Introducing Wrek

A Library For Executing Dependency Graphs

Richard Kallos

Samsung Ads Canada (formerly Adgear)

Code BEAM SF, March 2018

`https://gitlab.com/rkallos/code-beam-2018/raw/master/presentation.pdf`

Table of Contents

1. Introduction
2. Theory
   - Parallelism
   - Dependency Graphs
   - Topological ordering
3. Design of Wrek
   - General
   - Specific
4. Use
   - Wrek @ $WORK
   - Erlang to the rescue!
5. Conclusion

Table of Contents

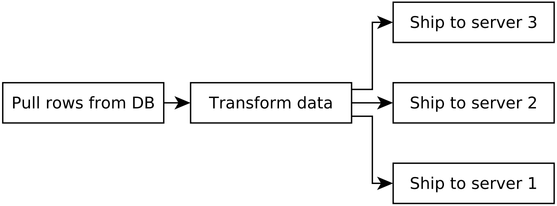
1. Introduction
2. Theory
   - Parallelism
   - Dependency Graphs
   - Topological ordering
3. Design of Wrek
   - General
   - Specific
4. Use
   - Wrek @ $WORK
   - Erlang to the rescue!
5. Conclusion

What is wrek?

Wrek is a library for executing task dependency graphs.

What is wrek?

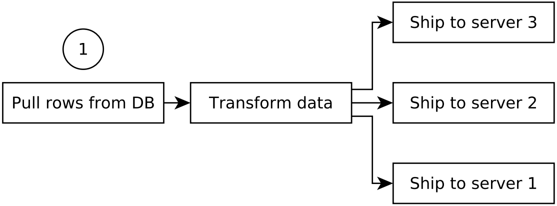
Given a graph like:



Wrek executes tasks in topological order

What is wrek?

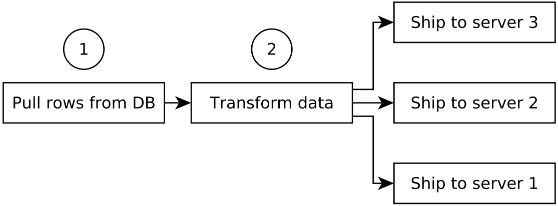
Given a graph like:



Wrek executes tasks in topological order

What is wrek?

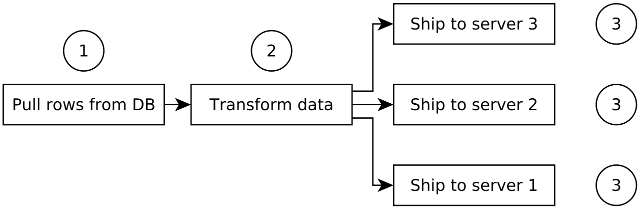
Given a graph like:



Wrek executes tasks in topological order

What is wrek?

Given a graph like:



Wrek executes tasks in topological order

Table of Contents

1 Introduction

2 Theory
- Parallelism
- Dependency Graphs
- Topological ordering

3 Design of Wrek
- General
- Specific

4 Use
- Wrek @ $WORK
- Erlang to the rescue!

5 Conclusion

Parallelism

There are many kinds of parallelism in computing.

Parallelism

There are many kinds of parallelism in computing.
Two kinds of parallelism that are often mentioned together are:

Parallelism

There are many kinds of parallelism in computing.
Two kinds of parallelism that are often mentioned together are:

- Data parallelism: Splitting data across processors

Parallelism

There are many kinds of parallelism in computing.
Two kinds of parallelism that are often mentioned together are:

- Data parallelism: Splitting data across processors
- Task parallelism: Splitting tasks across processors

Parallelism

There are many kinds of parallelism in computing.
Two kinds of parallelism that are often mentioned together are:

- Data parallelism: Splitting data across processors
- Task parallelism: Splitting tasks across processors

These two forms can be (and are) used together!

Parallelism

There are many kinds of parallelism in computing.
Two kinds of parallelism that are often mentioned together are:

- Data parallelism: Splitting data across processors
- Task parallelism: Splitting tasks across processors

These two forms can be (and are) used together!
e.g. Image processing consists of pipelines of data-parallel tasks

Dependency Graphs

- A dependency graph is a directed acyclic graph (DAG) whose edges model a dependency relation between vertices.

Dependency Graphs

- A dependency graph is a directed acyclic graph (DAG) whose edges model a dependency relation between vertices.
- An edge (a, b) in a dependency graph means "a depends on b".

Dependency Graphs

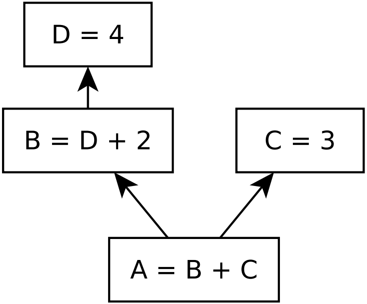
- A dependency graph is a directed acyclic graph (DAG) whose edges model a dependency relation between vertices.
- An edge (a, b) in a dependency graph means “a depends on b”.
- An edge (b, a) in the transpose of the graph means “b is a dependency of a”

Dependency Graphs

- A dependency graph is a directed acyclic graph (DAG) whose edges model a dependency relation between vertices.
- An edge (a, b) in a dependency graph means “a depends on b”.
- An edge (b, a) in the transpose of the graph means “b is a dependency of a”
- Vertices with no paths connecting them can execute concurrently

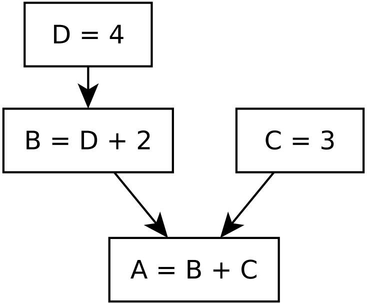
Dependency Graphs

Here is a dependency Graph...



Dependency Graphs

… and its transpose



D = 4

B = D + 2

C = 3

A = B + C

Dependency Graphs

In addition to being widespread in computing, dependency graphs are also used by humans!

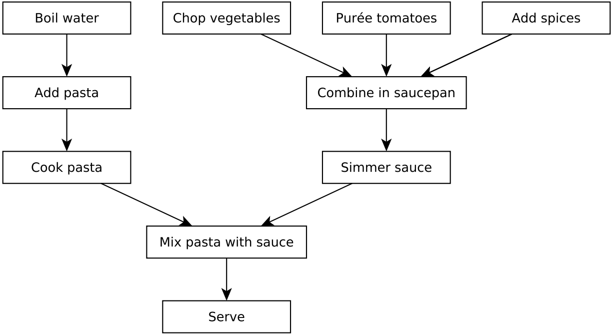
Dependency Graphs

In addition to being widespread in computing, dependency graphs are also used by humans!

- Many of the lists we make are topological orderings of dependency graphs
  e.g. to-do lists, cooking recipes, checklists
  e.g. 1. Foo the bar. 2. Baz the foo'd bar...

Dependency Graphs

Cooking recipes are topological orderings of dependency graphs.



Topological ordering

- Topological ordering: For every edge (u, v), u comes before v.

Topological ordering

- Topological ordering: For every edge (u, v), u comes before v.
- A topological ordering of a dependency graph is a valid evaluation order.

Topological ordering

- Topological ordering: For every edge (u, v), u comes before v.
- A topological ordering of a dependency graph is a valid evaluation order.
- e.g. [boil_water, chop_vegetables, add_pasta, purée_tomatoes, add_spices, ...]

Topological ordering

- Topological ordering: For every edge (u, v), u comes before v.
- A topological ordering of a dependency graph is a valid evaluation order.
- e.g. [boil_water, chop_vegetables, add_pasta, purée_tomatoes, add_spices, ...]
- Topo-sorting dependency graphs *discards information* about possible concurrency

A Thought

What if we could write arbitrary code as dependency graphs and have them execute with maximum concurrency?

Table of Contents

Design of Wrek

- OTP behaviours let library/application developers separate the *general* from the *specific*

Design of Wrek

- OTP behaviours let library/application developers separate the *general* from the *specific*
- General: Executing dependency graphs in proper order

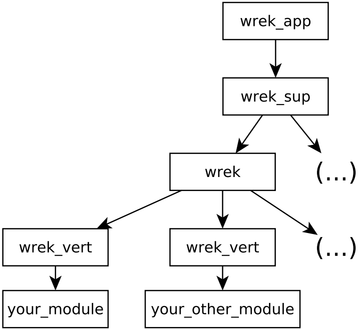
Design of Wrek

- OTP behaviours let library/application developers separate the *general* from the *specific*
- General: Executing dependency graphs in proper order
- Specific: The structure of dependency graphs

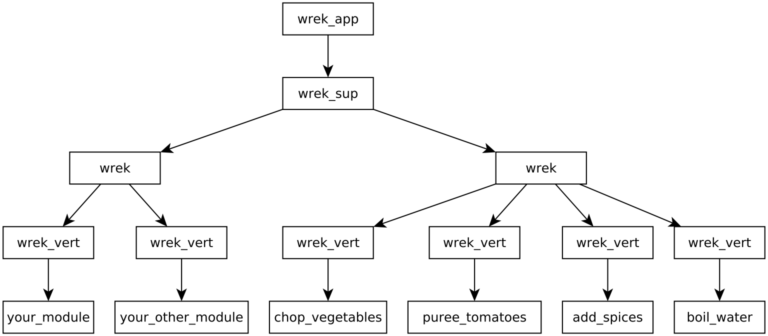
Design of Wrek

- OTP behaviours let library/application developers separate the *general* from the *specific*
- General: Executing dependency graphs in proper order
- Specific: The structure of dependency graphs
- Specific: Executing single vertices

General


wrek_app
wrek_sup
wrek
(...)
wrek_vert
wrek_vert
(...)
your_module
your_other_module

General


wrek_app
wrek_sup
wrek
wrek
wrek_vert
wrek_vert
wrek_vert
wrek_vert
wrek_vert
wrek_vert
your_module
your_other_module
chop_vegetables
puree_tomatoes
add_spices
boil_water

Specific

```
-module(wrek_vert).

-callback run(Args :: list(), Parent :: pid()) ->
    {ok, Result :: any()} | {error, Reason :: any()}.
```

Specific

```
-type dag_map() :: #{any() := vert_defn()} |
                   [{any(),   vert_defn()}].

-type vert_defn() :: #{
    module := module(),
    args   := list(),
    deps   := list()
}.
```

What happens when you call `wrek:start/2`

- The supplied map is read into a `digraph:graph()`

What happens when you call `wrek:start/2`

- The supplied map is read into a `digraph:graph()`
- All vertices with no queued or running dependencies are `spawn_linked`

What happens when you call `wrek:start/2`

- The supplied map is read into a `digraph:graph()`
- All vertices with no queued or running dependencies are `spawn_linked`
- Values returned from vertices are stored in labels for use by later vertices via `wrek_vert:get/3`

What happens when you call wrek:start/2

- The supplied map is read into a digraph:graph()
- All vertices with no queued or running dependencies are spawn_linked
- Values returned from vertices are stored in labels for use by later vertices via wrek_vert:get/3
- Calls gen_event:notify/2 with #wrek_event{} records to an optional gen_event process

What happens when you call `wrek:start/2`

- The supplied map is read into a `digraph:graph()`
- All vertices with no queued or running dependencies are `spawn_linked`
- Values returned from vertices are stored in labels for use by later vertices via `wrek_vert:get/3`
- Calls `gen_event:notify/2` with `#wrek_event{}` records to an optional `gen_event` process
- If `wrek_vert:Module:run/2` returns an error or throws an exception, the crash propagates to the rest of the graph.

Table of Contents

1 Introduction

2 Theory
- Parallelism
- Dependency Graphs
- Topological ordering

3 Design of Wrek
- General
- Specific

4 Use
- Wrek @ $WORK
- Erlang to the rescue!

5 Conclusion

Wrek @ $WORK

- Our edge servers (bidders) were all wasting a CPU core doing the same calculation

Wrek @ $WORK

- Our edge servers (bidders) were all wasting a CPU core doing the same calculation
- Solution: Do the calculation off the edge, and ship the result

Wrek @ $WORK

- Our edge servers (bidders) were all wasting a CPU core doing the same calculation
- Solution: Do the calculation off the edge, and ship the result
- Version 1 was implemented with...

Wrek @ $WORK

- Our edge servers (bidders) were all wasting a CPU core doing the same calculation
- Solution: Do the calculation off the edge, and ship the result
- Version 1 was implemented with...`cron` and `bash`

Wrek @ $WORK

- Version 1 mostly worked, but we realized this offered opportunities to take pressure off services on other servers

Wrek @ $WORK

- Version 1 mostly worked, but we realized this offered opportunities to take pressure off services on other servers
- If we are going to extend this new system, it would be better to create a more robust framework

Wrek @ $WORK

- Version 1 mostly worked, but we realized this offered opportunities to take pressure off services on other servers
- If we are going to extend this new system, it would be better to create a more robust framework
- Enter Erlang!

Erlang to the rescue!

- In order to iterate quickly, it made sense to have a library that could

Erlang to the rescue!

- In order to iterate quickly, it made sense to have a library that could
  - Run Erlang callbacks

Erlang to the rescue!

- In order to iterate quickly, it made sense to have a library that could
  - Run Erlang callbacks
  - Run our already-existing shell scripts (secretly topological orderings of dependency graphs)

Erlang to the rescue!

- In order to iterate quickly, it made sense to have a library that could
  - Run Erlang callbacks
  - Run our already-existing shell scripts (secretly topological orderings of dependency graphs)
- Wrek was the result

Erlang to the rescue!

- Wrek was able to easily slurp our existing scripts (thanks to `erlexec`)

Erlang to the rescue!

- Wrek was able to easily slurp our existing scripts (thanks to `erlexec`)
- This allowed for piecemeal replacement of large-ish scripts with dependency graphs of smaller scripts and Erlang callbacks

Erlang to the rescue!

- Wrek was able to easily slurp our existing scripts (thanks to `erlexec`)
- This allowed for piecemeal replacement of large-ish scripts with dependency graphs of smaller scripts and Erlang callbacks
- This offered better concurrency and (much) more information for logging/monitoring

Erlang to the rescue!

Thanks to Erlang/OTP, we are flexible to handle events generated by Wrek

Erlang to the rescue!

Thanks to Erlang/OTP, we are flexible to handle events generated by Wrek

- Exposing status of executing graphs via a HTTP endpoint

Erlang to the rescue!

Thanks to Erlang/OTP, we are flexible to handle events generated by Wrek

- Exposing status of executing graphs via a HTTP endpoint
- Establishing contracts between on- and off-edge servers

Table of Contents

1 Introduction

2 Theory
- Parallelism
- Dependency Graphs
- Topological ordering

3 Design of Wrek
- General
- Specific

4 Use
- Wrek @ $WORK
- Erlang to the rescue!

5 Conclusion

Conclusion

- Dependency graphs are useful for exposing opportunities for concurrency

Conclusion

- Dependency graphs are useful for exposing opportunities for concurrency
- Dependency graphs show up all over the place, in computing and in everyday life

Conclusion

- Dependency graphs are useful for exposing opportunities for concurrency
- Dependency graphs show up all over the place, in computing and in everyday life
- Wrek is an application that executes dependency graphs

Conclusion

- Dependency graphs are useful for exposing opportunities for concurrency
- Dependency graphs show up all over the place, in computing and in everyday life
- Wrek is an application that executes dependency graphs
- Erlang/OTP has been instrumental in letting us build and ship quickly, and start paying down our shell-script technical debt

Conclusion

- Dependency graphs are useful for exposing opportunities for concurrency
- Dependency graphs show up all over the place, in computing and in everyday life
- Wrek is an application that executes dependency graphs
- Erlang/OTP has been instrumental in letting us build and ship quickly, and start paying down our shell-script technical debt
- Big thanks to `digraph` and `erlexec`; they do the heavy lifting.

Conclusion

If you

Conclusion

If you

- Enjoy thinking about larger tasks being composed of dependency graphs of smaller tasks (Try it! It's fun!)

Conclusion

If you

- Enjoy thinking about larger tasks being composed of dependency graphs of smaller tasks (Try it! It's fun!)
- Want to incrementally replace a mess of shell scripts with smaller, more concurrent ones (or Erlang code)

Conclusion

If you

- Enjoy thinking about larger tasks being composed of dependency graphs of smaller tasks (Try it! It's fun!)
- Want to incrementally replace a mess of shell scripts with smaller, more concurrent ones (or Erlang code)

then you might enjoy Wrek!

Conclusion

- http://github.com/rkallos/wrek
- http://github.com/saleyn/erlexec
- http://erlang.org/doc/man/digraph.html

Conclusion

Thank you!