# Contents

1	Basic Test Results	2
2	README	4
3	Barrier.h	5
4	Barrier.cpp	6
5	JobContext.h	7
6	JobContext.cpp	9
7	Makefile	15
8	MapReduceFramework.cpp	16

#### 1 Basic Test Results

```
4
    Extracting tar...
6
8
    Checking for required files...
9
    -----
10
11
    Validating users...
12
    Got the following: ['e342791191', 'archsak']
    Make sure the usernames are correct
14
15
16
17
18
    Running 'make' test
19
    Passed basic make test
20
21
22
23
    Running compilation test
24
    Passed basic compile test
    For your convenience, we ran your library and printed the output here.
25
26
        stage 1, 100.00000%
27
        stage 3, 0.000000%
        stage 3, 33.333332%
28
29
        stage 3, 47.619049%
        stage 3, 66.666664%
30
        stage 3, 85.714287%
31
        stage 3, 100.000000%
        Done!
33
34
        The character w appeared 1 time
        The character u appeared 2 times
35
        The character {\tt t} appeared 5 times
36
37
        The character s appeared 7 times
        The character {\tt r} appeared 6 times
38
39
        The character o appeared 4 times \,
40
        The character n appeared 4 times
        The character {\tt m} appeared 1 time
41
42
        The character 1 appeared 3 times
43
        The character i appeared 8 times
        The character h appeared 3 times
44
45
        The character g appeared 2 times
        The character {\tt f} appeared 2 times
46
47
        The character {\tt e} appeared 6 times
        The character d appeared 3 times
        The character c appeared 4 times
49
50
        The character b appeared 1 time
        The character a appeared 7 times
51
        The character {\tt T} appeared 1 time
52
53
        The character M appeared 1 time
54
55
57
    # Passed pre-submit tests #
```

#### 60 ##########################

- Note: There may be more outputs above with errors. Passing this test does not mean your library ran properly.

# 2 README

```
e342791191, archsak
1
    Elsa Sebagh (342791191), Aharon Saksonov (207600164)
2
3
4
   FILES:
   MapReduceFramework.cpp
   JobContext.cpp
    JobContext.h
9 Barrier.cpp
10 Barrier.h
11
   Makefile
12
13
14
   REMARKS:
15
```

### 3 Barrier.h

```
#ifndef BARRIER_H
#define BARRIER_H
2
   #include <pthread.h>
4
    // a multiple use barrier
    class Barrier {
8
    public:
       Barrier(int numThreads);
9
        ~Barrier();
10
11
        void barrier();
12
    private:
13
      pthread_mutex_t mutex;
14
        pthread_cond_t cv;
15
16
        int count;
        int numThreads;
17
    };
18
19
   #endif //BARRIER_H
20
```

### 4 Barrier.cpp

```
#include "Barrier.h"
1
2
    #include <cstdlib>
    #include <cstdio>
3
4
    Barrier::Barrier(int numThreads)
            : mutex(PTHREAD_MUTEX_INITIALIZER)
             , cv(PTHREAD_COND_INITIALIZER)
8
             , count(0)
             , numThreads(numThreads)
9
    { }
11
12
13
    Barrier::~Barrier()
14
         if (pthread_mutex_destroy(&mutex) != 0) {
15
             fprintf(stderr, "[[Barrier]] error on pthread_mutex_destroy");
16
             exit(1);
17
18
         if (pthread_cond_destroy(&cv) != 0){
19
             fprintf(stderr, "[[Barrier]] error on pthread_cond_destroy");
20
21
             exit(1);
22
    }
23
24
25
26
    void Barrier::barrier()
27
         if (pthread_mutex_lock(&mutex) != 0){
28
29
             fprintf(stderr, "[[Barrier]] error on pthread_mutex_lock");
             exit(1);
30
31
32
         if (++count < numThreads) {</pre>
             if (pthread_cond_wait(&cv, &mutex) != 0){
33
                 fprintf(stderr, "[[Barrier]] error on pthread_cond_wait");
34
35
                 exit(1);
             }
36
         } else {
37
             count = 0;
38
             if (pthread_cond_broadcast(&cv) != 0) {
39
40
                 fprintf(stderr, "[[Barrier]] error on pthread_cond_broadcast");
41
                 exit(1);
42
             }
43
         if (pthread_mutex_unlock(&mutex) != 0) {
44
45
             fprintf(stderr, "[[Barrier]] error on pthread_mutex_unlock");
46
             exit(1);
        }
47
    }
48
```

#### 5 JobContext.h

```
//
// Created by user on 21/06/2024.
4
    #ifndef MAPREDUCEFRAMEWORK_JOBCONTEXT_H
    #define MAPREDUCEFRAMEWORK_JOBCONTEXT_H
6
    #include "MapReduceFramework.h"
    #include "Barrier.h"
9
10
   #include <pthread.h>
    #include <iostream>
11
    #include <memoru>
12
   #include <atomic>
    #include <vector>
14
15
    #include <utility>
    #include <set>
16
    #include <algorithm>
17
18
    #include <semaphore.h>
19
    using namespace std;
20
21
    struct ThreadContext;
22
23
    struct K2Comparator {
24
        bool operator()(const K2* key1, const K2* key2) const {
25
26
            return (*key1)<(*key2);
27
    };
28
29
30
31
    class JobContext
32
     public:
33
      JobContext (const MapReduceClient &client, const InputVec &inputVec, OutputVec &outputVec, int multiThreadLevel);
34
      ~JobContext ();
35
      void operator= (const JobContext &other);
36
37
      void waitForJob ();
38
      JobState getJobState ();
39
40
      void addThread (int id);
      InputVec getInputVec ();
41
42
      OutputVec getOutputVec ();
      Barrier* getBarrier ();
43
      long unsigned int getInputLength ();
44
45
      void setJobState (JobState state);
      const MapReduceClient &getClient () const;
46
47
      pthread_mutex_t jobMutex;
48
      pthread_cond_t jobCond;
49
      std::set<K2*, K2Comparator> getUniqueKeySet();
50
      std::vector<IntermediateVec> getIntermediateVectors();
51
      std::vector<IntermediateVec> getShuffledVectors();
52
53
      sem_t* getShuffleSemaphore();
        void insertToShuffledVectors(IntermediateVec
                                                           vectors);
54
55
        void insertToUniqueKeySet(K2* unique_key);
        void insertToIntermediateVectors(IntermediateVec intermediate_vector);
        void insertToOutputVec(K3* key, V3* value);
57
        void setJobFinished();
        int getMultiThreadLevel();
```

```
60
61
62
63
64
      private:
       void setJobState (stage_t stage, float percentage, bool finished = false);
65
       const MapReduceClient &client;
66
       const InputVec &inputVec;
67
68
       OutputVec &outputVec;
       bool isWaitingForJob;
69
       int multiThreadLevel;
70
71
       long unsigned int inputLength;
       std::vector <pthread_t> threads;
72
       std::vector<ThreadContext *> threadContexts;
73
 74
       JobState state;
75
76
       bool jobFinished;
       std::atomic<long unsigned int> atomic;
77
       std::set<K2*, K2Comparator> uniqueKeySet;
78
       std::vector<std::pair<K2*, V2*>>> intermediateVectors;
std::vector<std::pair<K2*, V2*>>> shuffledVectors;
79
80
       Barrier *barrier;
81
       sem_t shuffleSemaphore; // Semaphore to control shuffle synchronization
82
83
     };
84
85
     struct ThreadContext {
86
87
          int threadId;
          std::unique_ptr<std::vector<std::pair<K2*, V2*>>> intermediateVector;
88
89
         std::atomic<long unsigned int>& atomic;
90
         JobContext* jobContext;
91
         ThreadContext(int id, std::atomic<long unsigned int>& atomic, JobContext* jobContext)
92
93
              : threadId(id), intermediateVector(new std::vector<std::pair<K2*,
                                                   V2*>>()), atomic
94
95
                                                   (atomic),
                                                   jobContext(jobContext)
96
                {}
97
98
99
     };
100
     #endif //MAPREDUCEFRAMEWORK_JOBCONTEXT_H
101
```

#### 6 JobContext.cpp

```
#include "JobContext.h"
1
    #include <pthread.h>
   #include <iostream>
   #include <algorithm>
4
    using namespace std;
6
8
    void jobSystemError (string text)
9
      std::cout << "system error: " << text << std::endl;</pre>
10
      exit (1);
11
12
    void *runThread (void *context)
14
15
      auto *castContext = static_cast<ThreadContext *>(context);
16
      auto &jobContext = castContext->jobContext;
17
18
19
                        ----- MAP PHASE -----
20
21
      long unsigned int old_value = castContext->atomic.fetch_add (1);
22
23
24
      while (old_value < jobContext->getInputLength ())
25
26
          If this is the first iteration, set the job state to 0 - we're
27
          entering map stage
        if (old_value == 0)
28
29
          jobContext->setJobState ({MAP_STAGE, 0});
30
31
32
          Map over the input we got
        jobContext->getClient ().map (jobContext->getInputVec ()[old_value].first,
33
34
                                       jobContext->getInputVec ()[old_value].second,
                                       context);
35
          Update state
36
37
        float result = static_cast<float>(100.0f
                                           * static_cast<float>(castContext->atomic.load ())
38
39
40
                                           jobContext->getInputLength ());
        if (castContext->atomic.load ()
41
42
            >= jobContext->getInputLength () - 1)
43
          jobContext->setJobState ({MAP_STAGE, 100.0f});
44
45
46
        else if (old_value < jobContext->getInputLength () - 1)
47
          old_value = castContext->atomic.fetch_add (1);
49
50
          jobContext->setJobState ({MAP_STAGE, result});
51
52
53
54
       * Sorting
55
      if (!castContext->intermediateVector->empty ())
57
58
           Sorting the keys
```

```
60
         std::sort (castContext->intermediateVector->begin (),
                    castContext->intermediateVector->end (),
 61
 62
                     [] (const IntermediatePair &a, const IntermediatePair &b)
 63
                        return *a.first < *b.first;</pre>
 64
                    }):
 65
         jobContext->insertToIntermediateVectors (*(castContext->intermediateVector));
 66
 67
 68
         for (size_t i = 0; i < castContext->intermediateVector->size (); i++)
 69
            jobContext->insertToUniqueKeySet ((*castContext->intermediateVector)[i].first);
 70
 71
         }
 72
 73
       7
 74
       jobContext->getBarrier ()->barrier (); // Wait for everyone
 75
 76
                          ----- SHUFFLE PHASE -----
 77
 78
 79
       if (castContext->threadId == 0)
 80
 81
         castContext->atomic.exchange (0);
 82
         jobContext->setJobState ({SHUFFLE_STAGE, 0});
 83
 84
         auto uniqueKeySet = jobContext->getUniqueKeySet ();
 85
         // Convert the set to a vector
 86
 87
         std::vector < K2 *
         > uniqueKeySetVector (uniqueKeySet.begin (), uniqueKeySet.end ());
 88
 89
 90
         // Sort the vector of pointers
         std::sort (uniqueKeySetVector.begin (), uniqueKeySetVector.end (),
 91
 92
                     [] (const K2 *a, const K2 *b)
 93
                     {
                        return *b < *a:
 94
 95
                    });
         auto intermediateVectors = jobContext->getIntermediateVectors ();
 96
 97
         for (K2 *key: uniqueKeySetVector)
 98
           IntermediateVec key_vector;
 99
           for (auto &vector: intermediateVectors) // Note the use of '&' here
100
101
             while (!vector.empty () && !(*(vector.back ().first) < *key || *key <
102
103
                                                                                  .back ().first)))
104
105
             {
106
               key_vector.push_back (vector.back ());
               vector.pop_back ();
107
108
               castContext->atomic.fetch_add (1);
               float result = static_cast<float>(100.0f
109
                                                  * static_cast<float>(castContext->atomic.load ())
110
111
112
                                                  jobContext->getInputLength ());
               jobContext->setJobState ({SHUFFLE_STAGE, result});
113
             }
114
115
116
           jobContext->insertToShuffledVectors (key_vector);
117
118
119
         jobContext->setJobState ({SHUFFLE_STAGE, 100.0f});
120
         for (int i = 1; i < jobContext->getMultiThreadLevel(); ++i) {
121
           sem_post(jobContext->getShuffleSemaphore());
122
123
         castContext->atomic.exchange (0):
       }
124
125
       else
126
         sem_wait (jobContext->getShuffleSemaphore ()); // Wait until shuffle is done
127
```

```
128
       }
129
                          ----- REDUCE PHASE -----
130
131
       old_value = castContext->atomic.fetch_add (1);
132
133
       while (old_value < jobContext->getShuffledVectors ().size ())
134
135
       {
136
            If this is the first iteration, set the job state to 0 - we're
     //
            entering reduce stage
137
          if (old_value == 0)
138
139
           jobContext->setJobState ({REDUCE_STAGE, 0});
140
141
142
            Reduce over the intermediate we got
          jobContext->getClient ().reduce
143
144
              (&(jobContext->getShuffledVectors ()[old_value]),
145
              context);
            Update state
146
          float result = static_cast<float>(100.0f
147
                                            * static_cast<float>(castContext->atomic.load ())
148
149
                                            jobContext->getShuffledVectors ().size ());
150
151
          if (castContext->atomic.load ()
152
              >= jobContext->getShuffledVectors ().size () - 1)
153
            jobContext->setJobState ({REDUCE_STAGE, 100.0f});
154
155
            jobContext->setJobFinished();
            break;
156
157
158
          else if (old_value < jobContext->getShuffledVectors ().size () - 1)
159
160
            old_value = castContext->atomic.fetch_add (1);
161
            jobContext->setJobState ({REDUCE_STAGE, result});
162
163
       }
164
       return nullptr;
     }
165
166
     JobContext::JobContext (const MapReduceClient &client, const InputVec &inputVec,
167
168
                              OutputVec &outputVec, int multiThreadLevel)
          : client (client) ,inputVec (inputVec), outputVec (outputVec), isWaitingForJob(false),
169
           multiThreadLevel (multiThreadLevel), state ({UNDEFINED_STAGE, 0}),
170
171
            jobFinished (false), atomic (0)
172
173
       pthread_mutex_init (&jobMutex, nullptr);
174
       pthread_cond_init (&jobCond, nullptr);
       inputLength = inputVec.size ();
175
176
       barrier = new Barrier (multiThreadLevel);
       sem_init (&shuffleSemaphore, 0, 0); // Initialize semaphore with value 0
177
178
179
     // \it{TODO} Save this in bits somehow
180
       for (int i = 0; i < multiThreadLevel; i++)</pre>
181
          addThread (i);
182
183
184
     }
185
186
187
     JobContext::~JobContext ()
188
189
       for (auto context: threadContexts)
190
          context->intermediateVector->clear ();
191
192
         delete context;
193
       for (auto it = threads.begin (); it != threads.end ();)
194
195
```

```
{\tt pthread\_cancel\ (*it);} \ \ /\!/\ {\it Cancel\ the\ thread}
196
197
          // Erase the thread from the vector and advance the iterator
198
         it = threads.erase (it);
199
       pthread_mutex_destroy (&jobMutex);
200
201
       pthread_cond_destroy (&jobCond);
       sem_destroy (&shuffleSemaphore);
202
       delete barrier;
203
204
     //void JobContext::operator= (const JobContext &other)
205
     //{
206
207
     // client = other.client;
     // inputVec = other.inputVec;
208
     // outputVec = other.outputVec;
209
210
     // multiThreadLevel = other.multiThreadLevel;
     // jobFinished = other.jobFinished;
211
     // atomic_length = other.atomic_length;
212
     // inputLength = other.inputLength;
213
     // state = other.state;
214
     // threads = other.threads;
215
216
     // threadContexts = other.threadContexts;
     // jobMutex = other.jobMutex;
217
     // jobCond = other.jobCond;
218
     1/3
219
220
221
     void JobContext::waitForJob ()
222
     // if(isWaitingForJob) return;
223
       while (!jobFinished)
224
225
226
         pthread_cond_wait(&jobCond, &jobMutex);
227
228
229
       for(int i = 0; i < multiThreadLevel; i++){</pre>
         if(pthread_join(threads[i], nullptr) != 0){
230
231
           jobSystemError ("Error joining thread");
232
233
       isWaitingForJob = true;
^{234}
235
     }
236
237
238
239
     JobState JobContext::getJobState ()
240
       if (state.stage == MAP_STAGE)
241
242
         float result = static_cast<float>(atomic) / inputLength * 100.0f;
243
244
         if (result > 100) result = 100.0f;
245
         state.percentage = result;
246
^{247}
       return state;
248
249
     void JobContext::addThread (int id)
250
251
252
       pthread_t thread;
       auto *context = new ThreadContext (id, atomic, this);
253
254
255
       pthread_attr_t attr;
256
       pthread_attr_init (&attr);
257
       if (pthread_create (&thread, &attr, runThread, static_cast<void *>
258
        (context)) != 0)
259
         jobSystemError ("Could not create thread");
260
261
       pthread_attr_destroy (&attr);
262
263
       threadContexts.push_back (context);
```

```
264
       threads.push_back (thread);
265
266
267
     void JobContext::insertToShuffledVectors (IntermediateVec vectors)
268
269
       pthread_mutex_lock (&jobMutex);
       shuffledVectors.push_back (vectors);
270
       pthread_cond_broadcast (&jobCond);
271
272
       pthread_mutex_unlock (&jobMutex);
273
274
275
     InputVec JobContext::getInputVec ()
276
277
       return inputVec;
278
279
280
     OutputVec JobContext::getOutputVec ()
281
       return outputVec;
282
283
284
     long unsigned int JobContext::getInputLength ()
285
286
     {
287
       return inputLength;
288
289
     const MapReduceClient &JobContext::getClient () const
290
291
       return client;
292
293
294
     Barrier *JobContext::getBarrier ()
295
296
297
       return barrier;
298
299
     std::vector <IntermediateVec> JobContext::getShuffledVectors ()
300
301
       return shuffledVectors;
302
     }
303
304
     std::vector <IntermediateVec> JobContext::getIntermediateVectors ()
305
306
307
       return intermediateVectors;
308
309
310
     void JobContext::setJobState (JobState state)
311
312
       pthread_mutex_lock (&jobMutex);
313
       this->state = state;
       pthread_cond_broadcast (&jobCond);
314
315
       pthread_mutex_unlock (&jobMutex);
316
317
     void JobContext::insertToUniqueKeySet (K2 *uniqueKey)
318
319
320
       pthread_mutex_lock (&jobMutex);
       uniqueKeySet.insert (uniqueKey);
321
       pthread_cond_broadcast (&jobCond);
322
323
       pthread_mutex_unlock (&jobMutex);
324
325
326
     std::set<K2 *, K2Comparator> JobContext::getUniqueKeySet ()
327
328
       return uniqueKeySet;
329
330
331
     sem_t *JobContext::getShuffleSemaphore ()
```

```
332
       return &shuffleSemaphore;
333
334
335
336
     void
     JobContext::insertToIntermediateVectors (IntermediateVec intermediateVector)
337
338
       pthread_mutex_lock (&jobMutex);
339
340
        \verb|intermediateVectors.push_back| (\verb|intermediateVector|); \\
       pthread_cond_broadcast (&jobCond);
341
       pthread_mutex_unlock (&jobMutex);
342
343
344
     void JobContext::insertToOutputVec (K3 *key, V3 *value)
345
346
       pthread_mutex_lock (&jobMutex);
347
        outputVec.push_back (std::make_pair (key, value));
348
       pthread_cond_broadcast (&jobCond);
349
     pthread_mutex_unlock (&jobMutex);
}
350
351
352
     void JobContext::setJobFinished ()
353
354
       {\tt pthread\_mutex\_lock~(\&jobMutex);}
355
356
        jobFinished = true;
       pthread_cond_broadcast (&jobCond);
357
       pthread_mutex_unlock (&jobMutex);
358
359
360
361
     int JobContext::getMultiThreadLevel ()
362
       return multiThreadLevel;
363
     }
364
```

#### 7 Makefile

```
CC=g++
1
2
    CXX=g++
   RANLIB=ranlib
4
   LIBSRC=MapReduceFramework.cpp JobContext.cpp JobContext.h Barrier.cpp Barrier.h Makefile
   LIBOBJ=$(LIBSRC:.cpp=.o)
6
8
    INCS=-I.
   CFLAGS = -Wall -std=c++11 -g $(INCS)
9
   CXXFLAGS = -Wall -std=c++11 -g $(INCS)
11
   MAPREDUCELIB = libMapReduceFramework.a
12
   TARGETS = $(MAPREDUCELIB)
14
    TAR=tar
15
16
   TARFLAGS=-cvf
    TARNAME=ex3.tar
17
    TARSRCS=$(LIBSRC) Makefile README
18
19
   all: $(TARGETS)
20
21
    $(TARGETS): $(LIBOBJ)
22
        $(AR) $(ARFLAGS) $@ $^
23
24
        $(RANLIB) $@
25
26
27
        $(RM) $(TARGETS) $(OSMLIB) $(OBJ) $(LIBOBJ) *~ *core
28
29
        makedepend -- $(CFLAGS) -- $(SRC) $(LIBSRC)
30
31
        $(TAR) $(TARFLAGS) $(TARNAME) $(TARSRCS)
33
```

# 8 MapReduceFramework.cpp

```
#include <pthread.h>
    #include "MapReduceFramework.h"
   #include "JobContext.h"
    #include <iostream>
    #include <vector>
    #include <string>
    #include <map>
9
    using namespace std;
10
    std::map<JobHandle, JobContext *> jobs;
11
12
    void systemError (string text)
14
      std::cout << "system error: " << text << std::endl;</pre>
15
16
      exit (1);
17
    // Context is whatever we want it to be - we get it from map, and we're the
19
    // ones to call map.
20
21
    // Should probably contain the input, output and intermediate pointers
    void emit2 (K2 *key, V2 *value, void *context)
22
23
    // Add key and value to the intermediate vector of the calling thread
24
25
26
      auto *castContext = static_cast<ThreadContext *>(context);
27
      pthread_mutex_lock (&castContext->jobContext->jobMutex);
      castContext->intermediateVector->push_back (std::make_pair (key,
28
29
                                                                    value));
      pthread_mutex_unlock (&castContext->jobContext->jobMutex);
30
31
33
    void emit3 (K3 *key, V3 *value, void *context)
34
35
        auto* castContext = static_cast<ThreadContext*>(context);
36
37
        castContext->jobContext->insertToOutputVec (key, value);
38
39
40
    JobHandle startMapReduceJob (const MapReduceClient &client,
41
42
                                  const InputVec &inputVec, OutputVec &outputVec,
                                  int multiThreadLevel)
43
44
45
      JobContext *job = new JobContext (client, inputVec, outputVec,
                                         multiThreadLevel);
46
      JobHandle jobHandle = new JobHandle ();
47
      jobs[jobHandle] = job;
48
      return jobHandle;
49
50
51
    void waitForJob (JobHandle job)
52
53
      if (jobs.find (job) == jobs.end ())
54
55
        systemError ("Job not found");
57
      jobs[job]->waitForJob ();
58
```

```
60
61
     void getJobState (JobHandle job, JobState *state)
62
     auto it = jobs.find (job);
if (it == jobs.end ())
63
64
65
66
       systemError ("Job not found");
67
68
       *state = it->second->getJobState ();
    }
69
70
     void closeJobHandle (JobHandle job)
71
72
    {
      {
73
           std::lock_guard<std::mutex> lock(jobsMutex);
74
        auto it = jobs.find (job);
if (it == jobs.end ())
75
76
77
         {
          systemError ("Job not found");
78
79
80
         waitForJob (job);
         delete it->second;
81
82
         jobs.erase (it);
83
    }
84
85
86
```