✓	✓	✓	DOTFormat	


Parsing DAG: GraphMLReader

 zhangliye / GraphMLReader.jl

[Code](#) [Issues 1](#) [Pull requests 1](#) [Actions](#)

 **GraphMLReader.jl** Public

 **1 Open** ✓ **1 Closed**

 **Fixes**

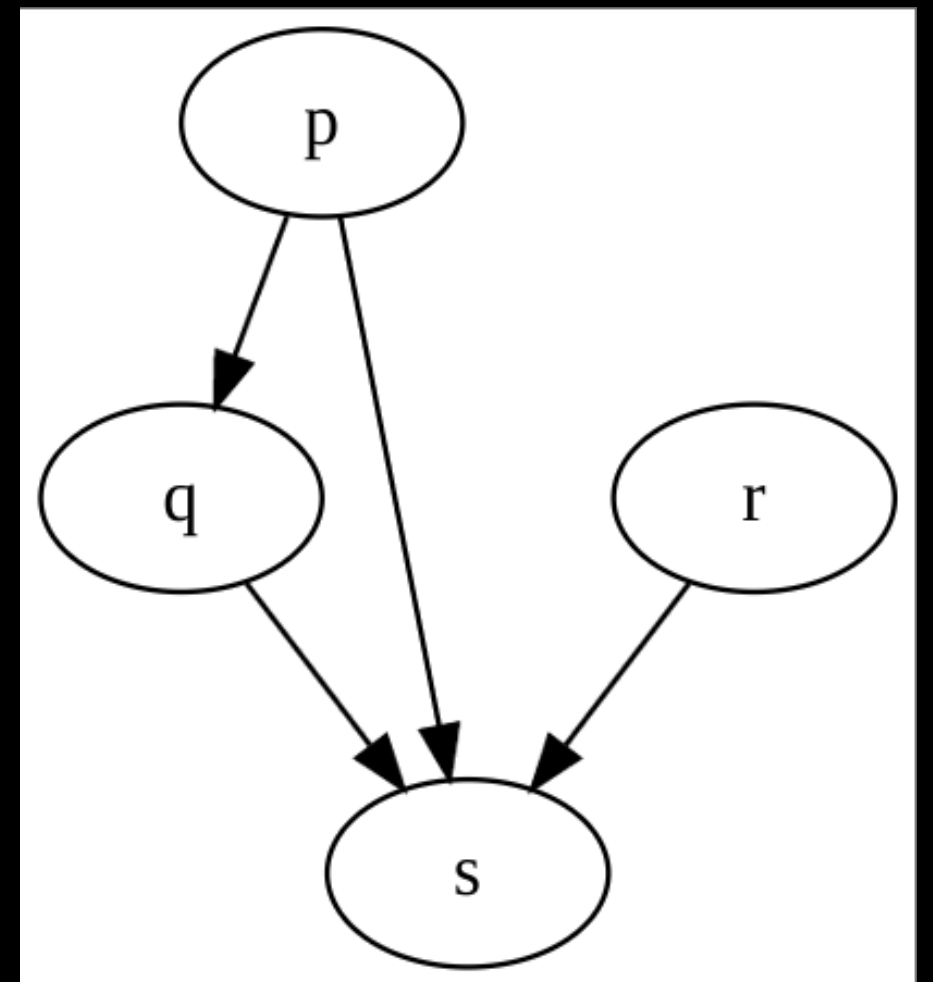
#3 opened on Mar 26 by SmalRat

Dagger: purpose



DAGGER

Essentially, **Dagger** is a scheduler, which “can run computations represented as directed-acyclic-graphs (DAGs) efficiently on many Julia worker processes and threads” (from Github readme)



Example from the
Dagger documentation

Dagger

Objectives:

- Schedule tasks DAG (data dependencies only).
- Tasks are simple mockups with the minimal data exchange
- Set the limit of concurrently running DAGs
- Re-run tasks DAG without the creation overhead

Dagger: API

Old Dagger API

```
function oldAPI_graph_setup(time_to_sleep)
  a = Dagger.delayed(taskA)(time_to_sleep, 1)
  b = Dagger.delayed(taskB)(time_to_sleep, 2, a)
  c = Dagger.delayed(taskC)(time_to_sleep, 3, a, b)
  d = Dagger.delayed(taskD)(time_to_sleep, 4, a, b, c)
  e = Dagger.delayed(taskE)(time_to_sleep, 5, a)
  f = Dagger.delayed(taskF)(time_to_sleep, 6, a, b, c, d, e)

  return f
end

ctx = Dagger.Sch.eager_context()
scheduled_graph = oldAPI_graph_setup(1)
result_reference = compute(ctx, scheduled_graph)
res = collect(ctx, result_reference)
```

Static predefined DAG execution

Modern Dagger API

```
function modernAPI_graph_setup(time_to_sleep)
  a = Dagger.@spawn (taskA)(time_to_sleep, 1)
  b = Dagger.@spawn (taskB)(time_to_sleep, 2, a)
  c = Dagger.@spawn (taskC)(time_to_sleep, 3, a, b)
  d = Dagger.@spawn (taskD)(time_to_sleep, 4, a, b, c)
  e = Dagger.@spawn (taskE)(time_to_sleep, 5, a)
  f = Dagger.@spawn (taskF)(time_to_sleep, 6, a, b, c, d, e)

  return f
end

scheduled_graph = modernAPI_graph_setup(1)
res = fetch(scheduled_graph)
```

Allows for the dynamic changes

Dagger: logging (v 0.18.8)

DaggerWebDash

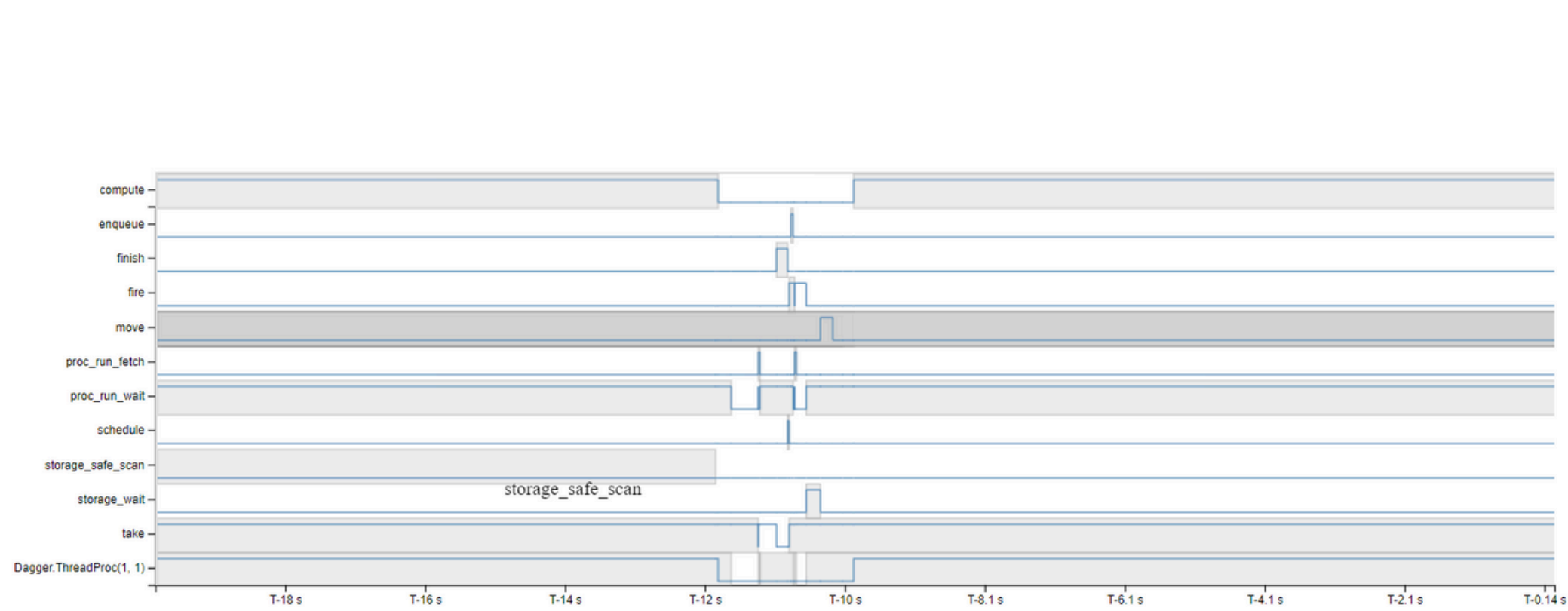
Raw logs processing



```
TERMINAL julialauncher + - - - - -
TimespanLogging.Timespan[TimespanLogging.Timespan(:init_proc, (worker = 1,), nothing, 0x0001c6bd382ccc94, 0x0001c6bd382f2624, Base.GC.Diff(13968, 0, 0, 123, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], TimespanLogging.Timespan(:init_proc, (worker = 1,), nothing, 0x0001c6bd67de29d8, 0x0001c6bd67e0f190, Base.GC.Diff(13872, 0, 0, 121, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], TimespanLogging.Timespan(:scheduler_init, nothing, OSProc(1), 0x0001c6bca991e17c, 0x0001c6bd67e7b7f0, Base.GC.Diff(72368704, 0, 0, 1010917, 1085, 0, 124564400, 1, 1), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], TimespanLogging.Timespan(:schedule, (thunk_id = 10,), (thunk_id = 10,), 0x0001c6bd67ea73f0, 0x0001c6bd67eeab3c, Base.GC.Diff(12752, 0, 0, 164, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], TimespanLogging.Timespan(:enqueue, (processor = Dagger.ThreadProc(1, 1), thunk_id = 10), nothing, 0x0001c6bd6a3e68c8, 0x0001c6bd6a4058e0, Base.GC.Diff(6768, 0, 0, 61, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], TimespanLogging.Timespan(:proc_run_wait, (worker = 1, processor = Dagger.ThreadProc(1, 1)), nothing, 0x0001c6bd6a428f48, 0x0001c6bd6a437d90, Base.GC.Diff(6352, 0, 0, 53, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], TimespanLogging.Timespan(:proc_run_fetch, (worker = 1, processor = Dagger.ThreadProc(1, 1)), (thunk_id = 10, proc_occupancy = 0x00000000, task_occupancy = 0xffffffff), 0x0001c6bd6a450624, 0x0001c6bd6a45d39c, Base.GC.Diff(6528, 0, 0, 60, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], TimespanLogging.Timespan(:fire, (worker = 1,), nothing, 0x0001c6bd67f627a4, 0x0001c6bd6a480874, Base.GC.Diff(350368, 0, 0, 4387, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], TimespanLogging.Timespan(:storage_wait, (thunk_id = 10, processor = Dagger.ThreadProc(1, 1)), (f = taskA, device = MemPool.CPURAMDevice), 0x0001c6bd6a4a933c, 0x0001c6bd6a4c8ea8, Base.GC.Diff(7440, 0, 0, 75, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], TimespanLogging.Timespan(:move, (thunk_id = 10, id = 0, processor = Dagger.ThreadProc(1, 1)), (f = taskA, data = taskA), 0x0001c6bd6a4e29fc, 0x0001c6bd6af3164c, Base.GC.Diff(34056, 0, 0, 428, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[0x000002034a91bc20])), TimespanLogging.Timespan(:move, (thunk_id = 10, id = -1, processor = Dagger.ThreadProc(1, 1)), (f = taskA, data = 1), 0x0001c6bd6af497d8, 0x0001c6bd6bca8960, Base.GC.Diff(33864, 0, 0, 426, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[0x000002034a91be10])), TimespanLogging.Timespan(:move, (thunk_id = 10, id = -2, processor = Dagger.ThreadProc(1, 1)), (f = taskA, data = 1), 0x0001c6bd6bcc19c4, 0x0001c6bd6bcd3a70, Base.GC.Diff(6352, 0, 0, 55, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[0x000002034a91be10]))]
```

DaggerWebDash: metrics

Overview



Status: *Connected*

Online?

Update?

Draw Interval: 1

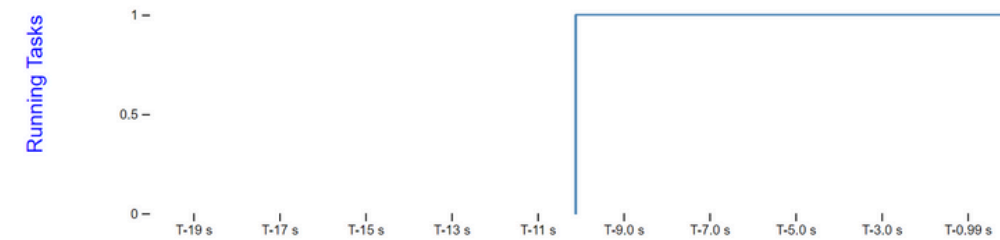
Worker: 1

Zoom: Zoom In Zoom Out

Seek: Prev Next Full

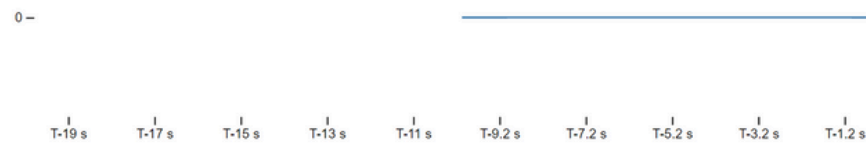
Seek Window: 20

Worker Saturation



CPU Load Average

Average Running Threads



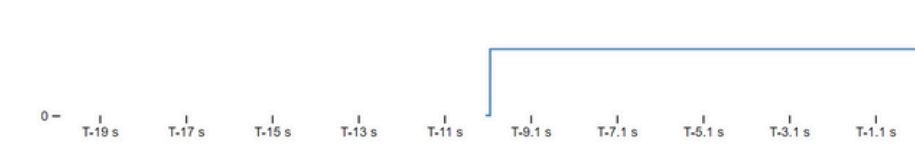
Available Memory

% Free

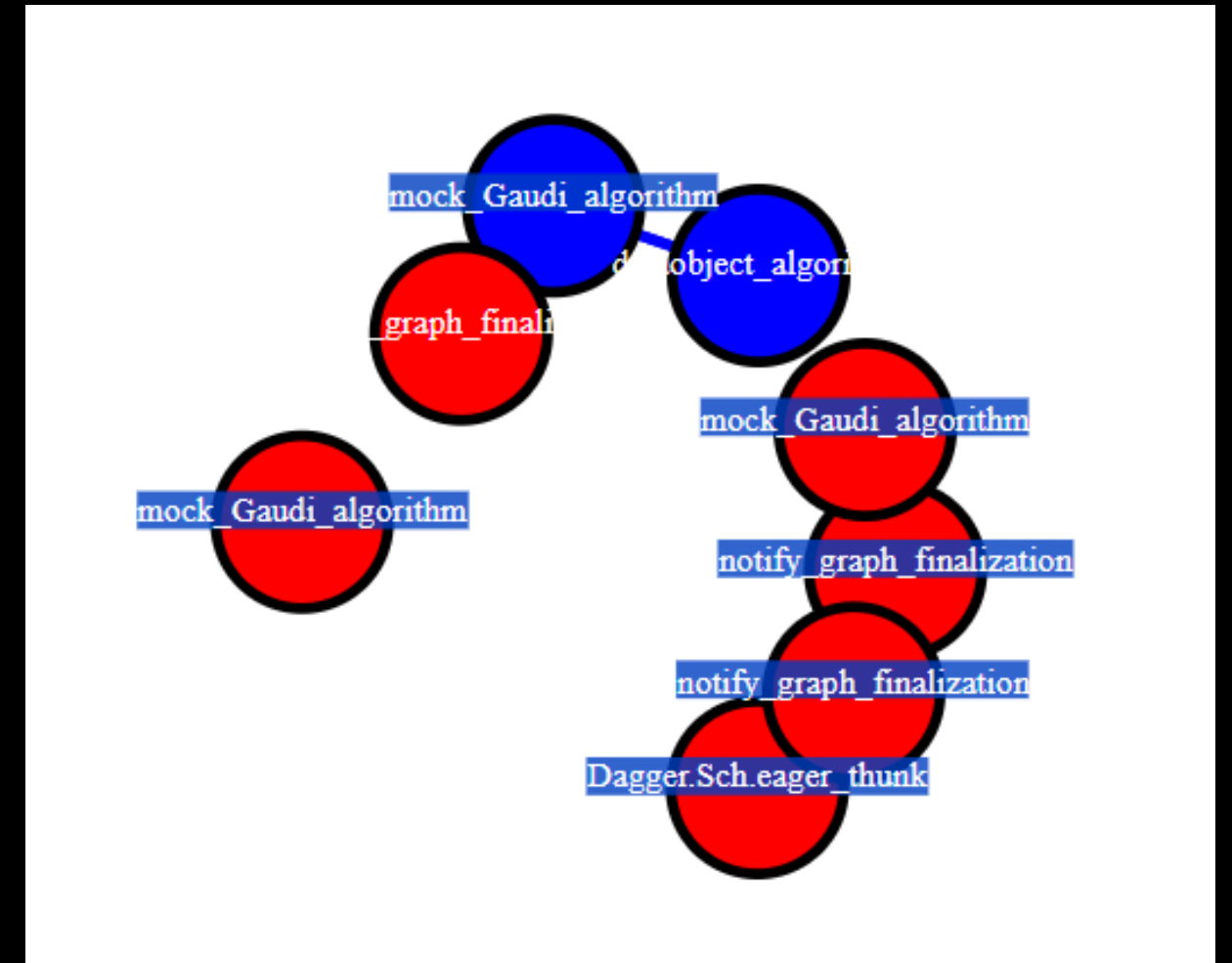
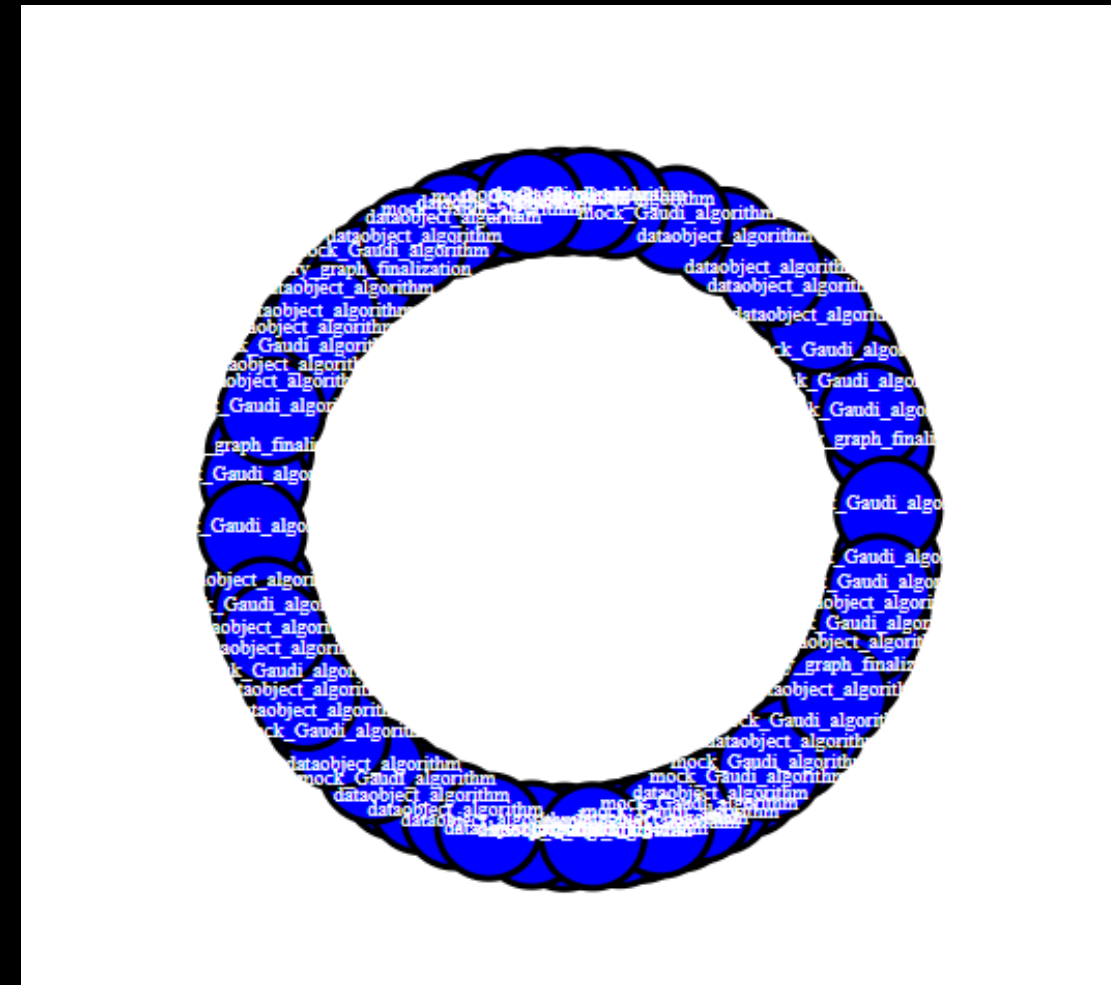
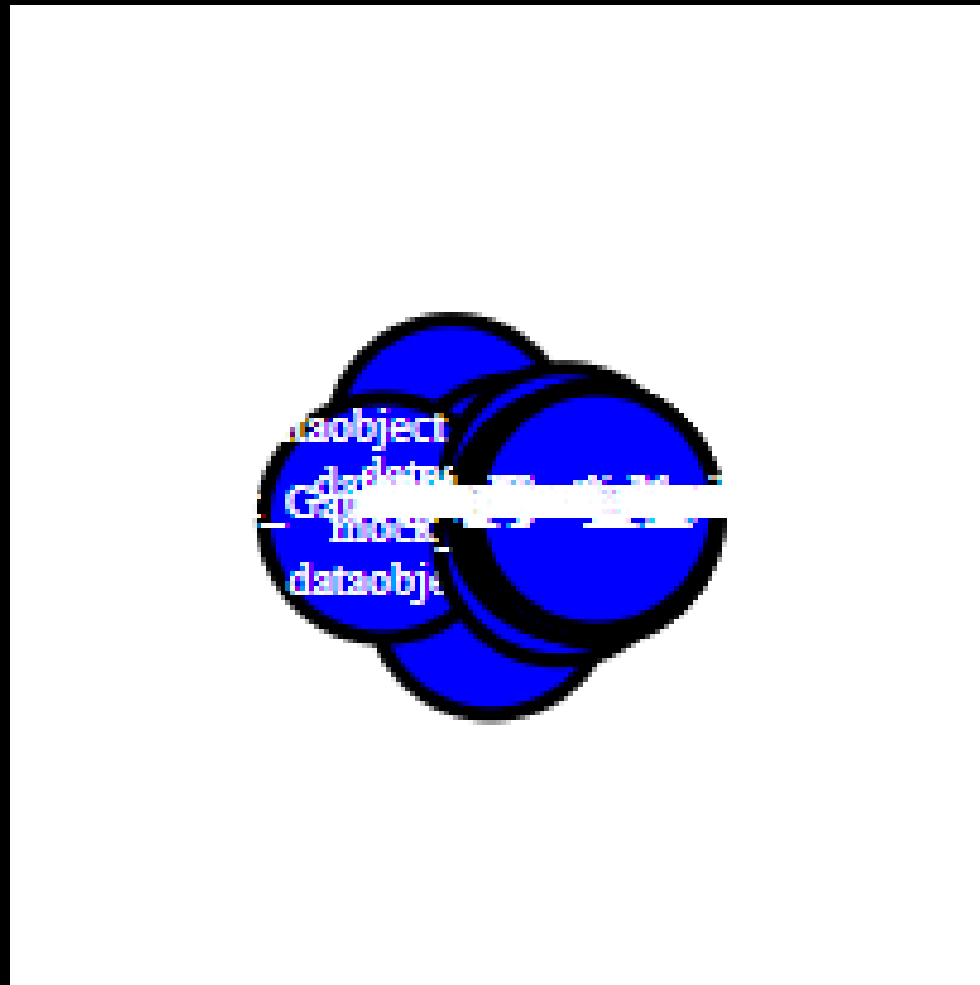


Allocated Bytes

Bytes



DaggerWebDash: DAGs



DaggerWebDash: errors

When using worker processes

```
From worker 5: | @ TimespanLogging C:\Users\Admin\
.julia\packages\TimespanLogging\DsD30\src\core.jl:294
From worker 5: | [19] (::Dagger.Sch.var"#138#146
"{Dagger.Sch.ProcessorInternalState, UInt64, RemoteChannel{Ch
annel{Any}}, Dagger.ThreadProc})()
From worker 5: | @ Dagger.Sch C:\Users\Admin\
.julia\packages\Dagger\Tx54v\src\sch\Sch.jl:1197
From worker 5: | @ TimespanLogging C:\Users\Admin\j
ulia\packages\TimespanLogging\DsD30\src\core.jl:229
From worker 5: | Error: Error during log aggregation
:
From worker 5: | exception =
From worker 5: | IOError: listen: address already
in use (EADDRINUSE)
From worker 5: | Stacktrace:
From worker 5: | [1] uv_error
From worker 5: | @ .\libuv.jl:100 [inlined]
From worker 5: | [2] #listen#13
From worker 5: | @ C:\Users\Admin\.julia\juli
aup\julia-1.10.2+0.x64.w64.mingw32\share\julia\stdlib\v1.10\S
ockets\src\Sockets.jl:628 [inlined]
From worker 5: | [3] listen
From worker 5: | @ C:\Users\Admin\.julia\juli
aup\julia-1.10.2+0.x64.w64.mingw32\share\julia\stdlib\v1.10\S
ockets\src\Sockets.jl:627 [inlined]
From worker 5: | [4] #listen#10
From worker 5: | @ C:\Users\Admin\.julia\juli
aup\julia-1.10.2+0.x64.w64.mingw32\share\julia\stdlib\v1.10\S
```

Upon receiving the request

```
IOError: write: operation canceled (ECANCELED)
Stacktrace:
 [1] uv_write(s::Sockets.TCPSocket, p::Ptr{UInt8}, n::UInt64
)
 @ Base .\stream.jl:1066
 [2] unsafe_write(s::Sockets.TCPSocket, p::Ptr{UInt8}, n::UI
nt64)
 @ Base .\stream.jl:1120
 [3] unsafe_write
 @ .\io.jl:698 [inlined]
 [4] unsafe_write(s::Sockets.TCPSocket, p::Base.RefValue{UIn
t16}, n::Int64)
 @ Base .\io.jl:696
 [5] write
 @ .\io.jl:699 [inlined]
 [6] write
 @ .\io.jl:702 [inlined]
 [7] writeframe(io::HTTP.Connections.Connection{Sockets.TCPS
ocket}, x::HTTP.WebSockets.Frame)
 @ HTTP.WebSockets C:\Users\Admin\.julia\packages\HTTP\sJD
5V\src\WebSockets.jl:187
 [8] send(ws::HTTP.WebSockets.WebSocket, x::String)
 @ HTTP.WebSockets C:\Users\Admin\.julia\packages\HTTP\sJD
5V\src\WebSockets.jl:542
 [9] client_handler(sock::HTTP.WebSockets.WebSocket, id::Int
64, port::Int64, port_range::UnitRange{Int64}, config_updated
::Base.RefValue{Bool}, config::Vector{Any}, seek_store::Nothi
```

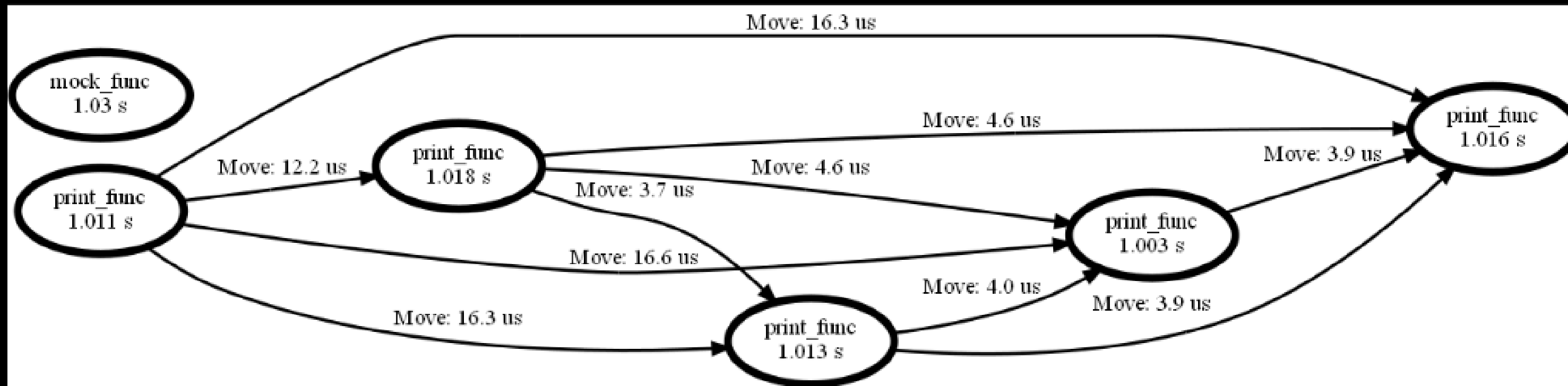
DaggerWebDash

Summary

- Plots are quite incomprehensible
- It is hard to summarize information for all the workers at once
- Random update times (from seconds to complete stagnation)
- Constantly throwing errors

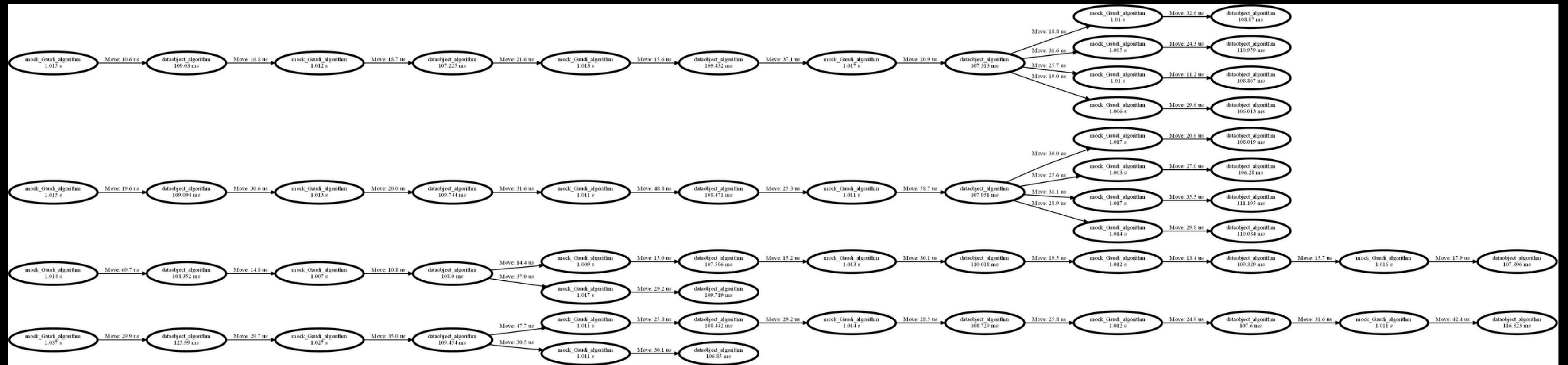
Using raw logs

Dagger provides means to convert logs to a DAG in the **.dot** format



DAG from logs: results

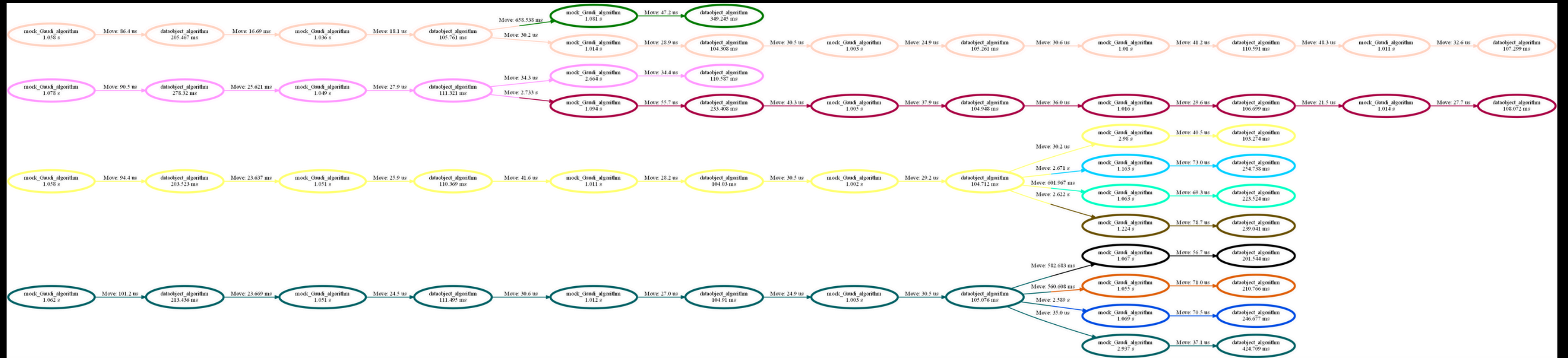
"Move" operations time: ~10-100 us



Sequential DAG execution (1 process, 1 thread)

DAG from logs: results

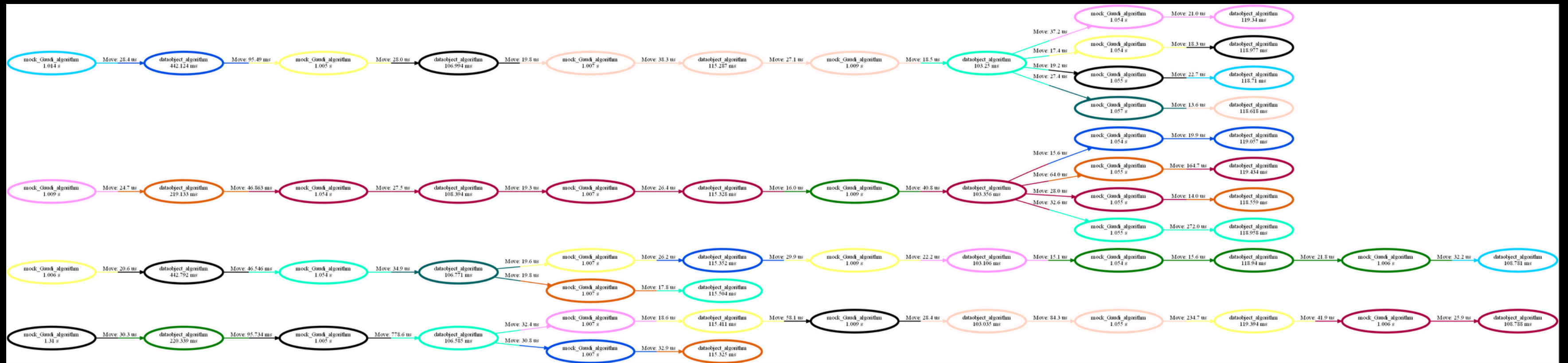
"Move" operations time: ~500-3000 ms (between processes)



Multiple processes DAG execution (13 processes, 1 thread each)

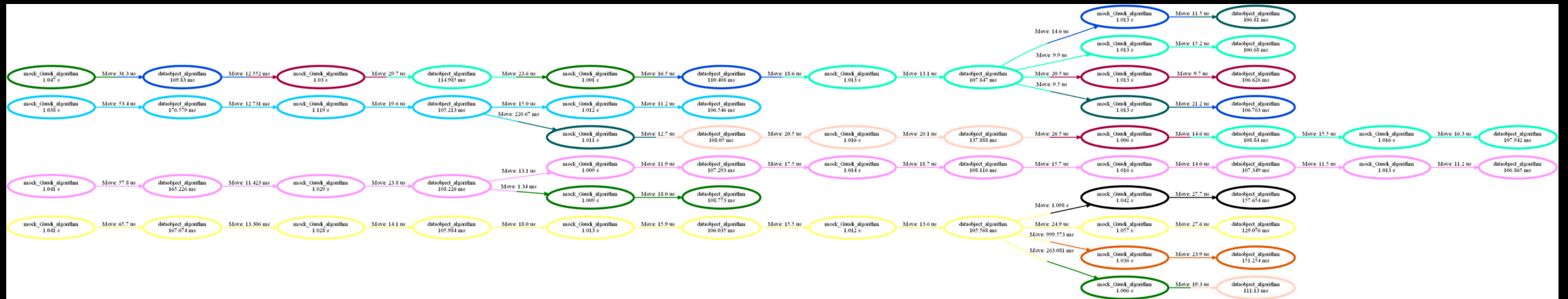
DAG from logs: results

“Move” operations time: ~10-100 us (however, there are some outliers)



Multithreaded DAG execution (1 process, 13 threads)

DAG from logs: results



DAG execution (Main process - 7 threads; 6 workers - 1 thread)

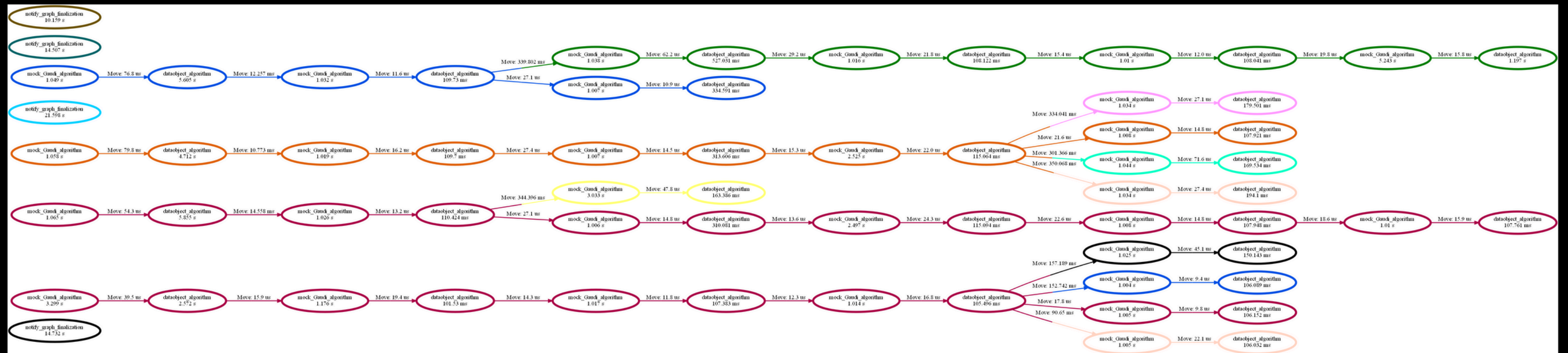
DAG from logs: results

```
ThunkFailedException:  
  Root Exception Type: CapturedException  
  Root Exception:  
ConcurrencyViolationError("lock
```

 Distributed.jl bug: ConcurrencyViolationError("lock must be held") ups
#478 opened on Mar 5 by schlichtanders

DAG execution (Main process - 1 thread; 6 workers - 2 threads)

Limit the number of concurrently running DAGs



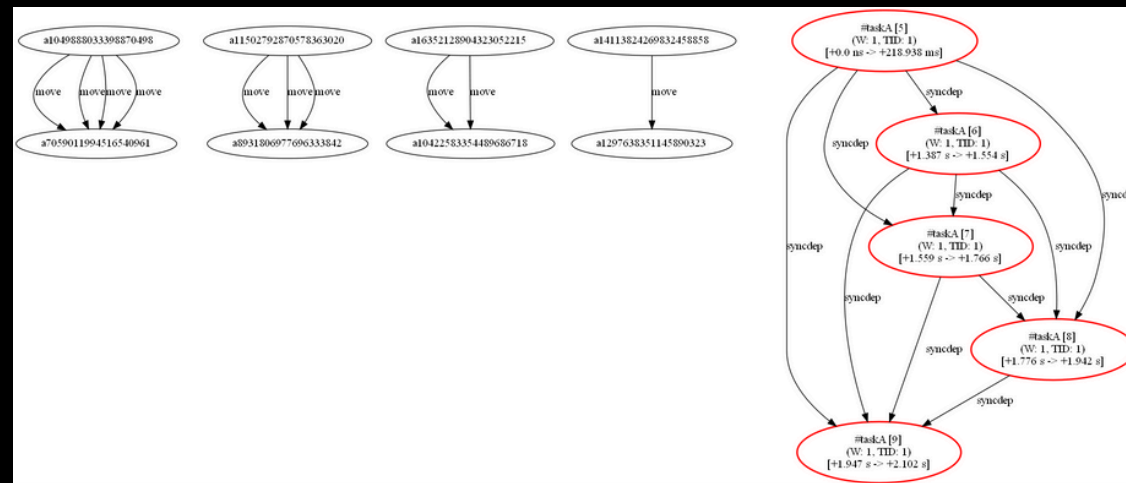
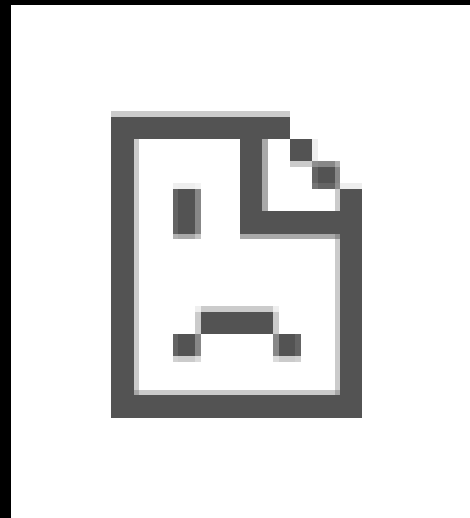
Each DAG notifies the dispatcher (process 1) by **RemoteChannel** upon its completion. After that, the new tasks are scheduled.

Dagger: logging (v 0.18.11)

DaggerWebDash

Render logs

Raw logs processing



```


TERMINAL julialauncher +
TimespanLogging.Timespan[TimespanLogging.Timespan(:init_proc, (worker = 1,), nothing, 0x00
01c6bd382ccc94, 0x0001c6bd382f2624, Base.GC.Diff(13968, 0, 0, 123, 0, 0, 0, 0, 0), Timespa
nLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UI
nt64[])], TimespanLogging.Timespan(:init_proc, (worker = 1,), nothing, 0x0001c6bd67de29d8,
0x0001c6bd67e0f190, Base.GC.Diff(13872, 0, 0, 121, 0, 0, 0, 0, 0), TimespanLogging.Profile
rResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], Time
spanLogging.Timespan(:scheduler_init, nothing, OSProc(1), 0x0001c6bca991e17c, 0x0001c6bd67e7b
7f0, Base.GC.Diff(72368704, 0, 0, 1010917, 1085, 0, 124564400, 1, 1), TimespanLogging.Profil
erResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], Time
spanLogging.Timespan(:schedule, (thunk_id = 10,), (thunk_id = 10,), 0x0001c6bd67ea73f0, 0x
0001c6bd67eeab3c, Base.GC.Diff(12752, 0, 0, 164, 0, 0, 0, 0, 0), TimespanLogging.ProfilerR
esult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], Timespan
Logging.Timespan(:enqueue, (processor = Dagger.ThreadProc(1, 1), thunk_id = 10), nothing, 0
x0001c6bd6a3e68c8, 0x0001c6bd6a4058e0, Base.GC.Diff(6768, 0, 0, 61, 0, 0, 0, 0, 0), Time
spanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UI
nt64[])], TimespanLogging.Timespan(:proc_run_wait, (worker = 1, processor = Dagger.ThreadP
roc(1, 1),), nothing, 0x0001c6bd6a428f48, 0x0001c6bd6a437d90, Base.GC.Diff(6352, 0, 0, 53,
0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTra
ces.StackFrame}}{0}, UInt64[])], TimespanLogging.Timespan(:proc_run_fetch, (worker = 1, pr
ocessor = Dagger.ThreadProc(1, 1),), (thunk_id = 10, proc_occupancy = 0x00000000, task_occup
ancy = 0xffffffff), 0x0001c6bd6a450624, 0x0001c6bd6a45d39c, Base.GC.Diff(6528, 0, 0, 60, 0
, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTrac
es.StackFrame}}{0}, UInt64[])], TimespanLogging.Timespan(:fire, (worker = 1,), nothing, 0x0
001c6bd67f627a4, 0x0001c6bd6a480874, Base.GC.Diff(350368, 0, 0, 4387, 0, 0, 0, 0, 0), Time
spanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0},
UInt64[])], TimespanLogging.Timespan(:storage_wait, (thunk_id = 10, processor = Dagger.Thr
eadProc(1, 1),), (f = taskA, device = MemPool.CPURAMDevice), 0x0001c6bd6a4a933c, 0x0001c6bd
6a4c8ea8, Base.GC.Diff(7440, 0, 0, 75, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt
64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[])], TimespanLogging.Tim
espan(:move, (thunk_id = 10, id = 0, processor = Dagger.ThreadProc(1, 1),), (f = taskA, dat
a = taskA), 0x0001c6bd6a4e29fc, 0x0001c6bd6af3164c, Base.GC.Diff(34056, 0, 0, 428, 0, 0, 0
, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.Sta
ckFrame}}{0}, UInt64[0x0000002034a91bc20])), TimespanLogging.Timespan(:move, (thunk_id = 10,
id = -1, processor = Dagger.ThreadProc(1, 1),), (f = taskA, data = 1), 0x0001c6bd6af497d8,
0x0001c6bd6bca8960, Base.GC.Diff(33864, 0, 0, 426, 0, 0, 0, 0, 0), TimespanLogging.Profil
erResult(UInt64[], Dict{UInt64, Vector{Base.StackTraces.StackFrame}}{0}, UInt64[0x0000002034
a91be10])), TimespanLogging.Timespan(:move, (thunk_id = 10, id = -2, processor = Dagger.Th
readProc(1, 1),), (f = taskA, data = 1), 0x0001c6bd6bcc19c4, 0x0001c6bd6bcd3a70, Base.GC.Di
ff(6352, 0, 0, 55, 0, 0, 0, 0, 0), TimespanLogging.ProfilerResult(UInt64[], Dict{UInt64, V

```

Doesn't work anymore


Dagger: fixing visualization

Fix write dag #531

 Open

SmalRat wants to merge 7 commits into `JuliaParallel:master` from `SmalRat:fix-write_DAG` 

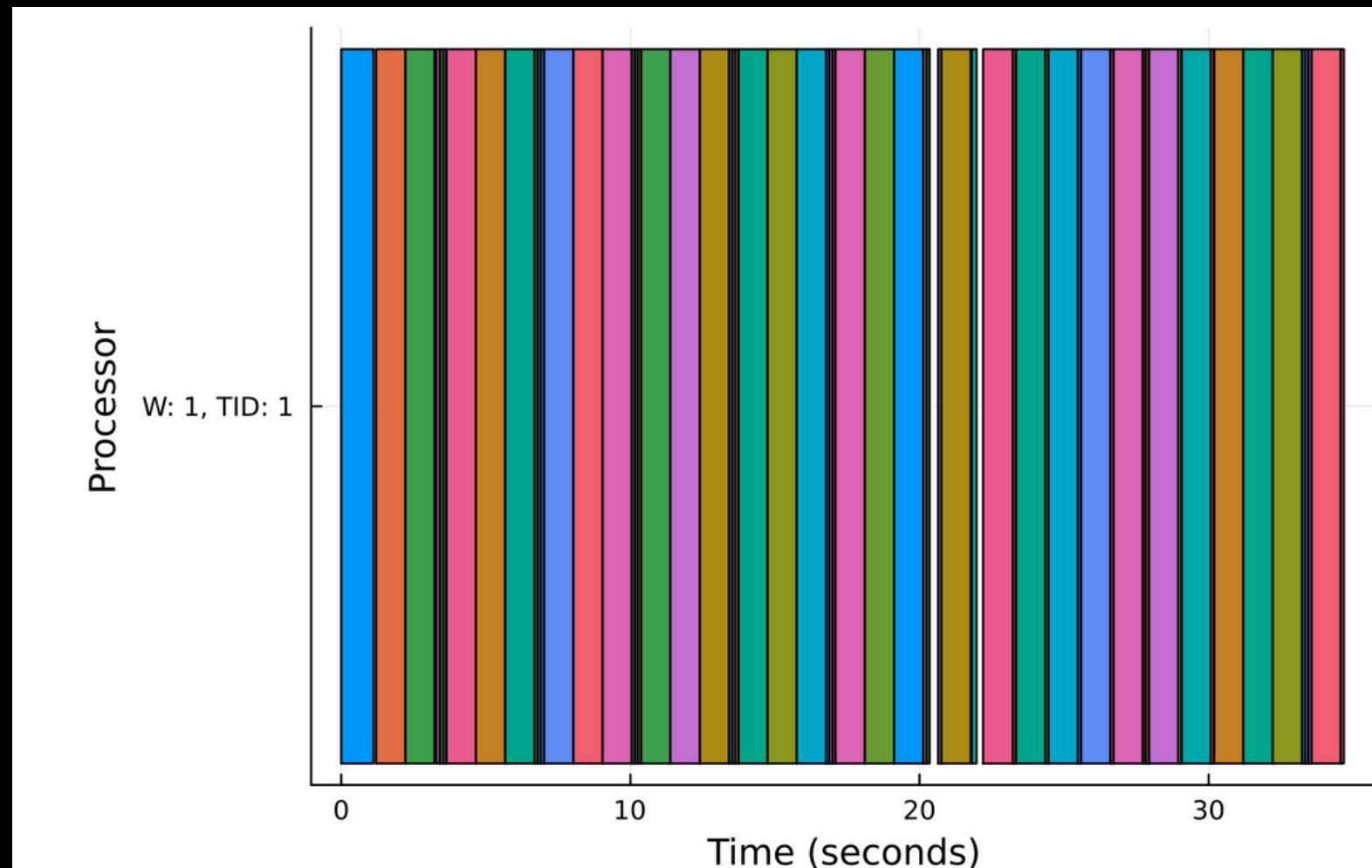
 Conversation 0

 Commits 7

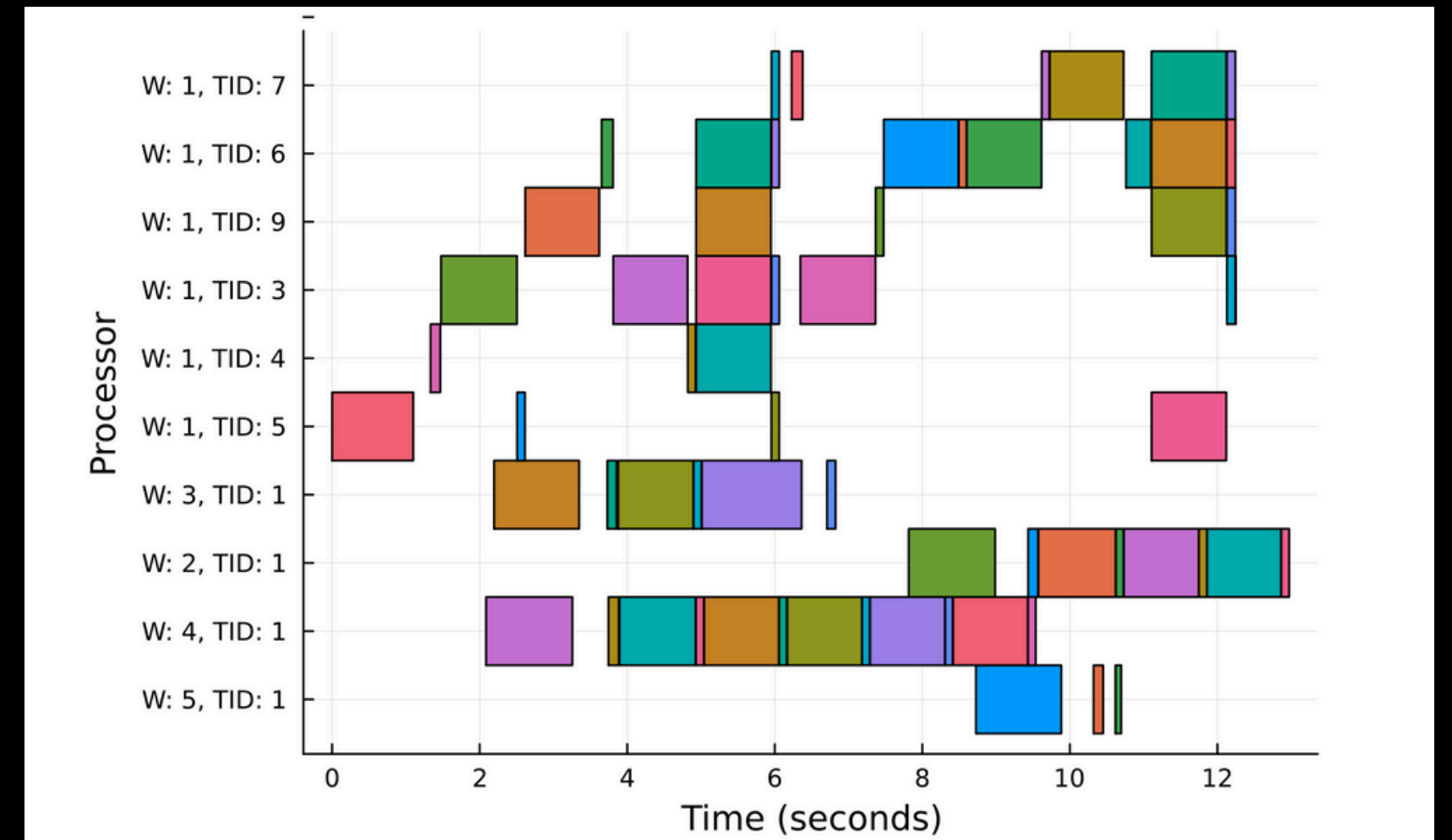
 Checks 2

 Files changed 2

Dagger 0.18.11: Gantt chart



1 process, 1 thread



Main process - 9 threads,
4 workers - 1 thread

Examples & Documentation

0.18.8

examples	
auxiliary	
modernAPI_tasks.jl	
oldAPI_example_tasks.jl	
oldAPI_tasks.jl	
examples_results	
logging	
log_sink	
results	
modernAPI_localeventlog.jl	
modernAPI_multieventlog.jl	
oldAPI_localeventlog.jl	
oldAPI_multieventlog.jl	
enable_logging.jl	
Logging_in_Dagger.md	
webdash.jl	
Examples.md	
killing_procs.jl	
parallel_complex_DAGs_scheduling.jl	
parallel_simple_DAGs_scheduling.jl	
processors_example.jl	

0.18.11

examples	
auxiliary	
example_tasks.jl	U
example_tasks.md	U
examples_results	
logging	
enable_logging	
write_DAG_modernAPI.jl	U
write_DAG_oldAPI_custom_logging.jl	U
write_DAG_oldAPI.jl	U
log_sink	
results	
modernAPI_localeventlog.jl	M
modernAPI_multieventlog.jl	
oldAPI_localeventlog.jl	
oldAPI_multieventlog.jl	M
enable_logging.jl	
Logging_in_Dagger.md	M
webdash.jl	M
killing_procs.jl	
parallel_complex_DAGs_scheduling.jl	M
parallel_simple_DAGs_scheduling.jl	M
processors_example.jl	

Dagger: summary

Issues

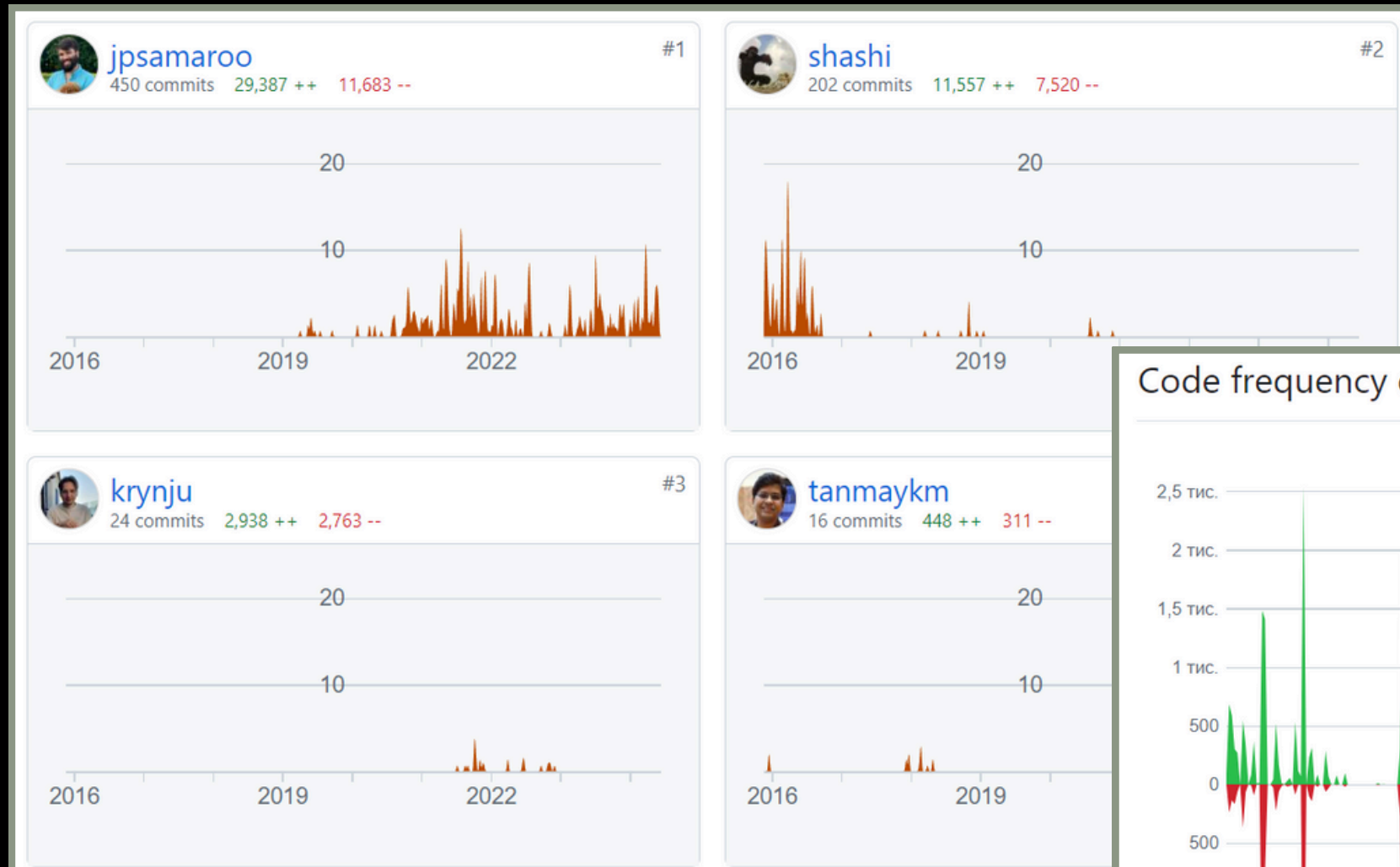
- Many features are undocumented
- Some documented features do not work
- Frequent updates are not backward compatible
- Relatively small community

Advantages

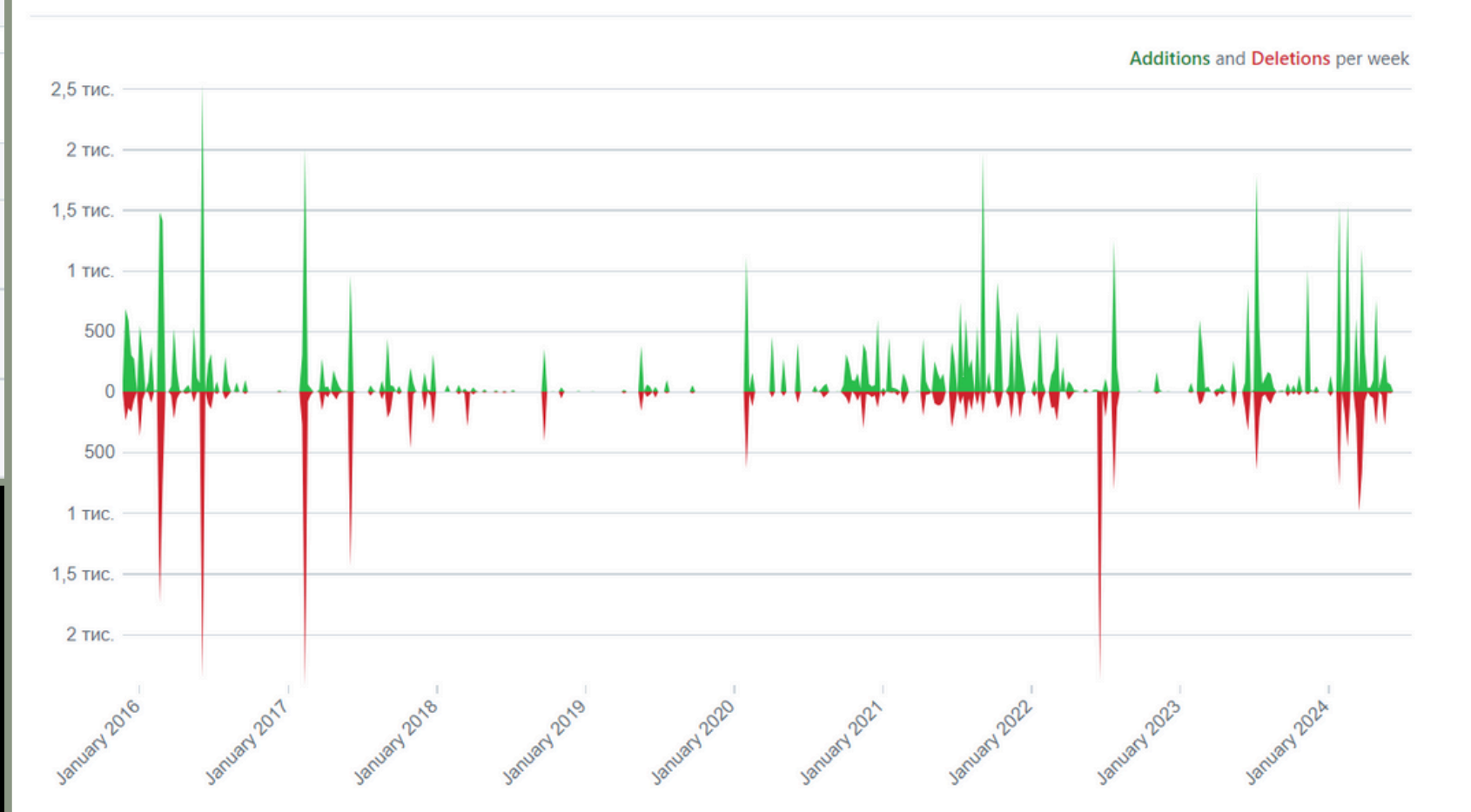
- Actively maintained
- Easy to use, when documented

Dagger is promising , but it may be too early to use it

Dagger: updates



Code frequency over the history of JuliaParallel/Dagger.jl



Thank you for your attention!