云计算资源调度优化算法

——B15070204 刘美含

1. 云计算资源调度问题概述

云计算是在互联网技术、先进信息处理技术的基础上，对分布式处理、并行处理和网格计算的进一步延伸，可将其视为对以上科学概念的商业实现。区别于以上概念，云计算的主要特点体现在，通过使计算任务分布于地理位置分散的数据中心，让用户以支付即使用的透明方式进行资源租用，实现资源按需分配及大规模资源的全局共享。

资源调度是指在特定的应用场景中，根据制定的资源使用规则，在不同资源使用者之间进行资源分配与重调整的过程。资源使用者提交多样化的计算任务(例如虚拟解决方案、仿真实验、数据分析等)，数据中心接收到任务后通常有两种途径实现计算任务的资源调度:（1）调整分配给计算任务的资源使用量；（2）调整分配给计算任务的资源。一个好的资源调度方案需要综合考虑负载均衡、资源利用率、运行时间、所耗费用等多方面的因素。当前资源调度研究多集中于对单个目标的优化，考虑的约束难以满足云计算实际的运营环境，导致应用过程中存在很大的瓶颈。本文对云计算资源调度经典算法进行分析，比较各算法的优缺点及其所适用的资源类型、任务类型。基于各算法的不足提出相应的改进策略。

2. 数学模型

表 1 问题涉及到的表示符号

符号	定义
t	任务
p	资源
y_{tp}	每个 t 分配给 p
w_{tp}	每个 t 在每个 p 上的执行时间
T	总任务数
P	总资源数

问题的数学模型如下：

$$\sum_{p\in P} y_{tp} = 1 \qquad \forall t \in T \tag{1}$$

$$m \geq \sum_{t\in T} w_{tp} y_{tp} \qquad \forall p \in P \tag{2}$$

其中公式（1）表示每个任务只能被分配到一台机器上；公式（2）表示问题求解的目标是最小化 makespan（即最小化整个任务集的结束时间）。

3．经典算法介绍

以下介绍几种求解异构环境下资源调度的经典启发式算法。包括算法的基本思想、实现步骤。问题假设如下：

（1） 有 T 个待分配任务；

（2） P 台可用机器；

（3） 每个任务只能分配到一台机器上执行；

（4） 第 j 个任务在第 i 台机器上的执行时间为 Time（i，j）。

3.1 Min-Min

算法简述:Min-min 算法将任务 Task 分配到执行时间最短的资源上，同时保证总的执行时间最短。但是这样导致处理能力强的资源一直处于工作状态，而其他资源一直处于空闲状态，反而不能体现分布式处理的优势。而且这样也会倒是处理能力强的资源损耗较快。

时间复杂渐进度：$O（PT）$

3.2 Min-Max

算法简述：Max-Min 算法非常类似于 Min-Min 算法。同样要计算每一任务在任一可用机器上的最早完成时间，不同的是 Max-Min 算法首先调度大任务，任务到资源的映射是选择最早完成时间最大的任务映射到所对应的机器上。

时间复杂渐进度：$O（PT）$

3.3RelativeCost

算法简述：此算法通过计算 relativecost，将 relativecost 较小的任务先分配。

$$rs(t,p)=\frac{t在p上的预计执行时间}{t在所有p上执行时间的平均值}$$

$$rd(t,p)=\frac{t在p上的预计完成时间}{t在所有p上的预计完成时间的平均}$$

$$\text{relativecost}=\text{rd}^{\partial}\times rs \qquad \partial=0.5$$

此算法改善了负载不平衡的问题，但是相对复杂也更耗时。

时间复杂渐进度：$O(PT^2)$

3.4Sufferage

算法简述：此算法将最优资源和次优资源相差最大的任务先分配给最优资源。

时间复杂渐进度：$O(PT^2)$

3.5PenaltyBased

算法简述：此算法先分配所有任务给其最优资源，但是容易造成负载不均衡，所以随后判断是否可以将最大负载的资源中的任务调度到最小负载资源再进行调整。可以调整的条件是任务在最小负载上的预计完成时间小于最大负载的资源预计完成时间与最小负载上的预计完成时间的平均时间，按照

$$\frac{\text{t在最大负载资源预计完成时间-t在最小负小负载预计完成}}{\text{t在最大负大负载资源预计时间}}$$

来决定哪个任务最先调度。

时间复杂渐进度：$O(TP+T^2)$

3.6ListSufferage

算法简述：此算法基于 3.4Sufferage 算法，但是算法所耗时间更少，关键是计算 priority 然后将此值由小到大进行排序，根据排序后的顺序分配任务。

priority 计算方法如下：

如果 p 是 t 的最优资源：

$$piority=\frac{次优资源预计执行时间}{最优资源预计执行时间}$$

否则：

$$piority = \frac{\text{最优资源预计执行时间}}{t\text{在}p\text{上的预计执行时间}}$$

时间复杂渐进度：$O(PT\log T)$

3.7 TPB

算法简述：此算法基于 3.5PenaltyBased 算法，同样是解决负载不均衡的问题，先将任务分配给执行最快的资源。先将任务分配给最优资源，然后根据任务在其他各个资源预计完成时间升序排序，若预计完成时间小于执行最快的资源预计完成时间，则将任务分配给负载最小资源。

时间复杂渐进度：$O(PT^2)$

4.算法实现

根据各个算法的时间复杂度可以看出，Min-Min 算法和 Min-Max 算法耗时较小。本文对 Min-Min 算法进行具体分析并编码实现。

4.1 算法流程

*Step1:*判断任务集合 T 是否为空，不为空，执行（2）；否则跳到步骤（7）。

*Step2:*对于任务集中的所有任务，求出它们映射到所有可用机器上的最早完成时间。

*Step3:*根据（2）的结果，找出最早完成时间最小的那个任务 *t* 和所对应的机器 *p*。

*Step4:*将任务 *t* 映射到机器 *p* 上；并将该任务从任务集合中删除。

*Step5:*更新机器 p 的期望就绪时间。

*Step6:*更新其它任务在机器 p 上的最早完成时间；回到（1）。

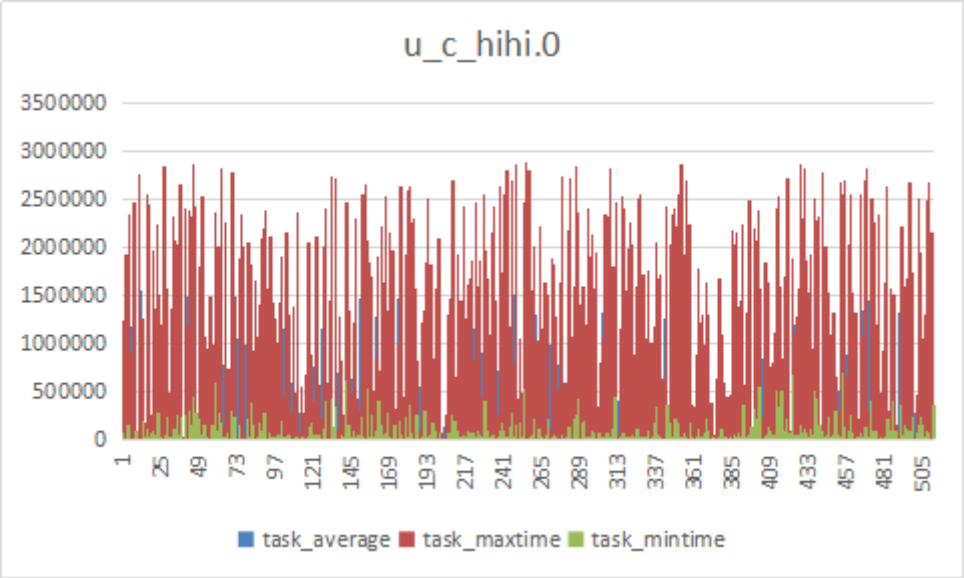
*Step7:*此次映射事件结束，退出程序。

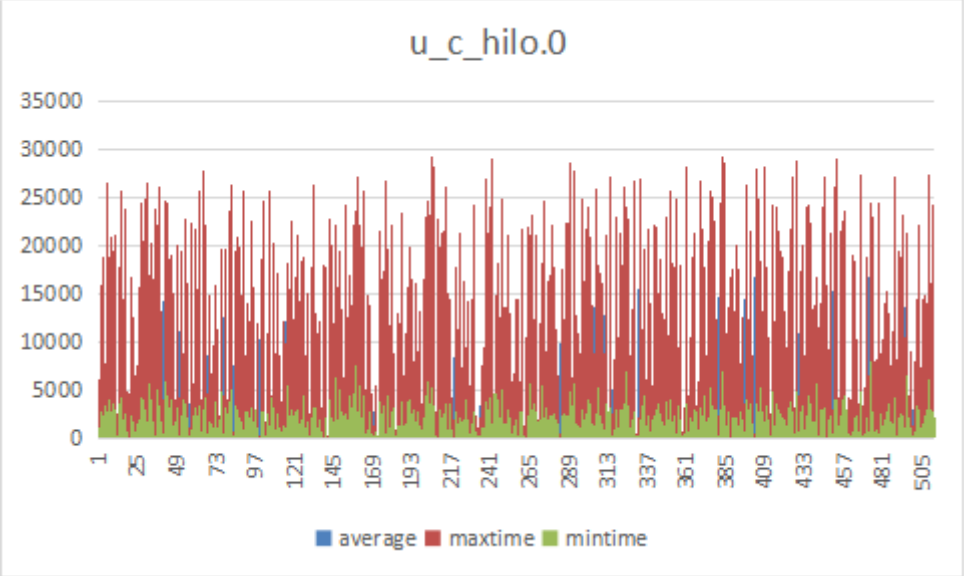
4.2 代码实现

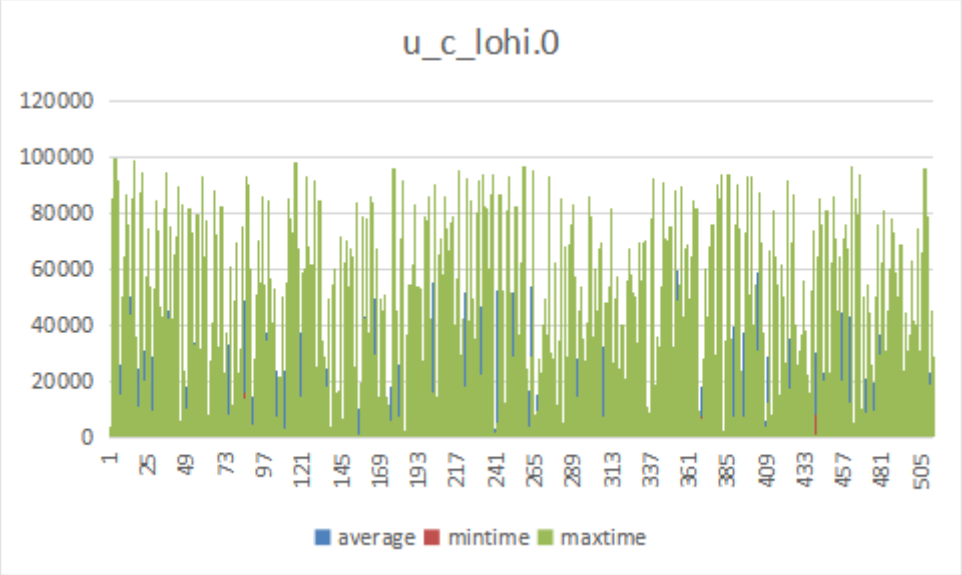
```
#include<stdio.h>
#include<stdlib.h>

void delete(double task[512][16],int min_min_task)//将任务Task映射到 P上后删除
{
    int i,j;
    for(i=min_min_task;i>=0;i--){
        for(j=0;j<16;j++){
            task[min_min_task][j]=task[min_min_task-1][j];
        }
    }
    for(j=0;j<16;j++){
        task[0][0]=0;
    }
}

int main()
{
    FILE *fp1;
    int i,j,t;
    double task[512][16]={0},p[16]={0},pro[16];
    double makespan;
    double min_time,min_min_time;
    int min_min_task,min_min_p,min_task,min_p;
    fp1=fopen("D:\\Almh\\study\\project\\cloudeclc\\data\\12\\u_s_lolo.0","r");
```

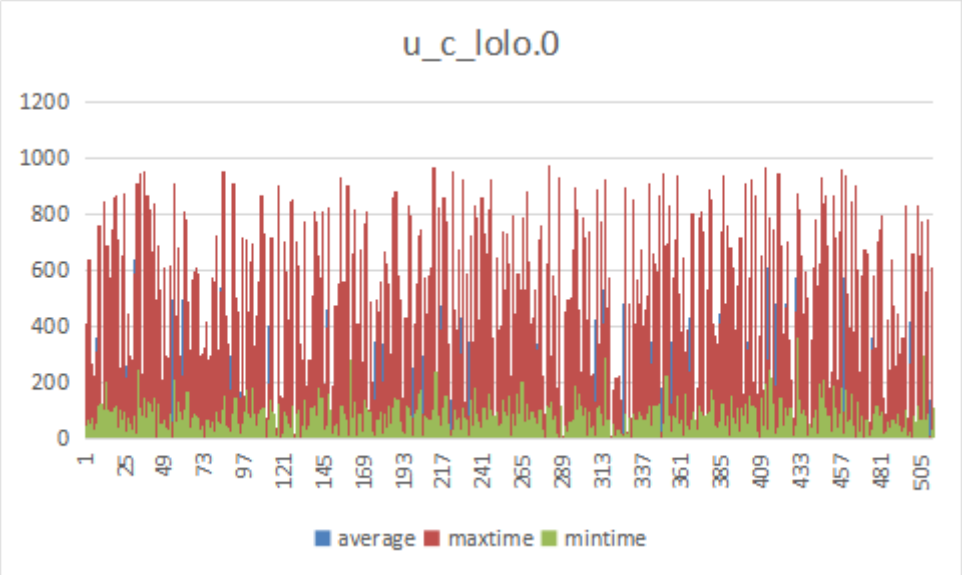
```
        if(fp1==0)
        {
            printf("error\n");
            exit(1);
        }
        for(i=0;i<512;i++)//将数据存入数组
        {
            j=0;
            for(;j<16;j++)
            {
                fscanf(fp1,"%lf",&task[i][j]);
            }
        }
        for(t=0;t<512;t++){
            min_min_time=p[0]+=task[t][0];
            min_min_task=t;
            min_min_p=0;
            for(i=t;i<512;i++){
                min_time=p[0]+=task[i][0];
                min_p=0;
                for(j=1;j<16;j++){
                    p[j]+=task[i][j];
                    if(min_time>p[j]){//找到每个任务最小完成时间
                        min_time=p[j];
                        min_p=j;
                    }
                }

            }
            pro[min_min_p]+=task[min_min_task][min_min_p];//部署任务
            for(j=0;j<16;j++){
                p[j]=pro[j];
            }
            delete(task,min_min_task);//删除任务
        }
        makespan=pro[0];
        for(i=1;i<16;j++){
            if(pro[i]<makespan){
                pro[i]=makespan;
            }
        printf("%lf",makespan);
        fclose(fp1);
        return 0;
        }

    }

            }
            pro[min_min_p]+=task[min_min_task][min_min_p];//部署任务
            for(j=0;j<16;j++){
                p[j]=pro[j];
            }
            delete(task,min_min_task);//删除任务
        }
        makespan=pro[0];
        for(i=1;i<16;j++){
            if(pro[i]<makespan){
                pro[i]=makespan;
            }
        printf("%lf",makespan);
        fclose(fp1);
        return 0;
        }

}
```

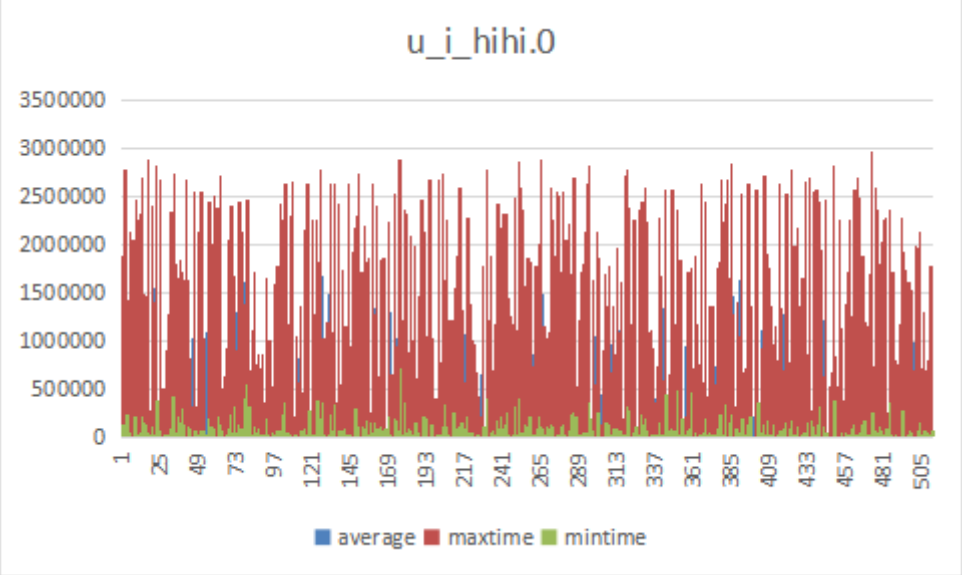
4.实验结果分析

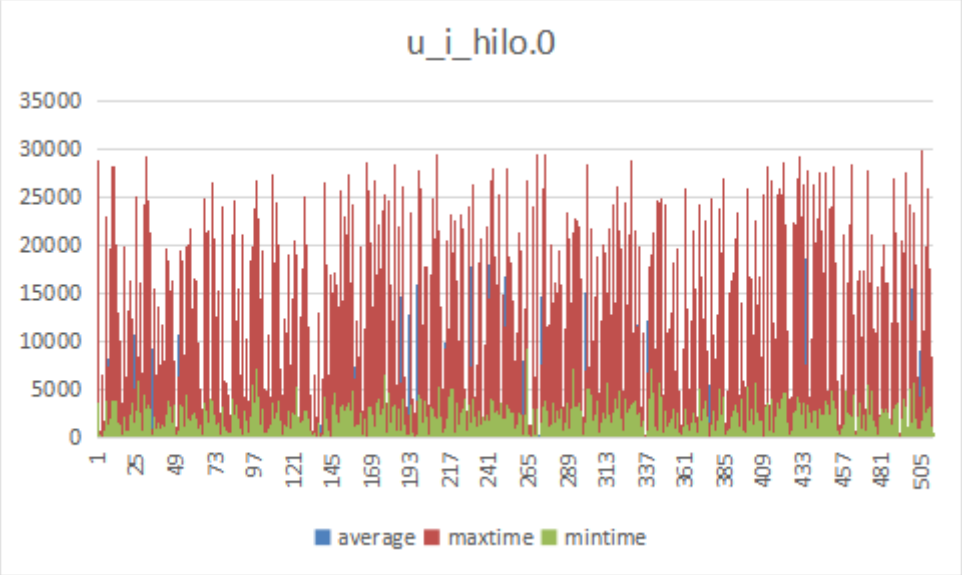


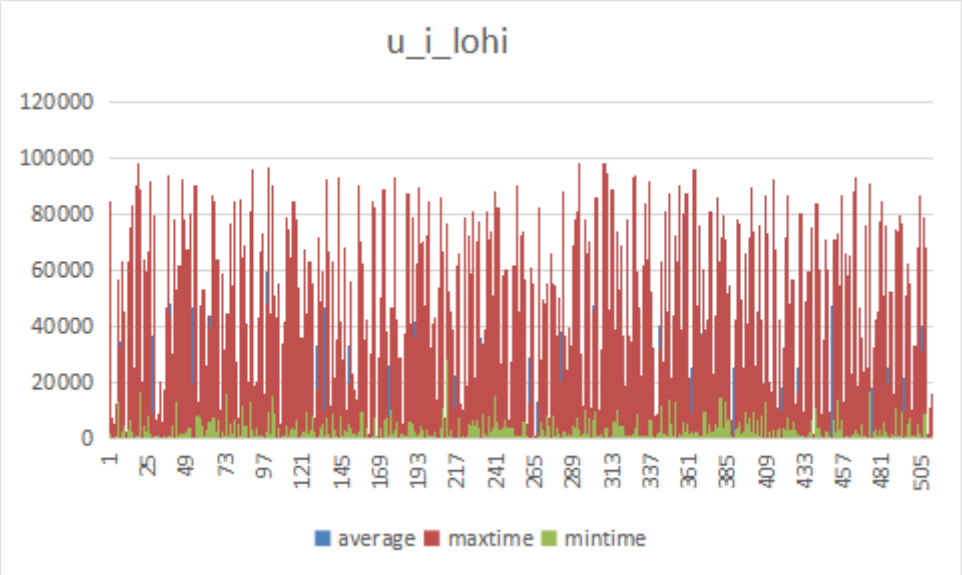


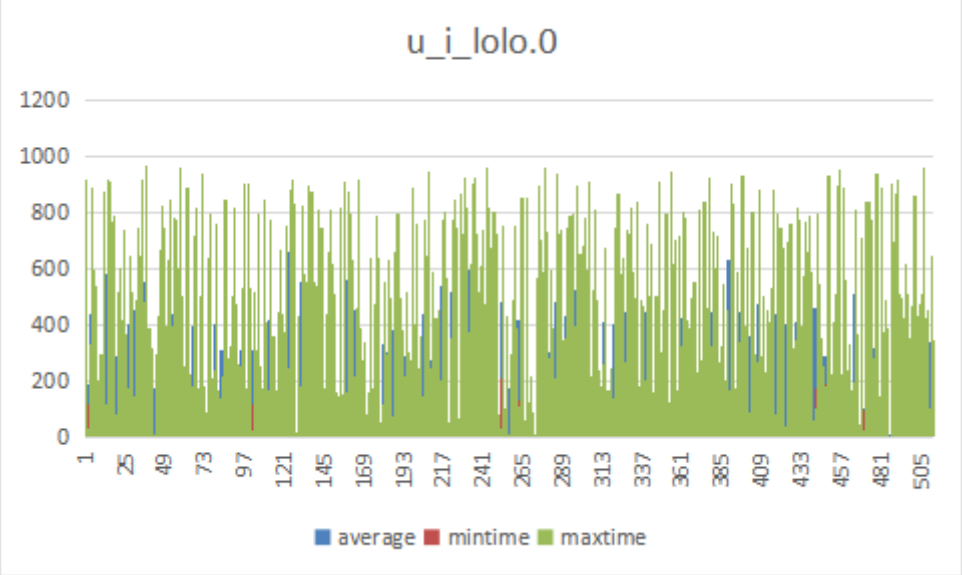


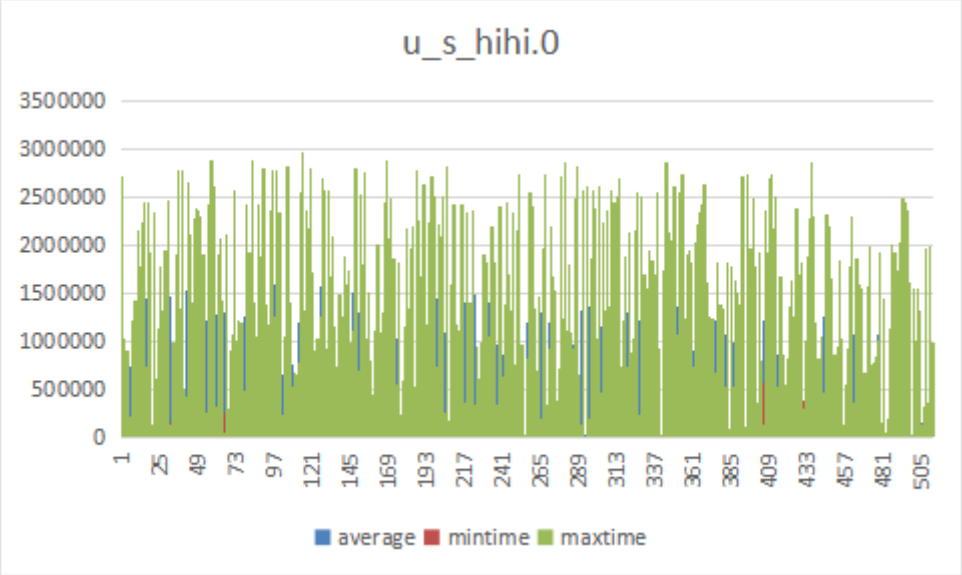


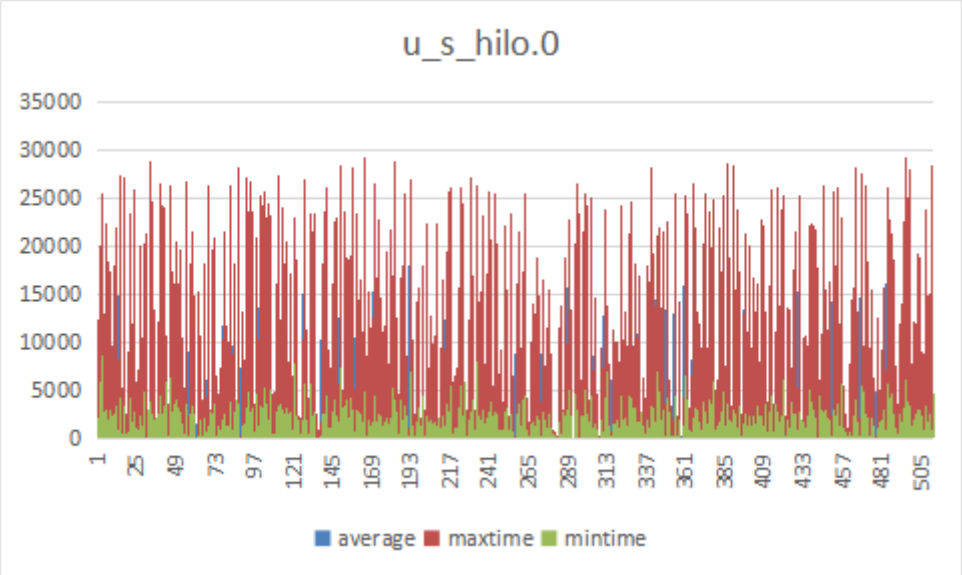


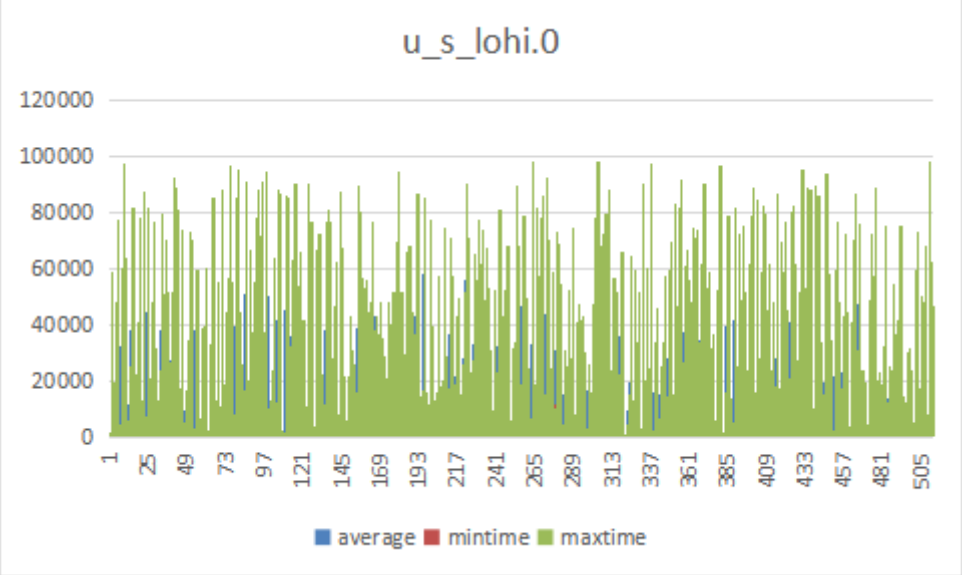


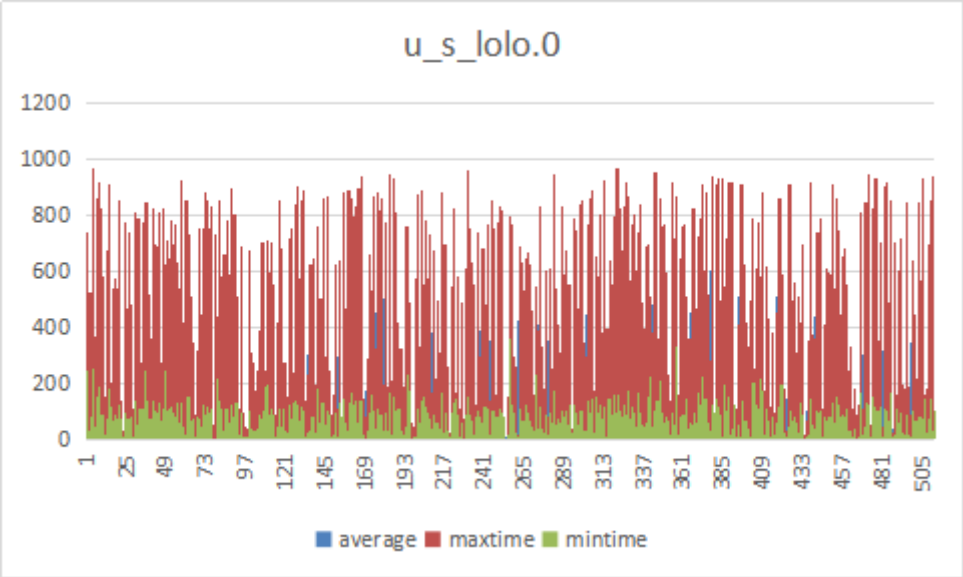


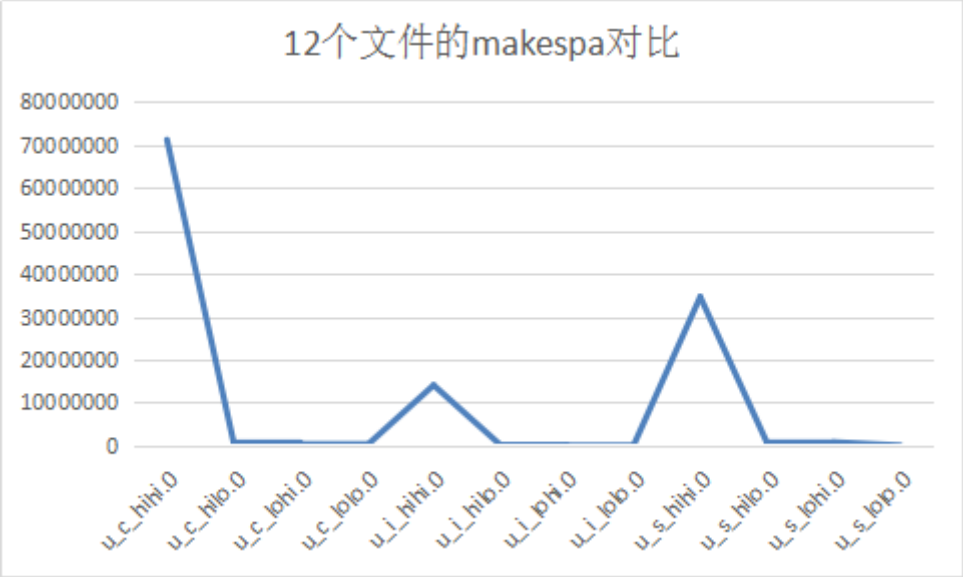












由折线图可以清楚的看出，在一定范围内的数据执行时间差异不大，但是当数据量到达某个极限时，时间激增。

5.课程总结

七个算法各有优劣，可以大致分为三类：第一类，是算出数值再比较处理（如：*RelativeCost\ListSufferage* 算法）；第二类，是 *Min-Min* 和 *Min-Max* 算法；第三类，是先分配给最优资源造成负载不平衡后再做调整（如：*Sufferage\PenaltyBased\TPB* 算法）。但是都存在几个明显的问题

从编码上来看：（1）当数据十分大的时候，算法无法解决空间使用过大的问题。用内存数组处理不是很妥当，虽然可以将大数据分组处理，但是占用了太多的内存。同时占用空间太大，处理掉的数据的空间无法清除。（2）现实环境总是各异的，具体问题具体分析，所有算法都是理想化的，只是现实问题的一个精华，但是并不能解决错综复杂的现实问题。关于这个问题，我也试图解决。通过全局变量和文件读写的方式，每次只做一个任务，任务完成后即删除，可以大大节省空间，但是之后发现，文件的多次打开关闭更加浪费时间，具体解

决方法也没有找到，这个方法出发点是好的，然而并不是很成功。
但是此思想也许可以为进一步研究所借鉴。

```
#include<stdio.h>
#include<stdlib.h>
static double maiche[4]={0};
static double Max=0;
static int Key=0;
static int Judge=0;
static int Count=0;
void findmin(double *array,int a,int count)
{
    double min;
    int i=1,key=0;
    min=array[0]+maiche[0];
    while(i<4)//找到某个任务在四个机器运行后的最短时间
    {
        if(array[i]+maiche[i]<min)
        {
            min=array[i]+maiche[i];
            key=i;
        }
        i++;
    }
    if(a==0)//初次运行赋值
    {
        Max=min;
        Key=key;
        Judge=1;
        Count=count;
    }
    else
    {
        if(min>=Max)
        {
            Max=min;
            Key=key;
            Count=count;
        }
    }
```

```
void minmax(int Key,double Min)
{
    switch(Key)
    {
        case 0:maiche[0]=Max;break;
        case 1:maiche[1]=Max;break;
        case 2:maiche[2]=Max;break;
        case 3:maiche[3]=Max;break;
    }
}


int main()
{
    FILE *fp1,*fp2,*fp3;
    double data[4];//一个数据在4个机器上的运行时间
    int count=1,j=6;//j为多少个数据，count计算行数
    fp1=fopen("D:\\Almh\\study\\project\\cloudeclc\\data\\512_16\\mytest.txt","r");
    fp2=fopen("D:\\Almh\\study\\project\\cloudeclc\\data\\512_16\\mytest0.txt","w+");
    fp3=fopen("D:\\Almh\\study\\project\\cloudeclc\\data\\512_16\\mytest1.txt","w+");
    if(fp1==0||fp2==0)
    {
        printf("error\n");
        exit(1);
    }
    while(fscanf(fp1,"%lf%lf%lf%lf",&data[0],&data[1],&data[2],&data[3])!=-1)//将文件一复制到文件二
    {
        printf("%lf\t%lf\t%lf\t%lf\n",data[0],data[1],data[2],data[3]);
        fprintf(fp2,"%lf %lf %lf %lf\n",data[0],data[1],data[2],data[3]);
    }
    printf("\n");
    /*
    rewind(fp2);
    while((fscanf(fp2,"%lf %lf %lf %lf",&data[0],&data[1],&data[2],&data[3]))!=-1)
    {
        printf("%lf\t%lf\t%lf\t%lf\n",data[0],data[1],data[2],data[3]);
    }*/
```

```
    fclose(fp2);
    while(j>0)
    {
        count=1;
        Judge=0;
        fp2=fopen("D:\\Almh\\study\\project\\cloudeclc\\data\\512_16\\mytest0.txt","r+");

        //printf("dayin2 ");
        while((fscanf(fp2,"%lf %lf %lf %lf",&data[0],&data[1],&data[2],&data[3]))!=-1)//打印2
        {
            //printf("%lf\t%lf\t%lf\t%lf\n",data[0],data[1],data[2],data[3]);
            findmin(data,Judge,count);
            count++;
        }
        //printf("\n");
        minmax(Key,Max);
        count=1;
        rewind(fp2);
        fp3=fopen("D:\\Almh\\study\\project\\cloudeclc\\data\\512_16\\mytest1.txt","w+");
        while((fscanf(fp2,"%lf %lf %lf %lf",&data[0],&data[1],&data[2],&data[3]))!=-1)//删除已完成任务 ，重写文件
        {
            if(count!=Count)
            {
                fprintf(fp3,"%lf %lf %lf %lf\n",data[0],data[1],data[2],data[3]);
            }
            count++;
        }
        fclose(fp2);
        rewind(fp3);
        /*
        printf("dayin3 ");
        while((fscanf(fp3,"%lf %lf %lf %lf",&data[0],&data[1],&data[2],&data[3]))!=-1)//打印3
        {
            printf("%lf\t%lf\t%lf\t%lf\n",data[0],data[1],data[2],data[3]);
        }
        printf("\n");*/
        rewind(fp3);
```

```
fclose(fp2);
while(j>0)
{
    count=1;
    Judge=0;
    fp2=fopen("D:\\Almh\\study\\project\\cloudeclc\\data\\512_16\\mytest0.txt","r+");

    //printf("dayin2 ");
    while((fscanf(fp2,"%lf %lf %lf %lf",&data[0],&data[1],&data[2],&data[3]))!=-1)//打印2
    {
        //printf("%lf\t%lf\t%lf\t%lf\n",data[0],data[1],data[2],data[3]);
        findmin(data,Judge,count);
        count++;
    }
    //printf("\n");
    minmax(Key,Max);
    count=1;
    rewind(fp2);
    fp3=fopen("D:\\Almh\\study\\project\\cloudeclc\\data\\512_16\\mytest1.txt","w+");
    while((fscanf(fp2,"%lf %lf %lf %lf",&data[0],&data[1],&data[2],&data[3]))!=-1)//删除已完成任务，重写文件
    {
        if(count!=Count)
        {
            fprintf(fp3,"%lf %lf %lf %lf\n",data[0],data[1],data[2],data[3]);
        }
        count++;
    }
    fclose(fp2);
    rewind(fp3);
    /*
    printf("dayin3 ");
    while((fscanf(fp3,"%lf %lf %lf %lf",&data[0],&data[1],&data[2],&data[3]))!=-1)//打印3
    {
        printf("%lf\t%lf\t%lf\t%lf\n",data[0],data[1],data[2],data[3]);
    }
    printf("\n");*/
    rewind(fp3);
```

```
    //printf("\n");
    fp2=fopen("D:\\Almh\\study\\project\\cloudeclc\\data\\512_16\\mytest0.txt","w+");
    while((fscanf(fp3,"%lf %lf %lf %lf",&data[0],&data[1],&data[2],&data[3]))!=-1)//将3复制到2
    {
        fprintf(fp2,"%lf %lf %lf %lf\n",data[0],data[1],data[2],data[3]);
    }
    fclose(fp2);
    fclose(fp3);
    j--;
    //printf("%d jieshu1lun ",j);
    //printf("\n");
}
printf("\na\t\tb\t\tc\t\td\n");
printf("%lf\t%lf\t%lf\t%lf\n",maiche[0],maiche[1],maiche[2],maiche[3]);
fclose(fp1);
fclose(fp2);
return 0;
```

从算法上来看：（1）系统角度，公平性:每个资源(不论优先级)都有机会被运行;较大的吞吐量。而算法一定程度上导致负载不均衡。用户角度，及时性:响应速度要快;较短的周转时间:不应当让用户等待时间过长。而算法只考虑到了最终时间的长度。
所以，云计算资源调度还有待进一步研究。