

Reduce Redux

Mark Greenstreet

CpSc 418 – Jan. 18, 2016

Outline:

- Finishing Reduce
- Scan

The Reduce Pattern

- It's a parallel version of *fold*, e.g. `lists:foldl` and `lists:foldr`.
- Reduce is described by three functions:
 - ▶ *Leaf()*: What to do at the leaves, e.g. `fun() -> count3s(Data) end`.
 - ▶ *Combine()*: What to do at intermediate nodes, e.g. `fun(Left, Right) -> Left+Right end`.
 - ▶ *Root()*: What to do with the final result. For count 3s, this is just the identity function.

The `wtree` module

- Part of the [course Erlang library](#).
- Operations on worker trees”
 - ▶ `wtree:create(NProcs) -> [pid()]`. Create a list of `NProcs` processes, organized as a tree.
 - ▶ `wtree:broadcast(W, Task, Arg) -> ok`. Execute the function `Task` on each process in `W`. Note: `W` means “worker pool”.
 - ▶ `wtree:reduce(P, Leaf, Combine, Root) -> term()`. A generalized reduce.
 - ▶ `wtree:reduce(P, Leaf, Combine) -> term()`. A generalized reduce where `Root` defaults to the identity function.

Store Locally

- Processes should store their data locally.
- How do we store data in a functional language?
 - ▶ Our processes are implemented as Erlang functions that receive messages, process the message, and make a tail-call to be ready to receive the next message.
 - ▶ We add a parameter to these functions, `ProcState`, that is a mapping from *Keys* to *Values*.
- What this means when we write code:
 - ▶ Functions such as *Leaf* for `wtree:reduce` or *Task* for `wtree:broadcast` have a parameter for `ProcState`.
 - ▶ `worker:put(ProcState, Key, Value) -> NewProcState`. Create a new version of `ProcState` that associates `Value` with `Key`.
 - ▶ `worker:get(ProcState, Key, Default) -> Value`. Return the value associated with `Key` in `ProcState`. If no such value is found, `Default` is returned. Note: `Default` can be a function in which case it is called to determine a default value – see the documentation.

Count3s using `wtree` – Design

- Example problem: I've got a list 4 billion elements distributed across 100 processes.
- What should *Leaf* do?
 - ▶ A process has a list of 40 million elements.
 - ▶ I want to know the total number of 3s.
 - ▶ What should each process report?
- What should *Combine* do?
 - ▶ I have the answers from my left and right subtrees, how should I combine them?
- What should *Root* do?
 - ▶ I have the result for *Combine* for the whole tree. What is the final answer for `count3s`?

Count3s using wtree – Code

```
count3s_par(N, P) ->
  W = wtree:create(P),
  wtree:rlist(W, N, 10, 'Data'),
  wtree:barrier(W),
  wtree:reduce(W,
    fun(ProcState) ->
      count3s(workers:get(ProcState, 'Data')) end,
    fun(Left, Right) -> Left+Right end,
    % Root is the identity function – that's the default.
  ).
```

- Let's try it:

```
1> W = wtree:create(4).
[<0.36.0>, <0.37.0>, <0.38.0>, <0.39.0>]
2> examples:count3s_par(W, 1000).
105
```

- Seems plausible, but how can we be sure?

retrieve – typical use

- `workers:retrieve(W, Key)` returns a list composed of the results of `get(ProcState, Key)` for each worker in `W`.
- Example:

```
3> examples:count3s_par(W, 80).
5
4> Data = workers:retrieve(W, 'Data').
[[8,7,5,5,2,2,9,7,2,1,1,2,5,2,4,8,8,4,2,2],
 [2,8,3,2,2,4,8,4,2,10,4,7,3,6,10,6,7,8,8,6],
 [4,5,5,1,4,10,4,6,10,4,8,9,10,8,1,10,5,9,8,5],
 [10,8,8,2,5,6,7,3,10,10,6,9,4,1,9,7,5,3,9,3]]
5> examples:count3s(lists:append(Data)).
5
```

- Notice how `Data` is a list of lists.
 - ▶ Each process returned a list.
 - ▶ Each of these lists is a list in `Data`.
 - ▶ The order of the lists in `Data` matches the order of the processes in `W`.

retrieve – general case

- `retrieve(W, Fun)` when `is_function(Fun, 1)`
Execute `Fun(ProcState)` in each process and make a list of the results. `ProcState` is the process state as described on slide [slide 4](#). Note that each process has its own `ProcState`. For example,

```
6> workers:retrieve(W, fun(ProcState) ->
    examples:count3s(wtree:get(ProcState, 'Data')) end).
[0,2,0,3]
```

- `retrieve(W, Fun)` when `is_function(Fun, 2)`
Execute `Fun(ProcState, ProcIndex)` in each process and make a list of the results. `ProcIndex` is the position of the worker process in `W` – in other words, the process `hd(W)`, is called with `ProcIndex=1`, the process `hd(tl(W))`, is called with `ProcIndex=2`, and so on.
- `retrieve(W, Key)`, when `Key` is anything other than a function of 1 or 2 arguments is “typical use” described on the previous slide.