一些总结小技巧和计数器以及代码

划分函数

按map函数输出的key决定当前的kv被划分到哪一个中间文件中。 工程经验上看,我们有时不仅仅要求其平衡划分，我们还可能要求其可以按照某种kv的元数据信息进行分区。 比如将一个主机下的文件划分到一起等定制的能力

顺序保证

有时我们需要支持对数据文件按key进行随机访问,或者有序输出,并且为了减少reduce任务的负担。 输出的每个map任务的中间文件都是保证按key递增有序的

输入输出类型

有时map 、reduce 函数的输入类型并不是纯文本的，这种情况下如果能输入输出结构化类型是一种理想的情况。这需要在map reduce函数之前实现一个类型适配器的组件,同时也可以实现read接口不仅仅从文件中读取数据也可以从数据库等数据源来读取。

状态信息

分布式系统的可观测性尤为重要, 通过在ping-pong过程中将每个worker节点的工作状态进行上报,存储在main进程中，并提供可访问的网页来展示系统运行的信息。即可实现可解释性。

计数器

如何在map reduce的处理进行埋点统计？以实现用户自定义的指标监控? 需要创建一个原子计数器的对象,由用户在map 和 reduce的函数中 在发生某些事件时 对其进行累加。 并通过ping-pong的心跳中将数据携带回 main进程 累加进去，但要注意每次消息的幂等性来保证不会导致重复累加或者少累加了计数的情况。

#include "mapreduce/mapreduce.h"

// User's map fuction

class WordCounter: public Mapper {

public:

virtual void Map(const MapInput &input) {

const string &text = input.value();

const **int** n = text.size();

for (**int** i = 0; i < n; ) {

// Skip past leading whitespace

while ((i < n) && **isspace**(text[i]))

++i;

// Find word end

**int** start = i;

while ((i < n) && !**isspace**(text[i]))

++i;

if (start < i)

Emit(text.substr(start, i-start), "1");

}

}

};

REGISTER\_MAPPER(WordCounter);

// User's reduce function

class Adder: public Reducer {

virtual void Reduce(ReduceInput \*input) {

// Iterate over all entries with the

// same key and add the values

int64\_t value = 0;

while (!input->done()) {

value += StringToInt(input->value());

input->NextValue();

}

// Emit sum for input->key()

Emit(IntToString(value));

}

};

REGISTER\_REDUCER(Adder);

**int** main(**int** argc, **char** \*\*argv) {

ParseCommandLineFlags(argc, argv);

MapReduceSpecification spec;

// Store list of input files into "spec"

for (**int** i = 1; i < argc; ++i) {

MapReduceInput \*input = spec.add\_input();

input->set\_format("text");

input->set\_filepattern(argv[i]);

input->set\_mapper\_class("WordCounter");

}

// Specify the output files:

// /gfs/test/freq-00000-of-00100

// /gfs/test/freq-00001-of-00100

// ...

MapReduceOutput \*out = spec.output();

out->set\_filebase("/gfs/test/freq");

out->set\_num\_tasks(100);

out->set\_format("text");

out->set\_reducer\_class("Adder");

// Optional: do partial sums within map

// tasks to save network bandwidth

out->set\_combine\_class("Adder");

// Tuning parameters: use at most 2000

// machines and 100MB of memory per task

spec.set\_machines(2000);

spec.set\_map\_megabytes(100);

spec.set\_reduce\_megabytes(100);

// Now run it

MapReduceResult result;

if (!MapReduce(spec, &result))

**abort**();

// Done: 'result' structure contains info

// about counters, time taken, number of

// machines used, etc.

return 0;

}