ENSC 351 - Lab 3: MapReduce

Nic Klaassen, Galen Elfert, Diane Wolf

October 17, 2018

1 Explanation of workload

The workload invented to fit the MapReduce framework better than word counts was [matrix multiplication | distributed sort problem | merge sort problem]...

1.1 Conception

[Matrix multiplication | distributed sort problem | merge sort problem] was chosen as a better alternative to word counts because...

1.2 Speed: Single-threaded implementation

1.3 Speed: MapReduce implementation

1.4 Comparison

2 Word count efficiency

Both implementations of the program counted the instances of words in fifty paragraphs (with a total length of 2261 words) of Lorem Ipsum. They were run on the same machine with hardware to support twelve threads. Ten executions of each implementation were conducted, with the duration measured by the built-in Linux `time` command. A call graph for both implementations was generated using Valgrind's Callgrind tool.

2.1 Single-threaded implementation

The single-threaded implementation of the word count ran for a mean wall time of 0.0140 seconds. Table 1 below shows the execution time, as well as CPU usage, for each of the ten single-threaded word count runs.

See Figure 2 for a graphical representation of the call map.

Execution times for single-threaded word count				
run #	user (s)	system (s)	wall (s)	CPU usage (%)
1	0	0.004	0	0
2	0	0.008	0.01	0
3	0.004	0	0.02	0
4	0	0.004	0.01	0
5	0	0.004	0.01	0
6	0.004	0	0.01	0
7	0	0.004	0.02	0
8	0.004	0	0.01	0
9	0.004	0	0.04	0
10	0	0.004	0.01	0
mean (s)	0.0016	0.0028	0.0140	0
std. dev. (s)	0.0021	0.0027	0.0107	0

Table 1: Duration of single-threaded implementation measured by `time`

2.2 MapReduce implementation

The MapReduce implementation of the word count was tested with four threads and then the full twelve threads the machine was capable of supporting. Tables 2 and 3 below show the execution time, as well as CPU usage, for each of the ten word counts run with MapReduce. Note that the greater the quantity of threads used to multithread, the slower the program execution became. As the thread count increased, the CPU usage also appeared to increase, going from an average of 10% with four threads to an average of 40% with 12 threads.

See Figure 2 for a graphical representation of the call map.

Execution times for MapReduce word count - 4 threads				
run #	user (s)	system (s)	wall (s)	CPU usage (%)
1	0.008	0	0.02	0
2	0.008	0	0.03	0
3	0.008	0	0.02	0
4	0.008	0	0.02	0
5	0.008	0	0.01	0
6	0.008	0	0.02	0
7	0.008	0	0.01	0
8	0.008	0	0.02	0
9	0.008	0	0.03	0
10	0.012	0	0.01	100
mean (s)	0.0084	0	0.0190	10.0000
std. dev. (s)	0.0013	0	0.0074	31.6228

Table 2: Duration of MapReduce implementation measured by `time`, with four threads

Execution times for MapReduce word count - 12 threads				
run #	user (s)	system (s)	wall (s)	CPU usage (%)
1	0.008	0.004	0.01	0
2	0.008	0.008	0.03	0
3	0.008	0.008	0.01	0
4	0.008	0.004	0.01	0
5	0.016	0	0.01	100
6	0.008	0.004	0.01	0
7	0.012	0.004	0.01	100
8	0.008	0.008	0.01	0
9	0.012	0	0.01	100
10	0.016	0	0.01	100
mean (s)	0.0104	0.0040	0.0120	40.0000
std. dev. (s)	0.0034	0.0033	0.0063	51.6398

Table 3: Duration of MapReduce implementation measured by `time`, with twelve threads

2.3 Comparison

3 Most appropriate workload for MapReduce

Data that needs sorting?

4 Impact of using multiple machines on execution speed

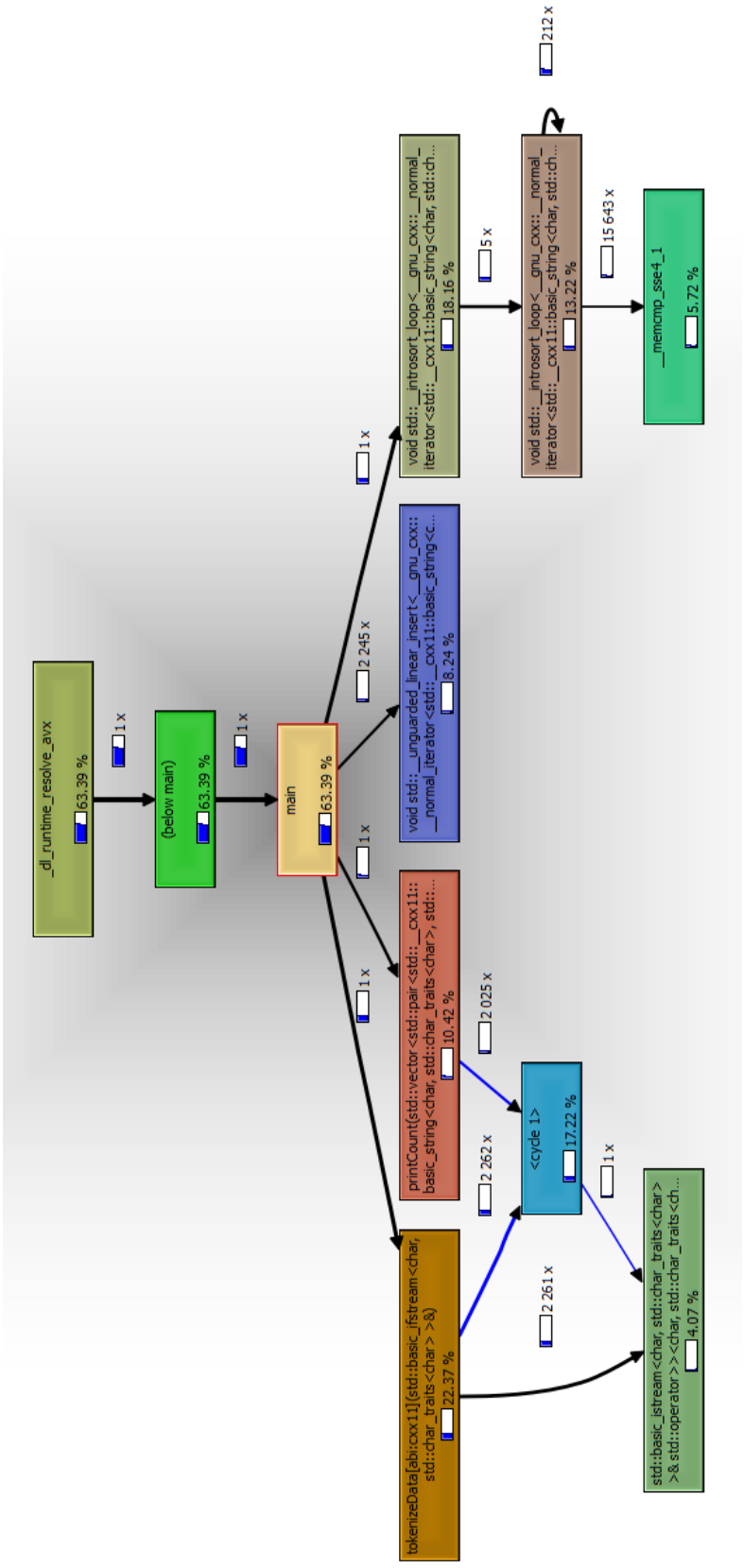

Figure 1: Call map for single-threaded implementation of word count

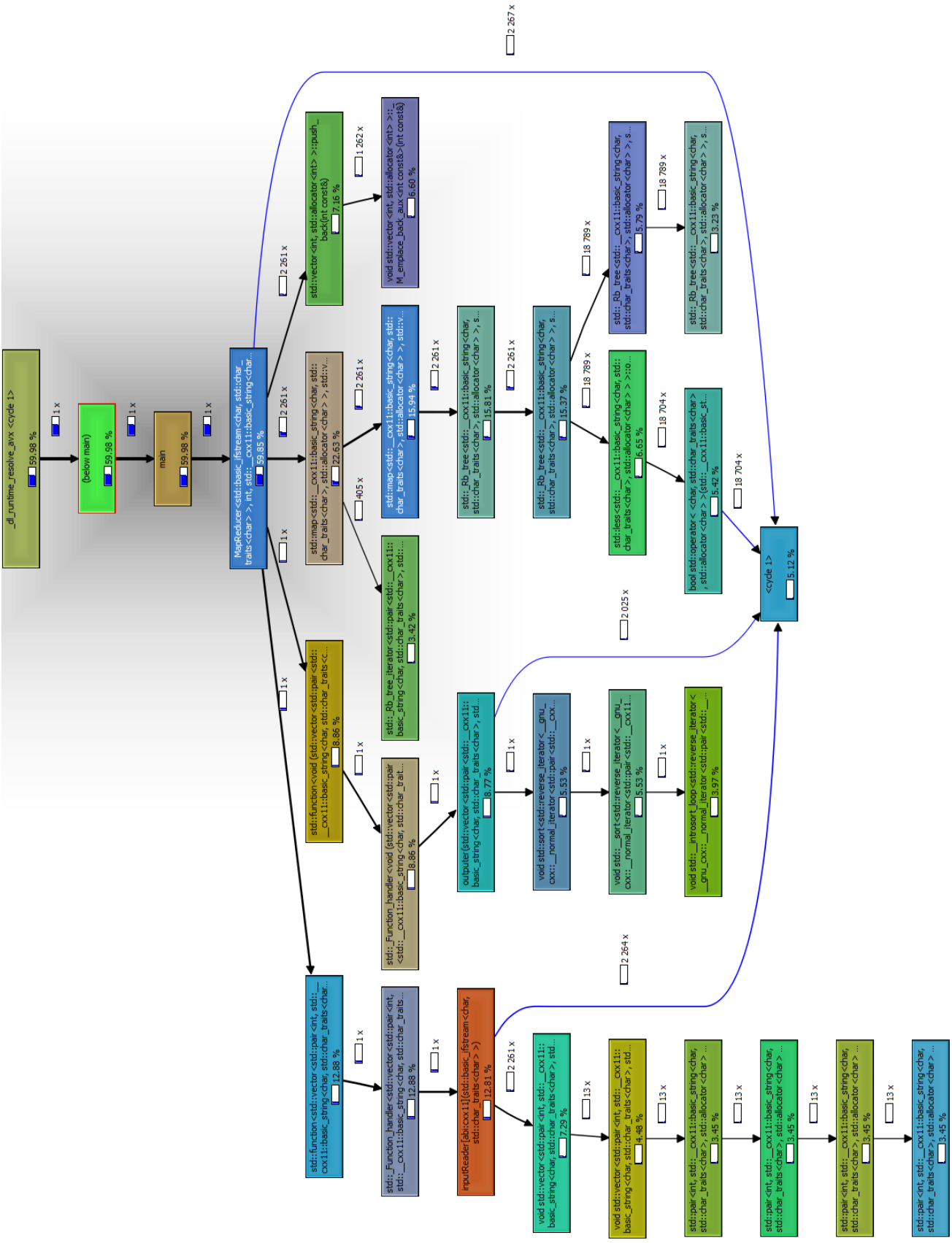

Figure 2: Call map for MapReduce implementation of word count (twelve threads)