FYP JC2104
Presentation

Imsong Jeon 1155087995
8 December 2021

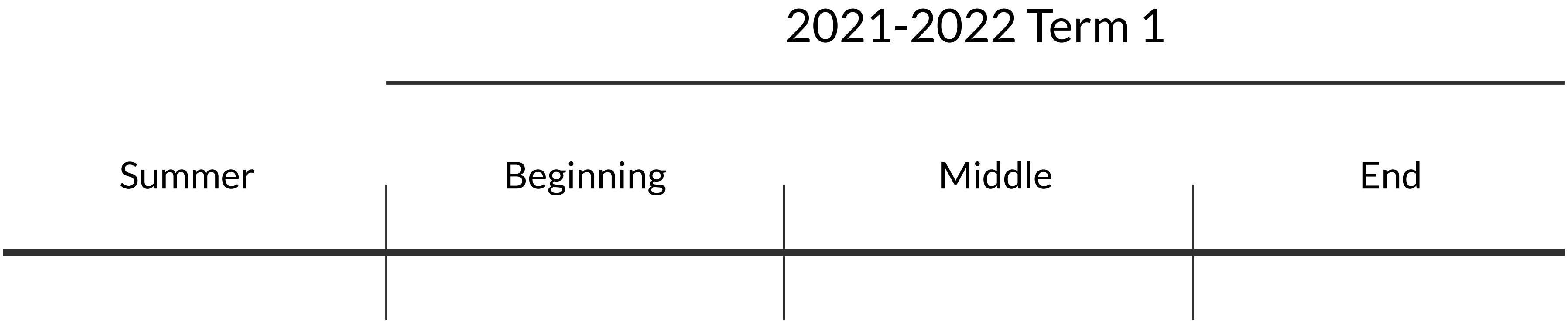
1. Objectives
2. Process
3. Results
4. Plans and Ideas
5. Conclusion

1. Objectives

FYP JC2104: Open Topics on Distributed Systems for Data Analytics

1. Self-Learning
2. Defining Problem and Solution

2. Process



2021-2022 Term 1

Summer

Beginning

Middle

End

2. Process

Summer

APACHE
hadoop

APACHE
Spark™

Beginning

Ursa

- C++
- CMake
- Cluster and Job Configuration

2. Process

Middle

Example Codes

hello_world.cc

pi.cc

word_count.cc

kmeans.cc

...

from close point of view:

- High-level APIs
- Source Code

from far point of view:

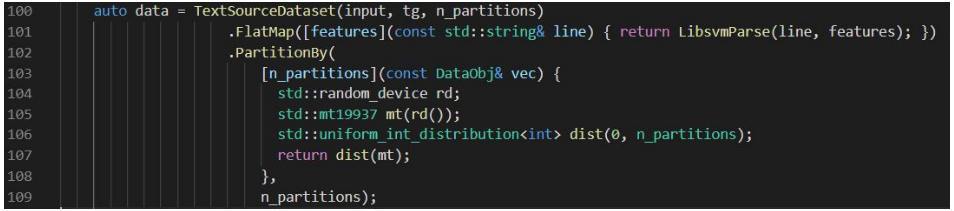
- Workflow
- Structure

2. Process

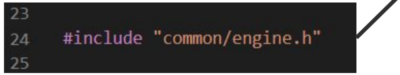
Middle

from close point of view:

- High-level APIs
- Source Code

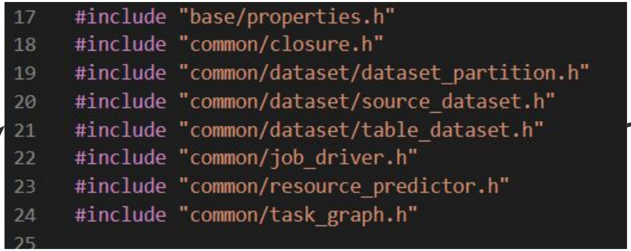


```
auto data = TextSourceDataset(input, tg, n_partitions)
                .FlatMap([features](const std::string& line) { return LibsvmParse(line, features); })
                .PartitionBy(
                    [n_partitions](const DataObj& vec) {
                      std::random_device rd;
                      std::mt19937 mt(rd());
                      std::uniform_int_distribution<int> dist(0, n_partitions);
                      return dist(mt);
                    },
                    n_partitions);
```



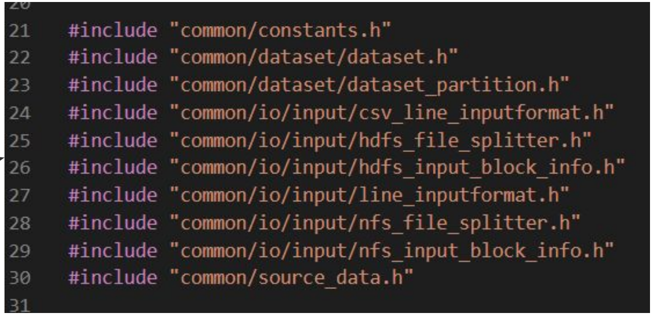
```
#include "common/engine.h"
```

kmeans.cc



```
#include "base/properties.h"
#include "common/closure.h"
#include "common/dataset/dataset_partition.h"
#include "common/dataset/source_dataset.h"
#include "common/dataset/table_dataset.h"
#include "common/job_driver.h"
#include "common/resource_predictor.h"
#include "common/task_graph.h"
```

common/engine.h



```
#include "common/constants.h"
#include "common/dataset/dataset.h"
#include "common/dataset/dataset_partition.h"
#include "common/io/input/csv_line_inputformat.h"
#include "common/io/input/hdfs_file_splitter.h"
#include "common/io/input/hdfs_input_block_info.h"
#include "common/io/input/line_inputformat.h"
#include "common/io/input/nfs_file_splitter.h"
#include "common/io/input/nfs_input_block_info.h"
#include "common/source_data.h"
```

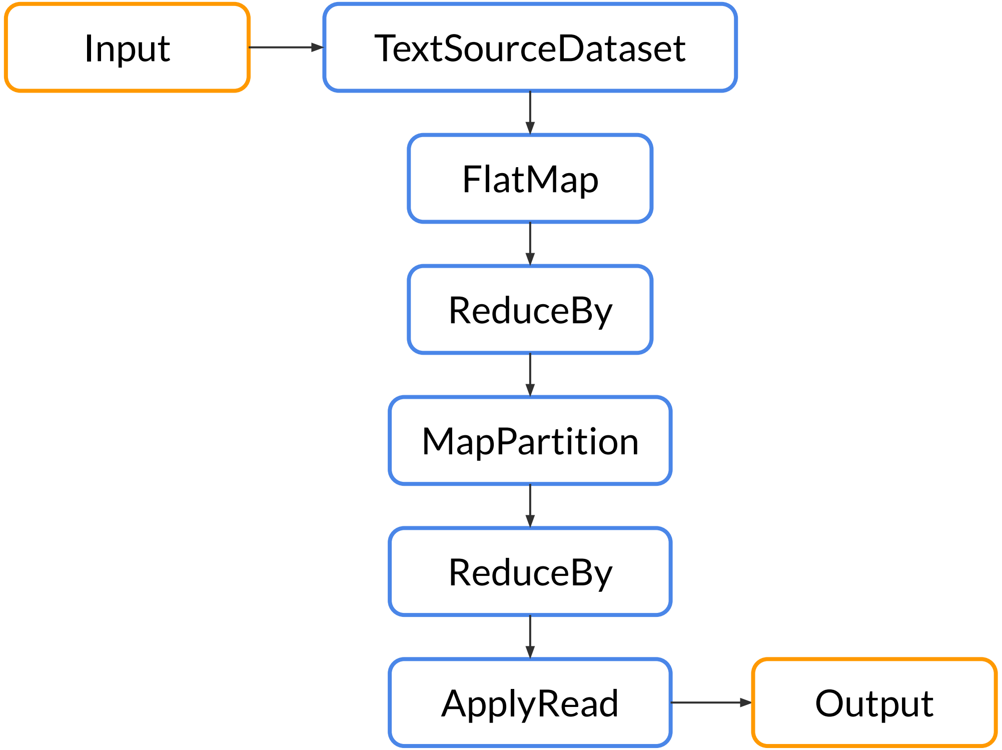
common/dataset/source_dataset.h

2. Process

Middle

from far point of view:

- Workflow (DAG)
- Overall Structure



Workflow of word_count.cc

2. Process

End

More Example Codes

Implementations

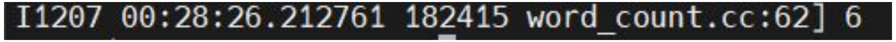
- Word Length Count
- Parallel Dijkstra

3. Results

Word Length Count

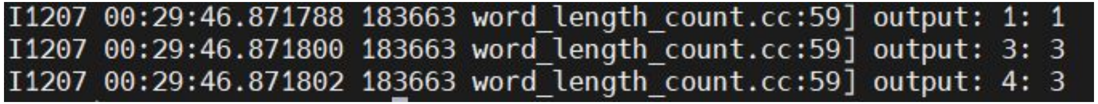
Input:
“I have two cats and two dogs”

Word Count output:



```
I1207 00:28:26.212761 182415 word_count.cc:62] 6
```

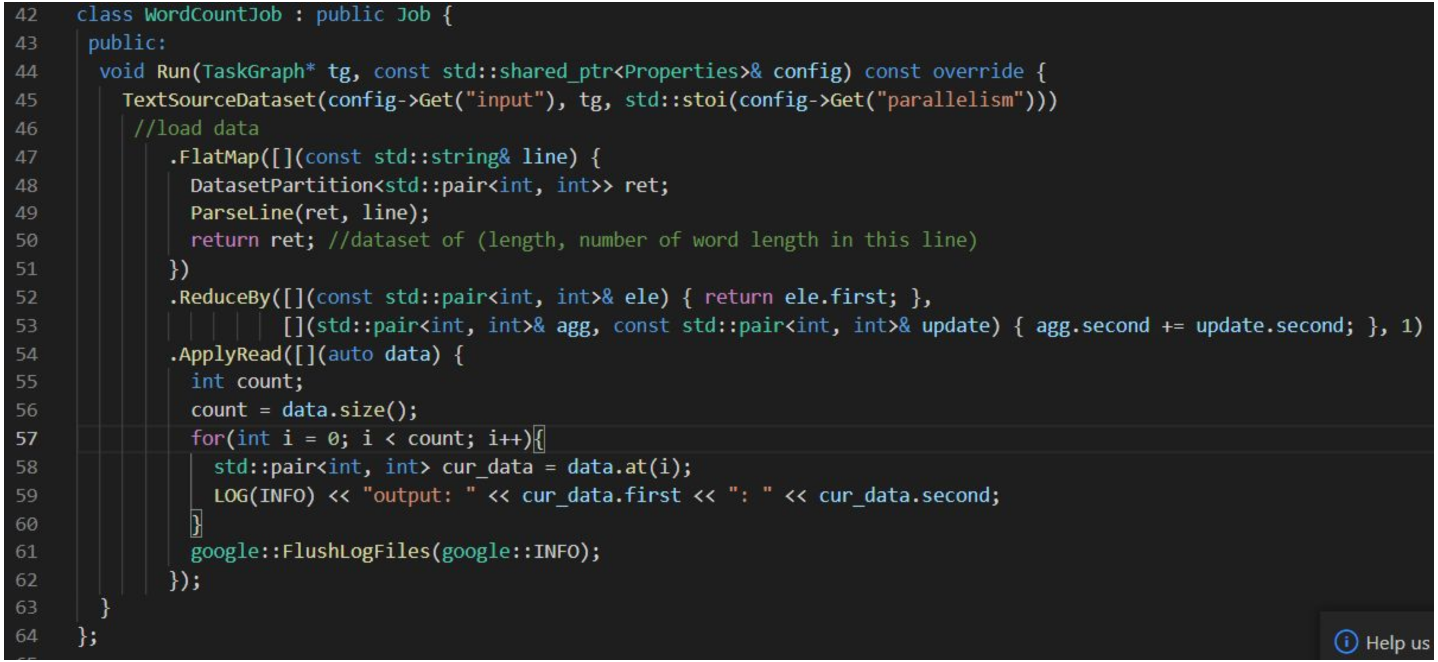
Word Length Count output:



```
I1207 00:29:46.871788 183663 word_length_count.cc:59] output: 1: 1
I1207 00:29:46.871800 183663 word_length_count.cc:59] output: 3: 3
I1207 00:29:46.871802 183663 word_length_count.cc:59] output: 4: 3
```

3. Results

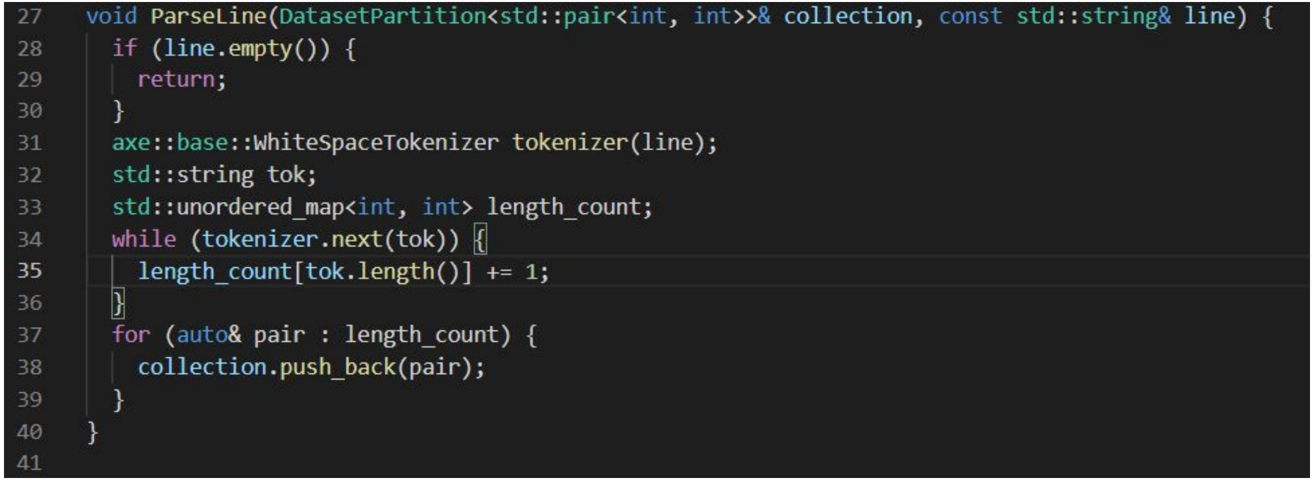
Word Length Count/Run



```cpp
class WordCountJob : public Job {
 public:
  void Run(TaskGraph* tg, const std::shared_ptr<Properties>& config) const override {
    TextSourceDataset(config->Get("input"), tg, std::stoi(config->Get("parallelism")))
     //load data
        .FlatMap([](const std::string& line) {
          DatasetPartition<std::pair<int, int>> ret;
          ParseLine(ret, line);
          return ret; //dataset of (length, number of word length in this line)
        })
        .ReduceBy([](const std::pair<int, int>& ele) { return ele.first; },
                  [](std::pair<int, int>& agg, const std::pair<int, int>& update) { agg.second += update.second; }, 1)
        .ApplyRead([](auto data) {
          int count;
          count = data.size();
          for(int i = 0; i < count; i++){
            std::pair<int, int> cur_data = data.at(i);
            LOG(INFO) << "output: " << cur_data.first << ": " << cur_data.second;
          }
          google::FlushLogFiles(google::INFO);
        });
  }
};
```

3. Results

Word Length Count/ParseLi ne

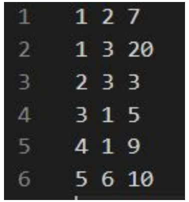


```cpp
void ParseLine(DatasetPartition<std::pair<int, int>>& collection, const std::string& line) {
  if (line.empty()) {
    return;
  }
  axe::base::WhiteSpaceTokenizer tokenizer(line);
  std::string tok;
  std::unordered_map<int, int> length_count;
  while (tokenizer.next(tok)) {
    length_count[tok.length()] += 1;
  }
  for (auto& pair : length_count) {
    collection.push_back(pair);
  }
}
```

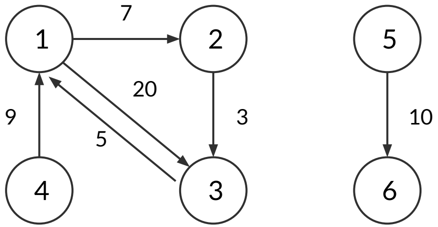
3. Results

Parallel Dijkstra

Input:

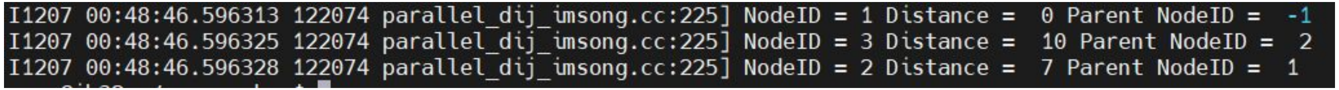


```
1 2 7
1 3 20
2 3 3
3 1 5
4 1 9
5 6 10
```



Configuration:
n-iterations: 2
start node: 1

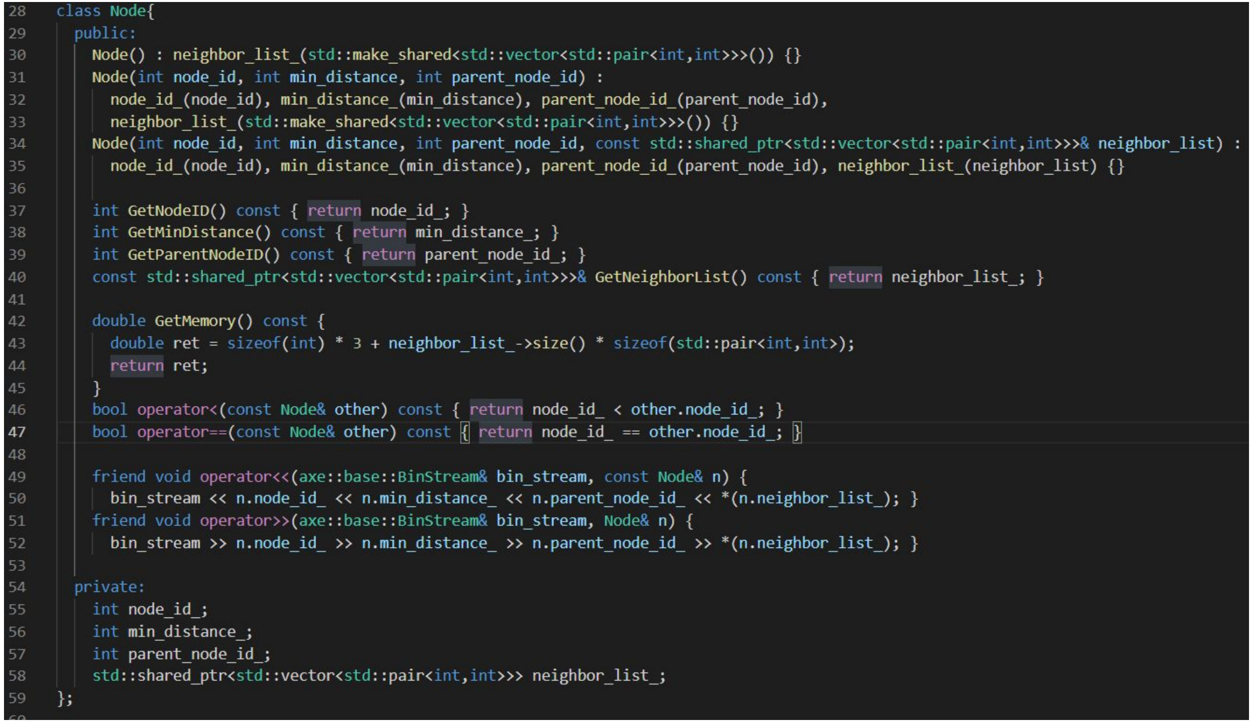
Output:



```
I1207 00:48:46.596313 122074 parallel_dij_umsong.cc:225] NodeID = 1 Distance =  0 Parent NodeID =  -1
I1207 00:48:46.596325 122074 parallel_dij_umsong.cc:225] NodeID = 3 Distance =  10 Parent NodeID =  2
I1207 00:48:46.596328 122074 parallel_dij_umsong.cc:225] NodeID = 2 Distance =  7 Parent NodeID =  1
```

3. Results

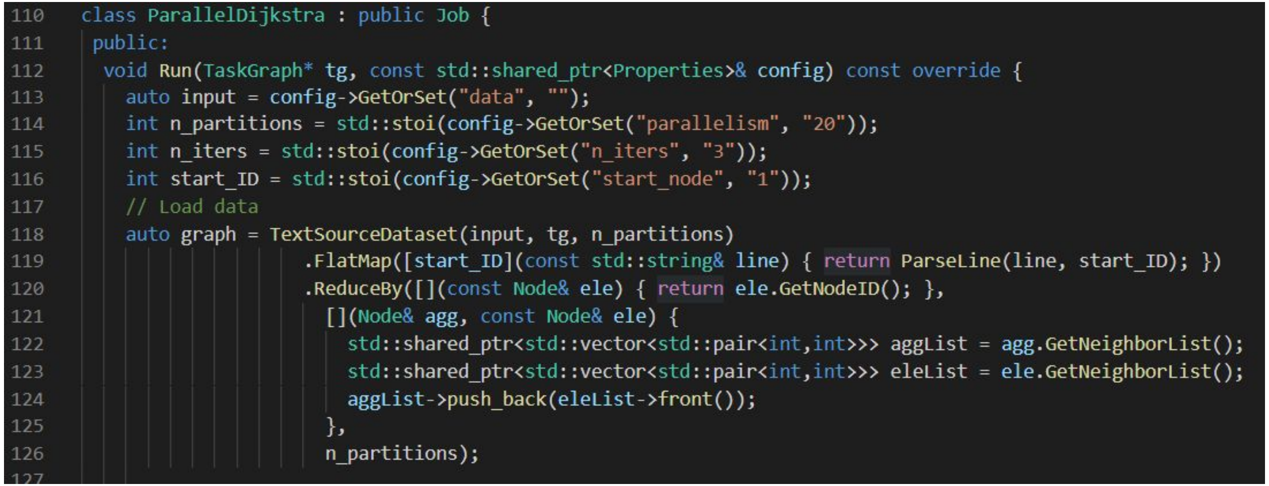
Parallel Dijkstra/Node class



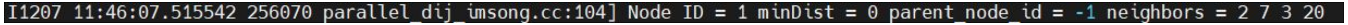
```cpp
class Node{
  public:
    Node() : neighbor_list_(std::make_shared<std::vector<std::pair<int,int>>>()) {}
    Node(int node_id, int min_distance, int parent_node_id) :
      node_id_(node_id), min_distance_(min_distance), parent_node_id_(parent_node_id),
      neighbor_list_(std::make_shared<std::vector<std::pair<int,int>>>()) {}
    Node(int node_id, int min_distance, int parent_node_id, const std::shared_ptr<std::vector<std::pair<int,int>>>& neighbor_list) :
      node_id_(node_id), min_distance_(min_distance), parent_node_id_(parent_node_id), neighbor_list_(neighbor_list) {}

    int GetNodeID() const { return node_id_; }
    int GetMinDistance() const { return min_distance_; }
    int GetParentNodeID() const { return parent_node_id_; }
    const std::shared_ptr<std::vector<std::pair<int,int>>>& GetNeighborList() const { return neighbor_list_; }

    double GetMemory() const {
      double ret = sizeof(int) * 3 + neighbor_list_->size() * sizeof(std::pair<int,int>);
      return ret;
    }
    bool operator<(const Node& other) const { return node_id_ < other.node_id_; }
    bool operator==(const Node& other) const { return node_id_ == other.node_id_; }

    friend void operator<<(axe::base::BinStream& bin_stream, const Node& n) {
      bin_stream << n.node_id_ << n.min_distance_ << n.parent_node_id_ << *(n.neighbor_list_); }
    friend void operator>>(axe::base::BinStream& bin_stream, Node& n) {
      bin_stream >> n.node_id_ >> n.min_distance_ >> n.parent_node_id_ >> *(n.neighbor_list_); }

  private:
    int node_id_;
    int min_distance_;
    int parent_node_id_;
    std::shared_ptr<std::vector<std::pair<int,int>>> neighbor_list_;
};
```

3. Results

Parallel Dijkstra/Run (part 1)



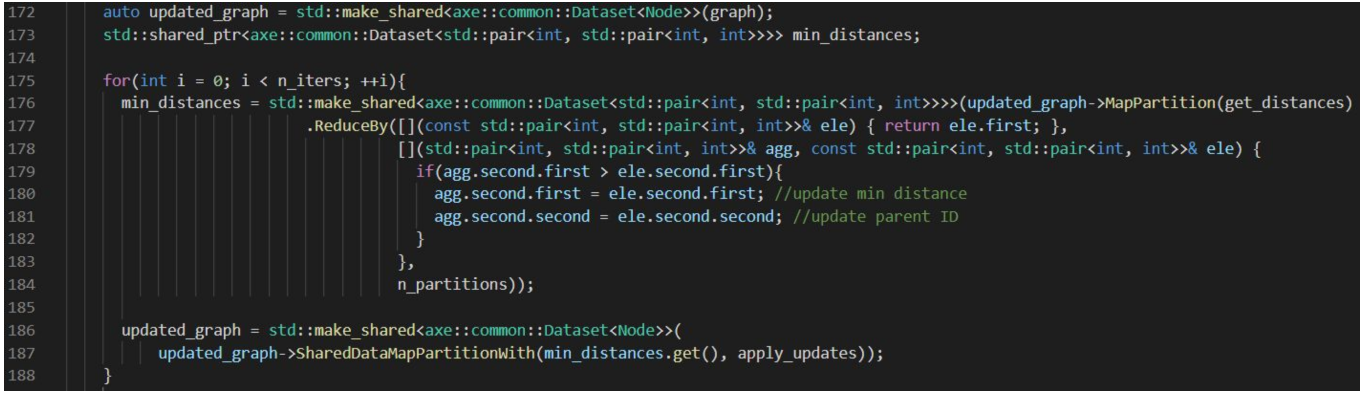
```cpp
class ParallelDijkstra : public Job {
 public:
  void Run(TaskGraph* tg, const std::shared_ptr<Properties>& config) const override {
    auto input = config->GetOrSet("data", "");
    int n_partitions = std::stoi(config->GetOrSet("parallelism", "20"));
    int n_iters = std::stoi(config->GetOrSet("n_iters", "3"));
    int start_ID = std::stoi(config->GetOrSet("start_node", "1"));
    // Load data
    auto graph = TextSourceDataset(input, tg, n_partitions)
                    .FlatMap([start_ID](const std::string& line) { return ParseLine(line, start_ID); })
                    .ReduceBy([](const Node& ele) { return ele.GetNodeID(); },
                      [](Node& agg, const Node& ele) {
                        std::shared_ptr<std::vector<std::pair<int,int>>> aggList = agg.GetNeighborList();
                        std::shared_ptr<std::vector<std::pair<int,int>>> eleList = ele.GetNeighborList();
                        aggList->push_back(eleList->front());
                      },
                      n_partitions);
```



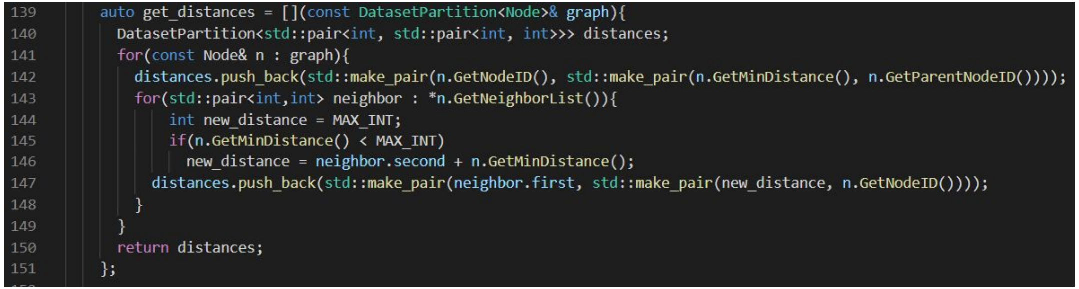
```
I1207 11:46:07.515542 256070 parallel_dij_imsong.cc:104] Node ID = 1 minDist = 0 parent_node_id = -1 neighbors = 2 7 3 20
```

3. Results

Parallel Dijkstra/Run (part 2)



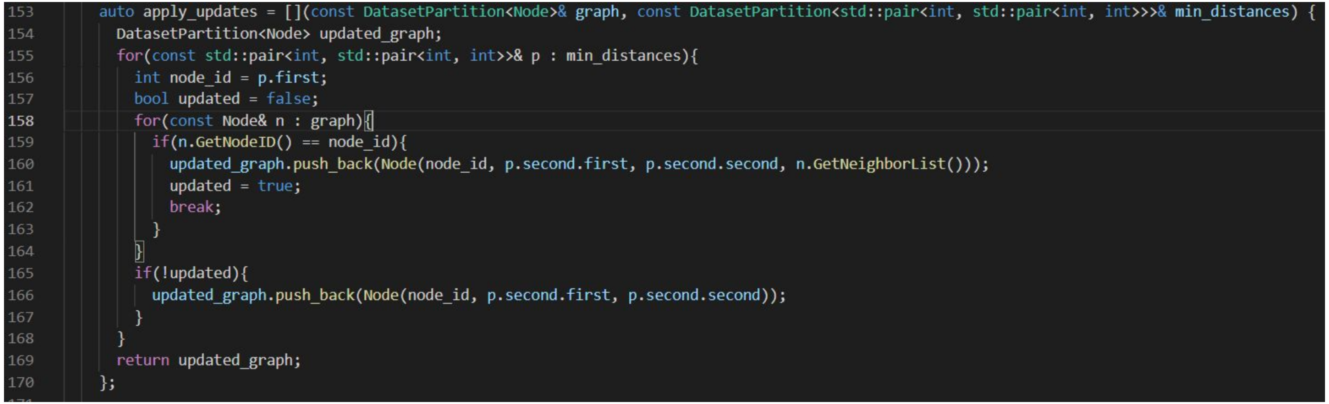
```cpp
auto updated_graph = std::make_shared<axe::common::Dataset<Node>>(graph);
std::shared_ptr<axe::common::Dataset<std::pair<int, std::pair<int, int>>>> min_distances;

for(int i = 0; i < n_iters; ++i){
  min_distances = std::make_shared<axe::common::Dataset<std::pair<int, std::pair<int, int>>>>(updated_graph->MapPartition(get_distances)
                      .ReduceBy([](const std::pair<int, std::pair<int, int>>& ele) { return ele.first; },
                                [](std::pair<int, std::pair<int, int>>& agg, const std::pair<int, std::pair<int, int>>& ele) {
                                  if(agg.second.first > ele.second.first){
                                    agg.second.first = ele.second.first; //update min distance
                                    agg.second.second = ele.second.second; //update parent ID
                                  }
                                },
                                n_partitions));

  updated_graph = std::make_shared<axe::common::Dataset<Node>>(
      updated_graph->SharedDataMapPartitionWith(min_distances.get(), apply_updates));
}
```



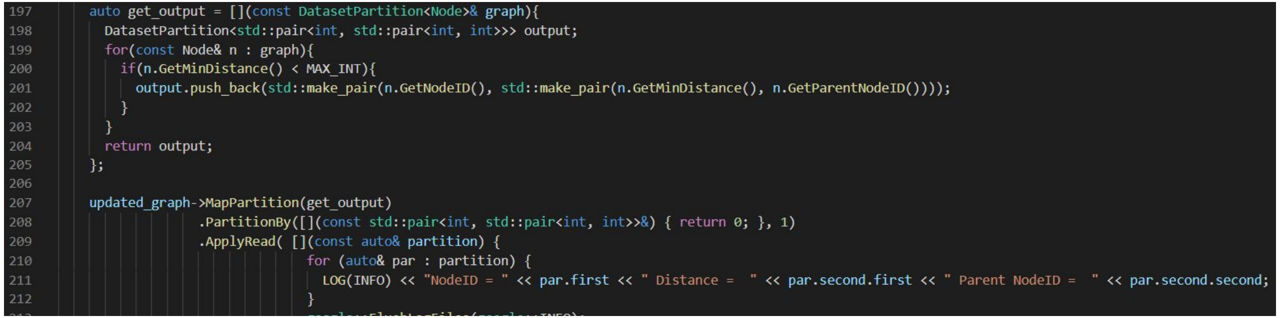
```cpp
auto get_distances = [](const DatasetPartition<Node>& graph){
  DatasetPartition<std::pair<int, std::pair<int, int>>> distances;
  for(const Node& n : graph){
    distances.push_back(std::make_pair(n.GetNodeID(), std::make_pair(n.GetMinDistance(), n.GetParentNodeID())));
    for(std::pair<int,int> neighbor : *n.GetNeighborList()){
        int new_distance = MAX_INT;
        if(n.GetMinDistance() < MAX_INT)
          new_distance = neighbor.second + n.GetMinDistance();
      distances.push_back(std::make_pair(neighbor.first, std::make_pair(new_distance, n.GetNodeID())));
    }
  }
  return distances;
};
```

3. Results

Parallel Dijkstra/Run (part 3)



```
auto apply_updates = [](const DatasetPartition<Node>& graph, const DatasetPartition<std::pair<int, std::pair<int, int>>>& min_distances) {
  DatasetPartition<Node> updated_graph;
  for(const std::pair<int, std::pair<int, int>>& p : min_distances){
    int node_id = p.first;
    bool updated = false;
    for(const Node& n : graph){
      if(n.GetNodeID() == node_id){
        updated_graph.push_back(Node(node_id, p.second.first, p.second.second, n.GetNeighborList()));
        updated = true;
        break;
      }
    }
    if(!updated){
      updated_graph.push_back(Node(node_id, p.second.first, p.second.second));
    }
  }
  return updated_graph;
};
```



```
auto get_output = [](const DatasetPartition<Node>& graph){
  DatasetPartition<std::pair<int, std::pair<int, int>>> output;
  for(const Node& n : graph){
    if(n.GetMinDistance() < MAX_INT){
      output.push_back(std::make_pair(n.GetNodeID(), std::make_pair(n.GetMinDistance(), n.GetParentNodeID())));
    }
  }
  return output;
};

updated_graph->MapPartition(get_output)
              .PartitionBy([](const std::pair<int, std::pair<int, int>>&) { return 0; }, 1)
              .ApplyRead( [](const auto& partition) {
                            for (auto& par : partition) {
                              LOG(INFO) << "NodeID = " << par.first << " Distance =  " << par.second.first << " Parent NodeID =  " << par.second.second;
                            }
```

4. Plans and Ideas

Plans:

- Implement and test with larger data and more workers
- More study
- Distributed PCA (Principal Component Analysis)

Ideas

- Linear Algebra Package
- Real-time interaction

5. Conclusion

Thank you!