# Resources Allocation

## Goal

The main goal is to propose a scheme to allocate resources on the sever automatically when a new task allocation request comes,and decide when to rent a new sever from the cloud service provider. Much prior work has been down by tutor Wang Hu. So I mainly add some functions in *bidschedular.py*,and the UI of this file is to receive job_allocation_request from *vclustermanager.py* and return job_allocation_response,which indicate which machine to place the tasks in the job and what kind of resources they will get.

## scheme

### 1. resources

On the machines we manage the resources like CPU and memory with *cgroup*, and using different functions in this tool we can achieve two kinds of restrictions of resources, one kind, which we will call reliable resources, means that if the restriction of the task is A, then this task will always get enough resources whenever it needs in condition that the resources it possesses is no more than A. The other kind is called restricted resources, which means is a task has be allocated this kind of resources with the amount of B,then it will possesses no more than B resouces,but when there is a resources shortage on the machine, it will get less resources. We will give the tasks with high priority reliable resources, when there is not enough reliable resources, we will allocate

the restricted resources to the tasks with low priority
It should be noted that the reliable resources allocated on a machine should be no more than the summary of the resources,or there will be conflict. But the summary oif restricted resources that can be allocated has no limit. As it only means the maximum resources a task can possess, if there are not enough resources it wll just share whatever is left with other tasks. And when all of the resources on a machine has already been allocated as reliable resources,several tasks with restricted resources an still be placed on this machine, as the tasks can barely use up the reliable resources it possess. If a task has got reliable resources with 512Mb memory, it will only use no more than 400M most of the time ,the ramaing resources can be shared by the tasks with restricted resources.And when the resources is indeed used up. the reliable tasks shall get their share, and the restricted ones will just wait. To be summary the raliable tasks are like lords carving the land of the country up,and the restricted tasks are rangers make a survival among the kindoms.

## 2.price and tasks

Our scheme is mainly based on a market competation system.The resources is allocated with the unit of slot, which is made up of certain amount of CPU and memory(the accurate size is not decided yet). And users request for resources with the form of jobs,which consist of several tasks, and each task will be conducted in a lxc, and different tasks in a job can be placed on different machines and get different kind of resources. When proposing a job request, the user must make a statement of how many slots each task needs and how much he will pay for per slot per time unit(hour,for example).

we will devide the price into several levels,the lowest level is the price for restricted resources,which is a fixed value. And the second level is the price for reliable resources.This level is where the competition happens. It has an initiate price,and above this initiate price users can give the bid of their own. It’s OK if there are enough resources, but if there are not,the tasks with higher bid will get the reliable resources and the loser will only get restricted resources.And the last level is the price to expand our sever system, in other words the price of per slot to rent servers from cloud service provider. As an example,digitalocean, a cloud service provider,provide sever charged by hour,and the price of different server is liner according to resources, the sever with one CPU,1G memory is $0.015 per hour and the sever with 2 core ,2G momory is $0.03per hour,so we can cauclute a certain cost of expand per slot according to the price of cloud service provider.When users offer a price of this cost,their tasks will always get enough reliable resources

## 3. bid scheme

when propose a job request,the user can choose from three strategies: first, if he just need the tasks to be done, he can choose the lowest level of price, it will restrict the resources he can get, but the works will be completed sooner or later. Second, if he want the tasks to be finished more quickly, he can choose the second level, and offer a bid according to his will. Based on his bid, some, may be all of his works,will get reliable resources and get a guarantee of speed. But notice that if there is a new request with higher bid, his tasks may be preempted and use restricted resources instead. At last, if he is eagerly to finish the tasks or he is carrying out an expriment which need to ensure the resources to guarantee the performence, he can choose to off a price up to the cost to expand our system, in this condition his tasks will always get enough reliable resources.

Using this scheme,the users will have multiple choices to meet their need. And with the competition system in level 2, our profit will be guaranteed.

But there may be some problems with renting servers. As discussed above, the servers we rented are charged by hour, we can choose when to release it. But until now we has not considered transfering tasks between machines in Docklet. And it’s the users who decide when a release a lxc, which means if we use the machines we rented the same as our machines,it may end up in the situation that several tasks with low bid, even just the price of restricted resources,running on the machine, keeping us away from releasing it,and the money we charged on these tasks is much less than that we paied for renting the machine. And whats worse, if the owners of these tasks keep them running for a long time, the loss will be heavy. So we will only run the tasks with the third level of price on the rented machine, and no restriction resources will be allocated. Once there is on task on the machine, we will release it immediately. But this will lead to relatively low utilization ratio of resources on the rentd machine. A better solution is expected.

## Implementation

The brief implementation of the scheme and several clips of code will be shown in this section. The main data structure used is as following:

```
class AllocationOfTask(object):
    __slots__ = 'id','userid','jobid','taskid','resources',
'bidprice','type',               'machineid','lxc_name'
```

```
class AllocationOfMachine(object):
 __slots__=['machineid',"resources","reliable_resources_all
ocation_summary",           'reliable_allocations', 'restri
cted_allocations','rented']
```

and the main funcion used is *allocate(job_allocation_request):*,it uses several functions,which are as following:

First,the function to examine if there is enough reliable resources on a machine and the function to allocate reliable resources:

```
def has_reliable_resources(allocation_of_machine,task_alloc
ation_request):
    if(task_allocation_request['resources']
       +allocation_of_machine.reliable_resources_allocation
_summary
       <= allocation_of_machine.resources):
        return True
    else:
        return False
```

```
def allocate_reliable_task(allocation_of_machine,task_allocation_request):
    global lxcname_allocation_dict
    if(has_reliable_resources(allocation_of_machine,task_allocation_request)):
        allocation_of_task = AllocationOfTask()
        allocation_of_task.id = uuid.uuid4()
        allocation_of_task.userid = task_allocation_request['userid']
        allocation_of_task.jobid = task_allocation_request['jobid']
        allocation_of_task.taskid = task_allocation_request['taskid']
        allocation_of_task.resources = task_allocation_request['resources']
        allocation_of_task.bidprice = task_allocation_request['bidprice']
        allocation_of_task.machineid = allocation_of_machine.machineid
        allocation_of_task.lxc_name = (allocation_of_task.userid
                                       + "-"
                                       + str(allocation_of_task.jobid)
                                       + "-"
                                       + str(allocation_of_task.taskid))
        allocation_of_task.type = 'reliable'
        bisect.insort(allocation_of_machine.reliable_alloca
tions, allocation_of_task)
        lxcname_allocation_dict[allocation_of_task.lxc_name
]=allocation_of_task

        # update allocation_summary
        allocation_of_machine.reliable_resources_allocation
_summary += task_allocation_request['resources']
        return {'status':'success', 'allocation':allocation
_of_task}
```

Then the function to check if a task can preempt enough reliable resources on a machine and the function to implement the preemption:

```
def can_preempt_reliable_resources(allocation_of_machine, t
ask_allocation_request):
#allocation_of_mation的reliable_allocations是按照bidprice从小
到大  序排列的
    to_be_preempted=0
    preempt_cost=0
    for a in allocation_of_machine.reliable_allocations:
        if (a.bidprice < task_allocation_request['bidprice'
]):
            to_be_preempted += a.resources
            preempt_cost +=a.bidprice
            if to_be_preempted >= task_allocation_request['
resources']:
                return preempt_cost
        else:
             break
    return -1
```

And the return value of this function is the summary of bidprice of the tasks will be preempted, after the check we will choose a machine where the summary is minimum to implement the preemption.

```
def try_preempt_resources(allocation_of_machine,task_alloca
tion_request):
        can_preempt = 0
        can_preempt_count = 0
        # 把被  占的可靠  源  成受限制  源
        for a in allocation_of_machine.reliable_allocations
:
            can_preempt+=a.resources
            can_preempt_count+=1
            #  成受限
            a.type = 'restricted'
            a.bidprice = restricted_price
            bisect.insort(allocation_of_machine.restricted_
allocations,a)
            # 更新allocation_machine的reliable_resources_all
ocation_summary
            allocation_of_machine.reliable_resources_alloca
```

```
tion_summary -= a.resources
            if can_preempt>=task_allocation_request['resour
ces']:
                break
        for i in range(0,can_preempt_count):
            change_cgroup_settings(allocation_of_machine.re
liable_allocations[i], 'restricted')
#此   想将被  占任  的价格改   一的restricte resources 的价格，那
#价格也需要修改，但是不确定   中是是按整个job   是  个task  ，待定

        # 把被  占的可靠  源从reliable_allocations中  除
        del allocation_of_machine.reliable_allocations[0:ca
n_preempt_count]

        allocation_of_task = AllocationOfTask()
        allocation_of_task.id = uuid.uuid4()
        allocation_of_task.userid = task_allocation_request
['userid']
        allocation_of_task.jobid = task_allocation_request[
'jobid']
        allocation_of_task.taskid = task_allocation_request
['taskid']
        allocation_of_task.resources = task_allocation_requ
est['resources']
        allocation_of_task.bidprice = task_allocation_reque
st['bidprice']
        allocation_of_task.resources = task_allocation_requ
est['resources']
```

```
        allocation_of_task.machineid = allocation_of_machin
e.machineid
        allocation_of_task.lxc_name = (allocation_of_task.u
serid
                                       + "-"
                                       + str(allocation_of_
task.jobid)
                                       + "-"
                                       + str(allocation_of_
task.taskid))
        allocation_of_task.type = 'reliable'
        bisect.insort(allocation_of_machine.reliable_alloca
tions, allocation_of_task)
        lxcname_allocation_dict[allocation_of_task.lxc_name
]=allocation_of_task

        # update allocation_summary
        allocation_of_machine.reliable_resources_allocation
_summary += task_allocation_request['resources']
        return {'status':'success', 'allocation':allocation
_of_task}
```

At last the function to allocate restricted resources. It’s similar to the functions above, so we won’t bother to present it here.

When receive a job request,first we check if it is not with the first level of price:

```
 if job_allocation_request['bidprice']>restricted_price
```

If so, we try to allocate reliable resources for it:

```
# 先从可靠　源最多的机器分配　源
 for i in range(int(job_allocation_request['tasks_count']))
:
            task_allocation_request = {
                'userid': job_allocation_request['userid'],
                'jobid': job_allocation_request['jobid'],
                'taskid': i,
                'bidprice': job_allocation_request['bidpric
e'],
                'resources': int(job_allocation_request['re
sources']),
            }
            logger.debug("a2")
            min=0xFFFF
            allocation_ID=0
            for j in range(0,len(allocations_list))
                if(allocations_list[j].reliable_resources_a
llocation_summary<min)
                    if allocations_list[j].rented==False or
 task_allocation_request[i].bidprice>=price_to_rent
                    min=allocations_list[j].reliable_resour
```

```
ces_allocation_summary
                    allocation_ID=j
            if(has_reliable_resources(allocations_list[allo
cation_ID],task_allocation_request)
                task_allocation_response = allocate_reliabl
e_task(allocations_list[allocation_ID],task_allocation_requ
est)
                job_allocation_response.append(task_allocat
ion_response)
```

Notice that when the machine is rented, only if the task has a third level price, it will be placed on the machine. If there are not enough reliable resources, we check if its price is up to the cost to expand,and if the answer is yes, rent a new server.

```
if(task_allocation_request.bidprice>=price_to_rent)
    for(k in machine_to_rent_list)
        if(k.resources>task_allocation_request.resources)
           rent(i.id)
           addNode(machine_num,i.resources,True)
           task_allocation_response =allocate_reliable_task
(allocations_list[machine_num],task_allocation_request)
           job_allocation_response.append(task_allocation_r
esponse)
           machine_num+=1
           times_of_renting+=1
           break
```

The machine_num is the current number of machines we have, the times_of_renting is the times that the system rent servers automatically.It is used to aid the manager to decide whether it’s time to buy a new server or to rent a server by month or year.And machine_to_rent_list is the list of all kinds of machines cloud service provider can provide, sorted by the amount of resources.
And if the price is not enough to rent a new server,we try to preempt other tasks:

```
    min=0xFFFF
    allocations_ID=0
    for j in range(0,len(allocations_list))
        cost=can_preempt_reliable_resources(allocations_list[j],task_allocation_request)
        if cost>0 and cost<min
            min=cost
            allocations_ID=j
    if(min<0xFFFF)
        task_allocation_response = try_preempt_resources(allocations_list[allocations_ID],task_allocation_request)
        job_allocation_response.append(task_allocation_response)
```

And if can’t preempt either,the other tasks will be allocated with restricted resources (the tasks have the same size and the same bidprice,if one can not get reliable resources, the remains can not either) And if the job has a fisrt level price, it will go directly to this stage :

```
global usages_list
sorted(usages_list, key=lambda x: x['cpu_utilization'], rev
erse=True)
j = 0
for i in range(len(job_allocation_response),int(job_allocat
ion_request['tasks_count'])):
    machineid = usages_list[j]['machineid']
    while machine_allocation_dict[machineid].rented==True
        j += 1
        if j>=machine_num
        j=0
        machineid = usages_list[j]['machineid']
    j+=1
    if j>=machine_num
    j=0
    allocation_of_machine = machine_allocation_dict[machine
id]
    task_allocation_request = {
         'userid': job_allocation_request['userid'],
         'jobid': job_allocation_request['jobid'],
         'taskid': i,
         'bidprice': job_allocation_request['bidprice'],
         'resources': int(job_allocation_request['resources
'])
            }
    task_allocation_response = allocate_task_restricted(all
ocation_of_machine,task_allocation_request)
    job_allocation_response.append(task_allocation_response
)
```

And when release resources,we should check if a rented machine has no tasks running,and then release it:

```
def release_allocation(lxc_name):
    allocation_of_task = lxcname_allocation_dict[lxc_name]
    allocation_of_machine = machine_allocation_dict[allocat
ion_of_task.machineid]
    if allocation_of_task.type == "reliable":
        i = bisect.bisect_left(allocation_of_machine.reliab
le_allocations,allocation_of_task)
        del allocation_of_machine.reliable_allocations[i]
        if len(allocation_of_machine.reliable_allocations)=
=0 and allocation_of_machine.rented==True
            releasemachine(allocation_of_machine.machineid)

    else:
        i = bisect.bisect_left(allocation_of_machine.restri
cted_allocations,allocation_of_task)
        del allocation_of_machine.restricted_allocations[i]
    return
```

## Experiment

The functions has been examined but as many UI is not completed yet,for exmaple:the HTML which present users with choices and collect informations, and the functions to rent a sever or release it, which is dealed with by another student as far as I know, so the code can not actually fit in the system. It’s only a scheme of the allocation strategy. And I will submit the code and document of the strategy to Wang Hu as a proposal.