Mrs: High Performance MapReduce for Iterative and Asynchronous Algorithms in Python

Jeff Lund, Chace Ashcraft, Andrew McNabb and Kevin Seppi

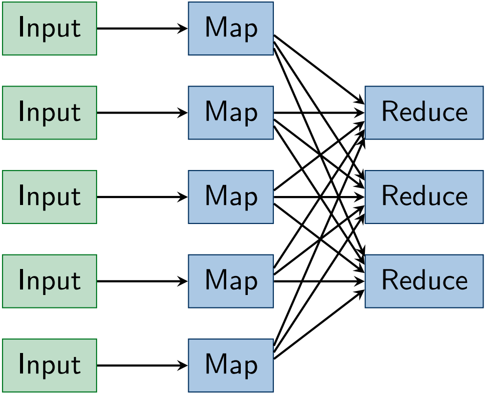
Brigham Young University

November 14, 2016

What is Mrs?

- Simple and easy to use MapReduce framework
- Implemented in pure Python
- Designed with scientific computing in mind

MapReduce


Input
Input
Input
Input
Input
Map
Map
Map
Map
Map
Reduce
Reduce
Reduce

Example: WordCount

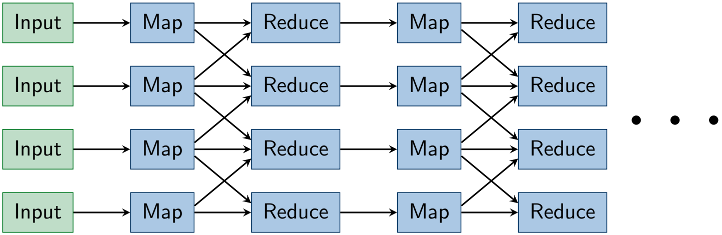
wordcount.py

```
import mrs

class WordCount(mrs.MapReduce):

    def map(self, line_num, line_text):
        for word in line_text.split():
            yield (word, 1)

    def reduce(self, word, counts):
        yield sum(counts)

if __name__ == '__main__':
    mrs.main(WordCount)
```

Why Python?

- Python is nearly ubiquitous
- Mrs needs no dependencies outside of standard library
- Familiarity and readability
- Easy interoperability
- Debugging and testing

Iterative MapReduce



Performance Challenges:

- CPU bound problems
- Communication time
- Task Management

Proposed Solutions

- Infrequent Checkpointing
- Reduce-Map task
- Generator-Callback Model
- Asynchronous Scheduling Model

How Often to Checkpoint

Let X be a random variable indicating a failure occurred during an iteration, then

$$X \sim Bernoulli\left(\frac{1}{f}\left(t + \frac{c}{n}\right)\right)$$

- n: Number of iterations between checkpoints
- t: Time to perform each iteration
- c: Extra time required for a checkpointed iteration
- f: Failures in a cluster

How Often to Checkpoint

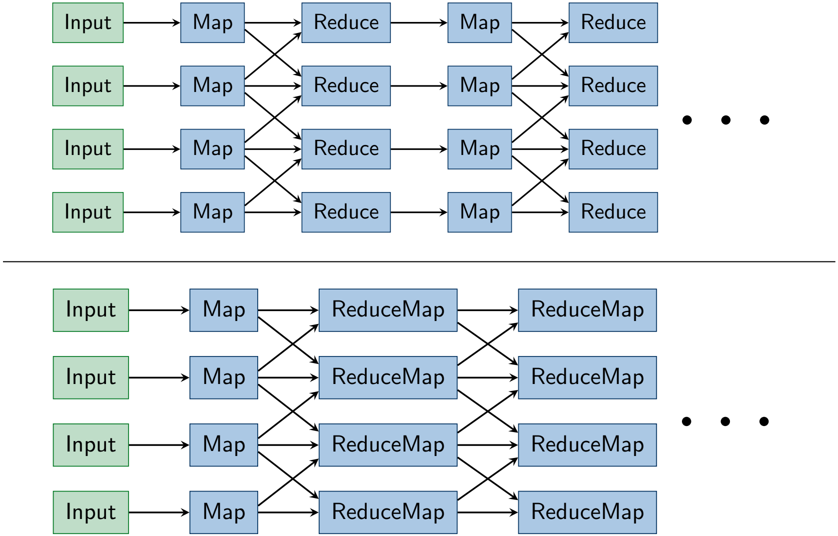
If $Y \sim Uniform(n)$ indicates the number of iterations since last checkpoint then the expected value of the number of seconds of extra work in an iteration is:

$$E\left[X\left(r+Yt\right)\right]=\frac{1}{f}\left(t+\frac{c}{n}\right)\left(r+\frac{n}{2}t\right)$$

and the breakeven number of iterations is

$$n=\max\left[1,\frac{1}{t}\left(\sqrt{\left(\frac{c}{2}+r\right)^{2}-2c(r-f)}-\left(\frac{c}{2}+r\right)\right)\right].$$

Iterative MapReduce: ReduceMap



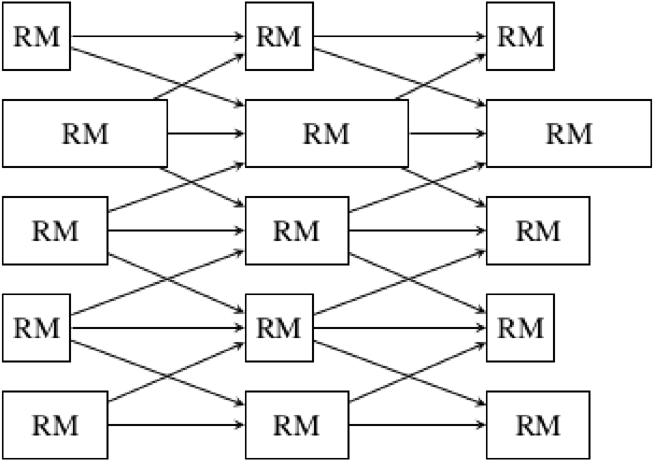
Generator-Callback Model

```
def run_batches():
    data_path = input_path
    for iteration in range(MAX_ITERATIONS):
        output_path = make_temp_path()
        job = new_job(data_path, map_func, reduce_func, output_path)
        job.wait_for_completion()
        data_path = output_path

        if iteration % CHECK_FREQUENCY == 0:
            data = read_all(data_path)
            perform_output(data)
            if converged(data):
                break
```

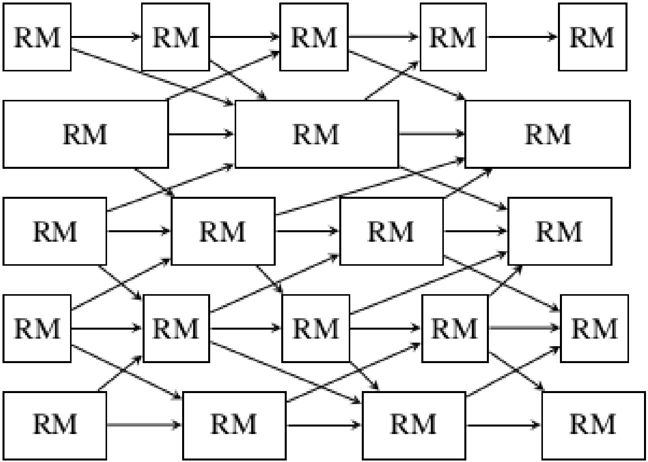
Generator-Callback Model

```
def generator(queue):
    dataset = input_data
    for iteration in range(MAX_ITERATIONS):
        output_path = make_temp_path()
        dataset = mapreduce(dataset, map_func, reduce_func, output_path)

        if iteration % CHECK_FREQUENCY == 0:
            queue.submit(dataset, callback)
        else:
            queue.submit(dataset, None)

def callback(data):
    data.read_all()
    perform_output(data)
    return !converged(data)
```

Task Dependencies: Synchronous MapReduce



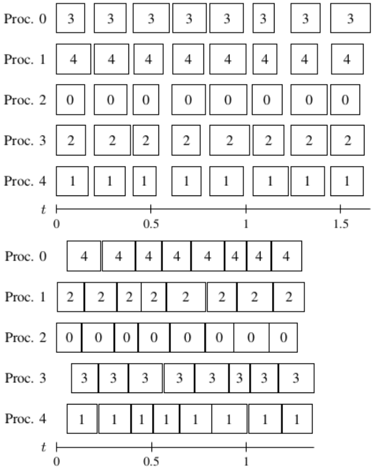
Task Dependencies: Asynchronous MapReduce



Task Execution Traces

Synchronous:

Asynchronous:



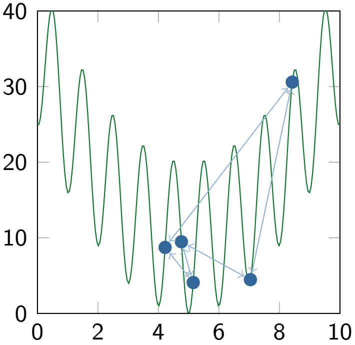
Performance and Case Studies

We demonstrate on two different problems:

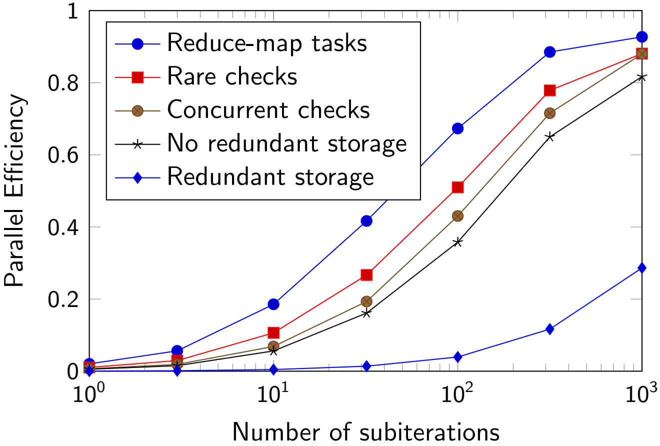
- Particle Swarm Optimization
  Minimize 250 degree Rosenbrock function
- Expectation Maximization
  Mixture of Multinomials model in the context of clustering text documents

Particle Swarm Optimization

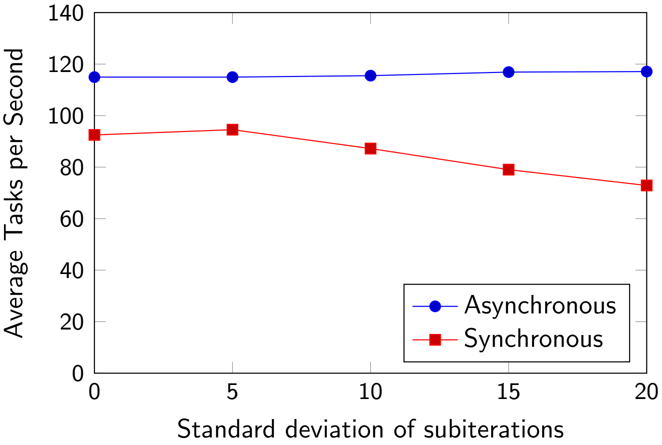
- Inspired by simulations of flocking birds
- Particles interact while exploring
- Map: motion and function evaluation
- Reduce: communication
- CPU bound problem



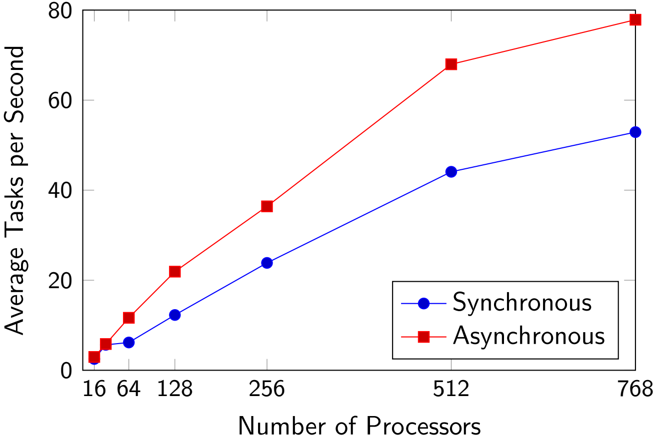
Particle Swarm Optimization



Particle Swarm Optimization: Asynchronous



Particle Swarm Optimization: Asynchronous



Expectation Maximization

Feature Set Size	80	252	8000	25298
Reduce-map tasks	0.411	0.357	0.277	0.193
Rare checks	0.362	0.314	0.253	0.18
Redundant storage	0.013	0.013	0.013	0.012

Parallel efficiency per iteration of EM for various feature set sizes.

Conclusion

By taking the following approaches, we have considerably improved performance for iterative parallel algorithms in Mrs:

- Infrequent Checkpointing
- Reduce-Map Task
- Generator-Callback Model
- Asynchronous Model

Where to find Mrs

Mrs Homepage with links to source, documentation, mailing list, etc:

`https://github.com/byu-aml-lab/mrs-mapreduce`

In case you forget the url, just google "mrs mapreduce" :)