2021华为软件精英挑战赛（杭厦第20名）

原创

weixin_38616018 于 2021-04-20 17:33:08 发布 960 收藏 11

分类专栏： 算法 文章标签： 算法

本文链接：https://blog.csdn.net/weixin_38616018/article/details/115914316

算法 专栏收录该内容

3 篇文章 0 订阅

订阅专栏

写在前面

距离华为软件精英挑战赛结束也有一段时间了

我是浙工大投降战队的队长，第一次参加这种比赛能打到复赛我还是比较满意的

这次比赛我最大的收获就是认识了好多厉害的大佬

希望我们杭厦赛区晋级的战队总决赛能拿到好成绩

也希望比赛结束了能交流一下的思路，扩展一下思维。

github

- https://github.com/william970/HUAWEI-CodeCraft-2021

成绩

杭厦赛区，复赛第20名

区域初赛 | 区域复赛 | 全球总决赛

在线训练 | 正式赛 | 在线训练 | 正式赛 | 在线训练

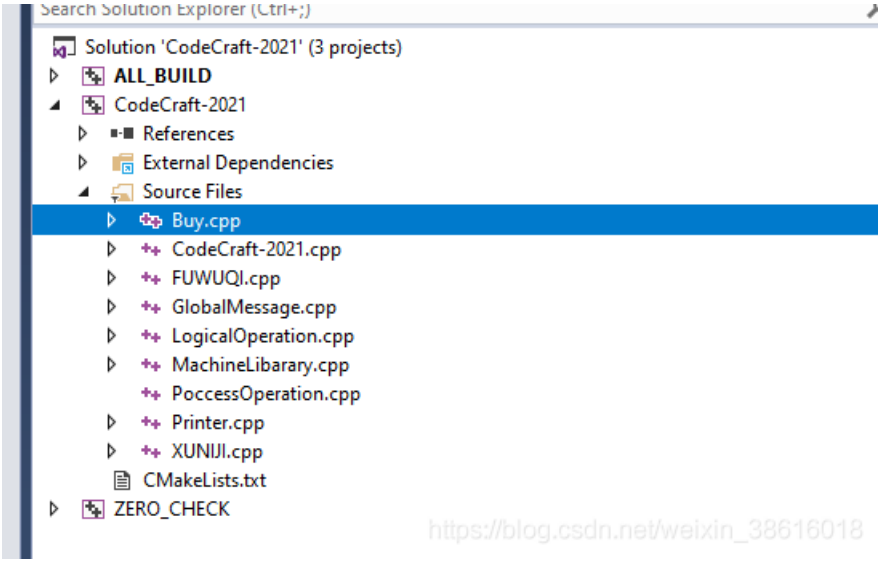
我的排名	团队名	总成本	迁移次数	运行时间	提交时间
20↑ 2	浙工大投降队	1653772069	198707	93.625	2021/04/11

排名	团队名	总成本	迁移次数	运行时间	提交时间
1	抬杠团团	1527714895	665172	111.848	2021/04/11
2	好想下班	1541496788	682855	43.383	2021/04/11
3↑ 1	零零妖	1541932781	662496	12.406	2021/04/11
4↓ 1	MXS	1542291158	542271	84.015	2021/04/11

大致思路

这个比赛，能优化的地方一共就三个地方，一个是购买服务器的策略，还有一个是插入虚拟机到服务器的策略，最后是迁移策略

代码结构



这个是整体的代码结构，最后因为编译问题，全部整合到了一个cpp里面，这样看着清晰一点

- GlobalMessage：单例模式，这里面存放了所有全局的信息，读进来的信息以及根据该信息做解析之后的信息都存放在这里
- fuwuqi，xuniji，logicalOperation：一些基础的数据结构还有添加删除操作
- Buy里面写了服务器购买逻辑
- poccessOperation：流程函数
- Printer：输出类
- MachineLibarary：单例模式，里面放了所有的机器的操作函数，比如添加服务器，增加虚拟机，删除虚拟机，以及为虚拟机分配一台服务器

算法

我们的算法还是比较朴实的，没有用到什么高级算法

算法主要分三块

1. 购买服务器（参考buy.cpp）

```
FUWUQI buyOptimalMachine2(const XUNIJI &xuniji);
```

3. 为虚拟机分配服务器（参考MachineLibarary里）

```
pair<MachineLibarary::AB, std::shared_ptr<FUWUQI> > addXuNiJi(const string& id, XUNIJI xuniji, unordered_set<shared_ptr<FUWUQI>> &unused_set, unordered_set<shared_ptr<FUWUQI>> &in_it); //添加虚拟机,新增排除列表
```

3. 迁移（参考MachineLibarary里的迁移函数）

```
vector<pair<string, pair<MachineLibarary::AB, std::shared_ptr<FUWUQI>> > > migaration(); //迁移
```

购买服务器

采用贪心算法，考虑了服务器的内存内核比和性价比为每台服务器打分，最后选取分数最高的服务器

优化点

- 虚拟机的内存内核比 用未来的所有还未分配的虚拟机取代当前虚拟机，加入全局的特征
- 优先选取内存内核能存放下请求序列里面最大的虚拟机的服务器，在代码中体现为提高该服务器的分数，因为有些小服务器后面空了，有一天大的虚拟机来时，如果放不下且没有合适的服务器就又要去买一台，这样很亏
- 同时考虑了内存内核比还有性价比

将虚拟机插入服务器

我设计的滑窗法
即服务器一个释放日期，来一台虚拟机的时候，优先插入释放日期大于等于虚拟机释放日期的服务器
如果不存在，则选取一台释放日期与虚拟机释放日期最接近的合适服务器，并把该服务器的释放日期赋值为该虚拟机的释放日期
然后还不存在，再选一台性价比最高的空的服务器插入

优化点

- 滑窗保证释放日期接近的服务器尽量放在同一台服务器上
- 把成本量化了，只要保证相同的资源下得到的利益最大化就行了，即相同成本下能放置的虚拟机多，内存内核利用率最大
- 保证AB节点平衡
- 考虑了插入时服务器的内存内核比，同时在一批内存内核比合适的服务器里面找剩余空间最小的

迁移

找利用率最低的服务器，将该服务器里面的虚拟机迁移到合适的服务器，迁移这一块主要是优化了速度，因为正式赛的时候一直超时（气哭），经过两次优化后速度快了两倍

优化点

- 我们考虑了时间复杂度，认为算法耗时主要是在迁移选服务器的阶段，我们采用分区的方法区内迁移，如果我分8个区的话那需要遍历的服务器就是以前的1/8，考虑到线上服务器是2核的，我们750个服务器作为一个分区，设计了4个线程去处理1/4的分区
- 做了减枝，即如果内核内存利用率大于1.9就直接跳过该服务器，不迁移该服务器下的虚拟机

代码

购买服务器

```
//为服务器打分
double Buy::xingJiaBiAlgorithm(const FUWUQI& fuwuqi, double memory_core_rate, int end_day) {
 //分子：看比例，是哪边失衡了，就按资源小的那个来算，例如，如果虚拟机内存核比2：1，
 //  现在有一个机器是90内存，核数5的，总天数10，当前天数6，购买价格10000，每日耗能500,性价比公式就是(5 + 0.2 * (90 / (2:1)
 - 5)) / (10000 + (10 - 6) * 500)
 //          其中0.2是chao_can1，是一个可调整的超参
 //先看比例是哪边大一些
 double fuwuqi_memory_core_rate = fuwuqi.restNeicunNeiheRate();
 int blance = 1.0;
 if (fuwuqi.neicun > GlobalMessage::Get().max_memory && fuwuqi.neihe > GlobalMessage::Get().max_core) {
  blance = 2.5;
 }
 if (fuwuqi_memory_core_rate > memory_core_rate) {
  return  blance * ((double)fuwuqi.neihe + BILI * (memory_core_rate / fuwuqi_memory_core_rate) * (((double)fuwuq
i.neicun / memory_core_rate) - fuwuqi.neihe)) /
   (fuwuqi.cost + 1.0 * fuwuqi.costOneDay * ((double)end_day - GlobalMessage::Get().current_Day));
 }
 return blance * ((double)fuwuqi.neicun / memory_core_rate + BILI * (fuwuqi_memory_core_rate / memory_core_rate)
 * ((double)fuwuqi.neihe - (double)fuwuqi.neicun / memory_core_rate)) /
  (fuwuqi.cost + 1.0 * fuwuqi.costOneDay * ((double)end_day - GlobalMessage::Get().current_Day));
}

//选取打分最高的服务器
FUWUQI Buy::buyOptimalMachine2(const XUNIJI& xuniji) {
 //1.1策略：根据没有分配的虚拟机决定性价比参数;
 int current_day = GlobalMessage::Get().current_Day;
 double memory = (double)xuniji.neicun * xuniji.continue_day + GlobalMessage::Get().all_unadd_memory;
 double core = (double)xuniji.neihe * xuniji.continue_day + GlobalMessage::Get().all_unadd_core;
 double xuniji_memory_core_rate = 1.0 * memory / core;
 double max_xingjiabi = -1;//最大性价比
 FUWUQI res;
 for (FUWUQI& p : GlobalMessage::Get().fuwuqi_vec) {
  if (xuniji.shuangjiedian == true && (xuniji.neihe > p.neihe || xuniji.neicun > p.neicun)) continue;//无法满足要
求
  if (xuniji.shuangjiedian == false && (xuniji.neihe > p.neihe / 2 || xuniji.neicun > p.neicun / 2)) continue;//
无法满足要求
  double x = xingJiaBiAlgorithm(p, xuniji_memory_core_rate, xuniji.end_day);
  if (x > max_xingjiabi) {
   max_xingjiabi = x;
   res = p;
  }
 }
 return res;
}
```

插入虚拟机到服务器

```
pair<MachineLibarary::AB, shared_ptr<FUWUQI>> MachineLibarary::findOptimalFuWuQi_charu(XUNIJI & xuniji)
{
 double memory = xuniji.neicun + GlobalMessage::Get().all_unadd_memory;
 double core = xuniji.neihe + GlobalMessage::Get().all_unadd_core;
 double xuniji_future_memory_core_rate_balance = 1.0 * memory / core;
 double xuniji_memory_core_rate = (double)xuniji.neicun / (double)xuniji.neihe;
 pair<MachineLibarary::AB, shared_ptr<FUWUQI>> best_choice;//选取的最优机器
 int currentDay = GlobalMessage::Get().current_Day; //当前日期
 //遍历所有机器，找比例最接近的节点
 if (xuniji.shuangjiedian == false) {
  //priority_queue < shared_ptr<FUWUQI>, vector<shared_ptr<FUWUQI>>,> min_dui;
  double diff_min_1 = 100000;
  double diff_min_2 = 100000;
```

```
double diff_min_2 = 100000;
//单节点
//unused_set.empty 表示插入 else 表示迁移
list<std::shared_ptr<FUWUQI>>::iterator fuwuqi_ptr;
for (fuwuqi_ptr = fuwuqi_list.begin(); fuwuqi_ptr != fuwuqi_list.end(); fuwuqi_ptr++) {
 if (!(*fuwuqi_ptr)->isUsed()) {
  continue;//优先选正在使用的服务器
 }
 if ((*fuwuqi_ptr)->notUseDay <= currentDay) {
  continue;//已经不使用了
 }
 //先从服务器闲置日期>=虚拟机释放日期的服务器中分配
 if ((*fuwuqi_ptr)->notUseDay >= xuniji.end_day) {
  int shi_heng = (*fuwuqi_ptr)->shiHeng();//shi_heng表示是不是A比B多太多了？要保持AB的平衡
  double memory_core_rateA = (*fuwuqi_ptr)->restNeicunNeiheRateA();
  double memory_core_rateB = (*fuwuqi_ptr)->restNeicunNeiheRateB();
  //如果A空间不足，就只判断B，如果B空间不足，就只判断A
  if ((*fuwuqi_ptr)->restNeiheA() < xuniji.neihe || (*fuwuqi_ptr)->restNeicunA() < xuniji.neicun) {
   shi_heng = 1;
  }
  if ((*fuwuqi_ptr)->restNeiheB() < xuniji.neihe || (*fuwuqi_ptr)->restNeicunB() < xuniji.neicun) {
   shi_heng = 0;
  }
  if (shi_heng == 0) {
   //如果A使用太少了，就选取A
   //判断是否有剩余空间
   if ((*fuwuqi_ptr)->restNeiheA() < xuniji.neihe || (*fuwuqi_ptr)->restNeicunA() < xuniji.neicun) {
    continue;
   }
   double temp = abs(memory_core_rateA - xuniji_memory_core_rate);
   double temp2 = (double)(*fuwuqi_ptr)->restNeiheA() / xuniji.neihe + (double)(*fuwuqi_ptr)->restNeicunA() /
xuniji.neicun;
   if (temp < (diff_min_1 - MIN_DIFF)) {
    diff_min_1 = temp;
    diff_min_2 = temp2;
    best_choice = make_pair(AB::A, (*fuwuqi_ptr));
   }
   else if (temp < (diff_min_1 + MIN_DIFF)) {
    if (temp2 < diff_min_2) {
     diff_min_1 = temp;
     diff_min_2 = temp2;
     best_choice = make_pair(AB::A, (*fuwuqi_ptr));
    }
   }
  }
  else if (shi_heng == 1) {
   //选取B
   if ((*fuwuqi_ptr)->restNeiheB() < xuniji.neihe || (*fuwuqi_ptr)->restNeicunB() < xuniji.neicun) {
    continue;
   }
   double temp = abs(memory_core_rateB - xuniji_memory_core_rate);
   double temp2 = (double)(*fuwuqi_ptr)->restNeiheB() / xuniji.neihe + (double)(*fuwuqi_ptr)->restNeicunB() /
xuniji.neicun;
   if (temp < (diff_min_1 - MIN_DIFF)) {
    diff_min_1 = temp;
    diff_min_2 = temp2;
    best_choice = make_pair(AB::B, (*fuwuqi_ptr));
   }
   else if (temp < (diff_min_1 + MIN_DIFF)) {
    if (temp2 < diff_min_2) {
```

```
      diff_min_1 = temp;
      diff_min_2 = temp2;
      best_choice = make_pair(AB::B, (*fuwuqi_ptr));
     }
    }
   }
   else {
    //A和B都参与竞选
    double tempA = abs(memory_core_rateA - xuniji_memory_core_rate);
    double tempA2 = (double)(*fuwuqi_ptr)->restNeiheA() / xuniji.neihe + (double)(*fuwuqi_ptr)->restNeicunA() /
xuniji.neicun;
    if (tempA < (diff_min_1 - MIN_DIFF)) {
     diff_min_1 = tempA;
     diff_min_2 = tempA2;
     best_choice = make_pair(AB::A, (*fuwuqi_ptr));
    }
    else if (tempA < (diff_min_1 + MIN_DIFF)) {
     if (tempA2 < diff_min_2) {
      diff_min_1 = tempA;
      diff_min_2 = tempA2;
      best_choice = make_pair(AB::A, (*fuwuqi_ptr));
     }
    }
    double tempB = abs(memory_core_rateB - xuniji_memory_core_rate);
    double tempB2 = (double)(*fuwuqi_ptr)->restNeiheB() / xuniji.neihe + (double)(*fuwuqi_ptr)->restNeicunB() /
xuniji.neicun;
    if (tempB < (diff_min_1 - MIN_DIFF)) {
     diff_min_1 = tempB;
     diff_min_2 = tempB2;
     best_choice = make_pair(AB::B, (*fuwuqi_ptr));
    }
    else if (tempB < (diff_min_1 + MIN_DIFF)) {
     if (tempB2 < diff_min_2) {
      diff_min_1 = tempB;
      diff_min_2 = tempB2;
      best_choice = make_pair(AB::B, (*fuwuqi_ptr));
     }
    }
   }
  }
 }
 //如果没有找到
 if (best_choice.second == nullptr) {
  map<int, std::shared_ptr<FUWUQI>> fuwuqi_mp;


  list<std::shared_ptr<FUWUQI>>::iterator fuwuqi_ptr;
  for (fuwuqi_ptr = fuwuqi_list.begin(); fuwuqi_ptr != fuwuqi_list.end(); fuwuqi_ptr++) {

   if (!(*fuwuqi_ptr)->isUsed()) {
    continue;//优先选正在使用的服务器
   }
   if ((*fuwuqi_ptr)->notUseDay <= currentDay) {
    continue;//已经不使用了
   }
   if ((*fuwuqi_ptr)->notUseDay < xuniji.end_day) {
    fuwuqi_mp[(*fuwuqi_ptr)->notUseDay] = *fuwuqi_ptr;
   }
  }
```

```
  map<int, std::shared_ptr<FUWUQI>>::reverse_iterator riter = fuwuqi_mp.rbegin();
  for (riter; riter != fuwuqi_mp.rend(); riter++) {
   int shi_heng = riter->second->shiHeng();
   if (riter->second->restNeiheA() < xuniji.neihe || riter->second->restNeicunA() < xuniji.neicun) {
    shi_heng = 1;
   }
   if (riter->second->restNeiheB() < xuniji.neihe || riter->second->restNeicunB() < xuniji.neicun) {
    shi_heng = 0;
   }
   if (shi_heng == 0) {
    //如果A使用太少了，就选取A
    //判断是否有剩余空间
    if (riter->second->restNeiheA() < xuniji.neihe || riter->second->restNeicunA() < xuniji.neicun) {
     continue;
    }
    riter->second->notUseDay = xuniji.end_day;
    best_choice = make_pair(AB::A, riter->second);
    break;
   }
   else if (shi_heng == 1) {
    //选取B
    if (riter->second->restNeiheB() < xuniji.neihe || riter->second->restNeicunB() < xuniji.neicun) {
     continue;
    }
    riter->second->notUseDay = xuniji.end_day;
    best_choice = make_pair(AB::B, riter->second);
    break;
   }
   else {
    riter->second->notUseDay = xuniji.end_day;
    if (riter->second->restNeiheA() > riter->second->restNeiheB()) {
     best_choice = make_pair(AB::A, riter->second);
    }
    else {
     best_choice = make_pair(AB::B, riter->second);
    }
    break;
   }
  }
 }
 //如果还没有找到,那就分配没有使用的服务器

 if (best_choice.second == nullptr) {
  double xingjiabi = -1;

  list<std::shared_ptr<FUWUQI>>::iterator fuwuqi_ptr;
  for (fuwuqi_ptr = fuwuqi_list.begin(); fuwuqi_ptr != fuwuqi_list.end(); fuwuqi_ptr++) {
   if ((*fuwuqi_ptr)->isUsed()) {
    continue;//选没有使用的服务器
   }
   if ((*fuwuqi_ptr)->restNeiheA() < xuniji.neihe || (*fuwuqi_ptr)->restNeicunA() < xuniji.neicun) {
    continue;
   }
   if ((*fuwuqi_ptr)->restNeiheB() < xuniji.neihe || (*fuwuqi_ptr)->restNeicunB() < xuniji.neicun) {
    continue;
   }
   if ((xuniji_future_memory_core_rate_balance * (*fuwuqi_ptr)->neihe + (*fuwuqi_ptr)->neicun) / (((double)xuni
ji.end_day - currentDay) * ((*fuwuqi_ptr)->costOneDay)) > xingjiabi) {
    xingjiabi = (xuniji_future_memory_core_rate_balance * (*fuwuqi_ptr)->neihe + (*fuwuqi_ptr)->neicun) / (((do
uble)xuniji.end_day - currentDay) * ((*fuwuqi_ptr)->costOneDay));
```

```
     (*fuwuqi_ptr)->notUseDay = xuniji.end_day;
     best_choice = make_pair(AB::A, *fuwuqi_ptr);
    }
   }
  }
 }
 else {
  //双节点,策略是选AB加起来比例最接近的,且差不多的
  int target_memory = xuniji.neicun / 2;
  int target_core = xuniji.neihe / 2;
  double diff_min_1 = 100000.0; //比例距离
  double diff_min_2 = 100000.0;
  list<std::shared_ptr<FUWUQI>>::iterator fuwuqi_ptr;
  for (fuwuqi_ptr = fuwuqi_list.begin(); fuwuqi_ptr != fuwuqi_list.end(); fuwuqi_ptr++) {
   if (!(*fuwuqi_ptr)->isUsed()) {
    continue;//优先选正在使用的服务器
   }
   if ((*fuwuqi_ptr)->notUseDay <= currentDay) {
    continue;//已经不使用了
   }
   //先淘汰内存不足的
   if ((*fuwuqi_ptr)->restNeiheA() < target_core || (*fuwuqi_ptr)->restNeiheB() < target_core ||
    (*fuwuqi_ptr)->restNeicunA() < target_memory || (*fuwuqi_ptr)->restNeicunB() < target_memory) {
    continue;
   }
   if ((*fuwuqi_ptr)->notUseDay >= xuniji.end_day) {
    double memory_core_rate = (*fuwuqi_ptr)->restNeicunNeiheRate();
    double temp = abs(memory_core_rate - xuniji_memory_core_rate);
    double temp2 = (double)(*fuwuqi_ptr)->restNeihe() / xuniji.neihe + (double)(*fuwuqi_ptr)->restNeicun() / xun
iji.neicun;
    if (temp < (diff_min_1 - MIN_DIFF)) {
     diff_min_1 = temp;
     diff_min_2 = temp2;
     best_choice = make_pair(AB::AB, *fuwuqi_ptr);
    }
    else if (temp < (diff_min_1 + MIN_DIFF)) {
     if (((double)(*fuwuqi_ptr)->restNeihe() / xuniji.neihe + (double)(*fuwuqi_ptr)->restNeicun() / xuniji.neicu
n) < diff_min_2) {
      diff_min_1 = temp;
      diff_min_2 = temp2;
      best_choice = make_pair(AB::AB, *fuwuqi_ptr);
     }
    }
   }
  }

  if (best_choice.second == nullptr) {
   map<int, std::shared_ptr<FUWUQI>> fuwuqi_mp;
   list<std::shared_ptr<FUWUQI>>::iterator fuwuqi_ptr;
   for (fuwuqi_ptr = fuwuqi_list.begin(); fuwuqi_ptr != fuwuqi_list.end(); fuwuqi_ptr++) {
    if (!(*fuwuqi_ptr)->isUsed()) {
     continue;//优先选正在使用的服务器
    }
    if ((*fuwuqi_ptr)->notUseDay <= currentDay) {
     continue;//已经不使用了
    }
    if ((*fuwuqi_ptr)->notUseDay < xuniji.end_day) {
     fuwuqi_mp[(*fuwuqi_ptr)->notUseDay] = (*fuwuqi_ptr);
    }
```

```
    }
    map<int, std::shared_ptr<FUWUQI>>::reverse_iterator riter = fuwuqi_mp.rbegin();//从后面往前面插
    for (riter; riter != fuwuqi_mp.rend(); riter++) {
     //先淘汰内存不足的
     if (riter->second->restNeiheA() < target_core || riter->second->restNeiheB() < target_core ||
      riter->second->restNeicunA() < target_memory || riter->second->restNeicunB() < target_memory) {
      continue;
     }
     riter->second->notUseDay = xuniji.end_day;
     best_choice = make_pair(AB::AB, riter->second);
     break;
    }
   }
   //如果还没有找到,那就分配没有使用的服务器
   if (best_choice.second == nullptr) {
    double xingjiabi = -1;
    list<std::shared_ptr<FUWUQI>>::iterator fuwuqi_ptr;
    for (fuwuqi_ptr = fuwuqi_list.begin(); fuwuqi_ptr != fuwuqi_list.end(); fuwuqi_ptr++) {
     if ((*fuwuqi_ptr)->isUsed()) {
      continue;//选没有使用的服务器
     }
     if ((*fuwuqi_ptr)->restNeiheA() < xuniji.neihe || (*fuwuqi_ptr)->restNeicunA() < xuniji.neicun) {
      continue;
     }
     if ((*fuwuqi_ptr)->restNeiheB() < xuniji.neihe || (*fuwuqi_ptr)->restNeicunB() < xuniji.neicun) {
      continue;
     }
     double temp = (xuniji_future_memory_core_rate_balance * (*fuwuqi_ptr)->neihe + (*fuwuqi_ptr)->neicun) / (xun
iji.end_day - currentDay) * ((*fuwuqi_ptr)->costOneDay);
     if (temp > xingjiabi) {
      xingjiabi = temp;
      (*fuwuqi_ptr)->notUseDay = xuniji.end_day;
      best_choice = make_pair(AB::AB, *fuwuqi_ptr);
     }
    }
   }
  }
  return best_choice;
}
```

迁移

```
vector<vector<pair<string, pair<MachineLibarary::AB, std::shared_ptr<FUWUQI>>>>> qianyi_fenqu;

void run_qianyi(int beg, int end, int fenqu_max_count, vector<vector<shared_ptr<FUWUQI>>>& all_fuwuqi_fenqu, vec
tor<unordered_set<shared_ptr<FUWUQI>>>& all_fuwuqi_fenqu_set) {
 //double be = clock();
 for (int i = beg; i < end; i++) {
  if (all_fuwuqi_fenqu[i].size() <= 1) {
   //cout << "小于1" << endl;
   break;
  }
  unordered_set<shared_ptr<FUWUQI>> v;//已经迁移过的服务器
  unordered_set<shared_ptr<FUWUQI>> v2;//不用迁移的服务器
  int fengqucount = 0;
  while (fengqucount < fenqu_max_count) {
   double min_size = 100000;
   shared_ptr<FUWUQI> min_fuwuqi_ptr = nullptr;
   for (auto& fuwuqi_ptr : all_fuwuqi_fenqu[i]) {
    if (fuwuqi_ptr->id_xuniji_dict.size() == 0) continue;
```

```
    if (fuwuqi_ptr->id_xuniji_dict.size() == 0) continue;
    if (v.find(fuwuqi_ptr) != v.end()) continue;
    if (v2.find(fuwuqi_ptr) != v2.end()) continue;
    double temp = (1.0 * ((double)fuwuqi_ptr->neiheUsedA + fuwuqi_ptr->neiheUsedB) / fuwuqi_ptr->neihe + 1.0 * (
(double)fuwuqi_ptr->neicunUsedA + fuwuqi_ptr->neicunUsedB) / fuwuqi_ptr->neicun);
    if (temp > 1.9) {
     v2.insert(fuwuqi_ptr);
     continue;
    }
    //if (max_cnt - cnt < fuwuqi_ptr->id_xuniji_dict.size()) continue;
    //int distance = fuwuqi_ptr->id_xuniji_dict.size();
    //double fuwuqi_core = fuwuqi_ptr->coreUsedA + fuwuqi_ptr->coreUsedB;
    //double distance = fuwuqi_core * memory_core_rate + fuwuqi_memory;
    if (temp < min_size) {
     min_size = temp;
     min_fuwuqi_ptr = fuwuqi_ptr;
    }
   }
   //double afterfindfuwuqi = clock();
   //findoptfuwuqi += (afterfindfuwuqi - beforefindfuwuqi) / 1000;
   if (min_fuwuqi_ptr == NULL) {
    break;
   }
   v.insert(min_fuwuqi_ptr);//不能迁移到自己身上
   bool end_flag = false;
   //将该机器上的虚拟机全部迁移出去
   vector<string> need_migra_xuniji;
   need_migra_xuniji.reserve(100);
   //double beforecharuxuniji = clock();
   for (auto& xuniji : min_fuwuqi_ptr->id_xuniji_dict) {
    if (fengqucount >= fenqu_max_count) {
     end_flag = true;
     break;
    }
    //先分配机器
    pair<MachineLibarary::AB, std::shared_ptr<FUWUQI>> old_p = MachineLibarary::get_instance()->id_fuwuqi_dict[x
uniji.first];
    pair<MachineLibarary::AB, std::shared_ptr<FUWUQI>> p = MachineLibarary::get_instance()->addXuNiJi(xuniji.fir
st, xuniji.second, v, all_fuwuqi_fenqu_set[i]);
    if (p.second == nullptr) {
     continue;
    }
    else {
     //分配成功
     qianyi_fenqu[i].push_back(make_pair(xuniji.first, p));
     fengqucount++;
     //移除原机器上的资源
     need_migra_xuniji.push_back(xuniji.first);
     if (old_p.first == MachineLibarary::AB::AB) {
      //双节点部署
      old_p.second->neiheUsedA -= xuniji.second.neihe / 2;
      old_p.second->neicunUsedA -= xuniji.second.neicun / 2;
      old_p.second->neiheUsedB -= xuniji.second.neihe / 2;
      old_p.second->neicunUsedB -= xuniji.second.neicun / 2;
     }
     else if (old_p.first == MachineLibarary::AB::A) {
      //在A上部署的
      old_p.second->neiheUsedA -= xuniji.second.neihe;
      old_p.second->neicunUsedA -= xuniji.second.neicun;
     }
```

```
    else if (old_p.first == MachineLibarary::AB::B) {
     //在B上部署的
     old_p.second->neiheUsedB -= xuniji.second.neihe;
     old_p.second->neicunUsedB -= xuniji.second.neicun;
    }
   }
  }
  /*double aftercharuxuniji = clock();
  charuxuniji += (aftercharuxuniji - beforecharuxuniji) / 1000;*/
  //移除ID
  for (int j = 0; j < need_migra_xuniji.size(); j++) {
   min_fuwuqi_ptr->id_xuniji_dict.erase(need_migra_xuniji[j]);
  }
  if (min_fuwuqi_ptr->id_xuniji_dict.size() != 0) {
   //确定最大时间
   min_fuwuqi_ptr->notUseDay = 0;
   for (auto& x : min_fuwuqi_ptr->id_xuniji_dict) {
    if (x.second.end_day > min_fuwuqi_ptr->notUseDay) {
     min_fuwuqi_ptr->notUseDay = x.second.end_day;
    }
   }
  }
  if (end_flag) {
   break;
  }
 }
}
//double after = clock();
//cout << (after - be) / 1000 << endl;
}


vector<pair<string, pair<MachineLibarary::AB, std::shared_ptr<FUWUQI>>>> MachineLibarary::migaration() {
 int max_count = GlobalMessage::Get().xuniji_count * 5 / 1000;
 vector<pair<string, pair<MachineLibarary::AB, std::shared_ptr<FUWUQI>>>> res;//迁移集合
 res.reserve(100);
 int FENQU = 0;
 int fuwuqicount = 0;
 fuwuqicount = fuwuqi_list.size();
 FENQU = fuwuqicount / 750 + 1 ;
 qianyi_fenqu.reserve(FENQU);
 qianyi_fenqu.resize(FENQU);
 for (int i = 0; i < qianyi_fenqu.size(); i++) {
  qianyi_fenqu[i].clear();
 }
 vector<shared_ptr<FUWUQI>> all_fuwuqi_vec;//所有服务器
 vector<vector<shared_ptr<FUWUQI>>> all_fuwuqi_fenqu(FENQU);//所有分区
 vector<unordered_set<shared_ptr<FUWUQI>>> all_fuwuqi_fenqu_set(FENQU);
 for (auto& fuwuqi_ptr : fuwuqi_list) {
  if (fuwuqi_ptr->id_xuniji_dict.size() == 0) {
   continue;
  }
  all_fuwuqi_vec.push_back(fuwuqi_ptr);
 }
 sort(all_fuwuqi_vec.begin(), all_fuwuqi_vec.end(), cmp);
 for (int i = 0; i < all_fuwuqi_vec.size(); i++) {
  all_fuwuqi_fenqu[i % FENQU].push_back(all_fuwuqi_vec[i]);
  all_fuwuqi_fenqu_set[i % FENQU].insert(all_fuwuqi_vec[i]);
 }
```

```
 vector<std::thread> threads;
 //区内迁移
 int fenqu_max_count = max_count / FENQU;
 //double be = clock();
 if (FENQU <= 4) {
  for (int i = 0; i < FENQU; i++) {
   //run_qianyi(i, i + 1, fenqu_max_count, all_fuwuqi_fenqu, all_fuwuqi_fenqu_set);
   threads.push_back(std::thread(std::bind(run_qianyi, i, i+1, fenqu_max_count, all_fuwuqi_fenqu, all_fuwuqi_fen
qu_set)));

  }
 }
 else {
  for (int i = 0; i < 4; i++) {
   //run_qianyi(FENQU * i / 4, FENQU * (i + 1) / 4, fenqu_max_count, all_fuwuqi_fenqu, all_fuwuqi_fenqu_set);
   threads.push_back(std::thread(std::bind(run_qianyi, FENQU*i/4, FENQU * (i+1) / 4, fenqu_max_count, all_fuwuqi
_fenqu, all_fuwuqi_fenqu_set)));
  }
 }

 for (auto& t : threads) {
  t.join();
 }
 /*double after = clock();
 cout << (after - be) / 1000 << endl;*/

 for (auto& p : qianyi_fenqu) {
  for (auto& q : p) {
   res.push_back(q);
  }
 }
 /*allcount += count;
 static double alltime = 0.0;
 cout << count<<"/"<< allcount << endl;*/
 /*cout << "找服务器" << findoptfuwuqi << endl;
 cout << "插虚拟机" << charuxuniji << endl;*/
 return res;
}
```

结尾

打这个比赛时，还是有很多骚想法因为时间问题没有实现，确定自己的算法能力还有待提升，前几名的大佬成本又低，时间还少，想破脑袋也想不出来，最后也陷入了没有任何思路的情况，总的来说还是自己太弱了，这次比赛更是刺激了我要好好提升自己的念头，每天必学两小时，争取缩小和大佬们之间的鸿沟
最后感觉这种比赛还是挺有意思的，既可以提升思维，又能结识大佬，以后可能还会参加！