调度大赛报告

- 阿里巴巴系统软件

Speaker
地球漫步

Alibaba
System
Software

团队成员介绍

- 团队：地球漫步
- 昵称：上帝本人
- 姓名：乔德平
- 爱好：到处溜达

MacBook调度记录-时间点统计（提交评审版）

格式为：得分－达到该得分的时间

因为算法的特点，不好改为按照时间点输出，而是按照实例迭代步数输出，结果看起来分数下降的不够平滑

该数据为手工指定机器数量计算得到的

	10分钟内	20分钟内	30分钟内	40分钟内	50分钟内	60分钟内	60分钟外
A数据	无	4982.96416-16.85	4888.079375-22.25	4857.038284-31.95	4838.205156-47.24	4838.205156-47.24	4838.205156-47.24
B数据	7478.989039-9.53	4932.909585-19.60	4932.909585-19.60	4907.384713-30.65	4907.384713-30.65	4906.181764-50.04	4906.181764-50.04
C数据	10427.640961-9.81	7255.088509-16.08	7003.468028-29.57	7003.468028-29.57	6887.808126-48.64	6887.808126-48.64	6803.484118-84.92
D数据	7357.884997-9.70	7042.62325-16.40	6891.187526-25.81	6891.187526-25.81	6750.059954-42.46	6750.059954-42.46	6681.770625-78.25
E数据	8628.884347-9.04	8556.978685-16.56	8499.219433-29.52	8499.219433-29.52	8499.219433-29.52	8499.219433-29.52	8499.219433-29.52
总分	无	6554.1128378	6442.9727894	6431.6595968	6376.5354764	6376.2948866	6345.7722192

MacBook调度记录-时间点统计（当前版本）

格式为：得分－达到该得分的时间

因为算法的特点，不好改为按照时间点输出，而是按照实例迭代步数输出，结果看起来分数下降的不够平滑

该数据为手工指定机器数量计算得到的

	10分钟内	20分钟内	30分钟内	40分钟内	50分钟内	60分钟内	60分钟外
A数据	5515.185331-8.91	4767.930950-15.54	4744.896294-21.06	4741.197017-33.23	4741.197017-33.23	4741.197017-33.23	4741.197017-33.23
B数据	5001.027941-9.23	4833.571229-19.85	4831.469012-28.36	4831.469012-28.36	4831.469012-28.36	4831.469012-28.36	4831.469012-28.36
C数据	7121.564229-9.66	6922.638375-17.34	6874.844770-27.54	6874.844770-27.54	6840.139143-43.41	6840.139143-43.41	6794.191798-140.73
D数据	6914.990564-8.83	6840.907509-13.59	6780.685138-21.33	6732.429810-37.73	6732.429810-37.73	6732.429810-37.73	6652.869359-124.98
E数据	8625.25396-8.61	8552.185787-15.27	8495.853294-29.08	8495.853294-29.08	8495.853294-29.08	8451.694764-55.27	8355.715742-446.47
总分	6635.604405	6383.446770	6345.5497016	6335.1587806	6328.2176552	6319.3859492	6275.0885856

算法一整体流程

- 思路分析
    - 数据分为在线实例和离线任务，需不需要一起调度?
        - 在线实例的时间贯穿整个时间线，离线任务想对时间较短
        - 自然的想法是在线实例部署好后不再移动，将离线任务塞进去。
            - 类似西瓜和芝麻放一起，先放西瓜再把芝麻倒进去
- 整体流程如下
    1. 在线实例部署
    2. 在线实例迁移
    3. 离线任务部署
    4. ~~离线任务优化（该步骤为5号之后临时加入，代价很大，效果一般）~~

算法－步骤1－在线实例部署

整体流程

- 思路分析
  - 对于已部署的状态，当中有的机器分高，有的机器分低，一个自然的想法是将高分机器和低分机器混合一下。
  - 不断对机器按照得分重新排序，混合高分机器和低分机器得到更低的总分。
  - 如何混合？我采取了穷举的方案，如果实例过多可以控制一下穷举次数
- 在线实例部署整体流程
  - 初始化部署
  - 高低分机器混合迭代过程

算法－步骤1－在线实例部署

初始化部署

- 思路分析
  - 多次试验后估计该步骤对后面的高低分混合策略影响不大，怎么快怎么来
  - 我目前采用的是FirstFitDecreasing策略来初始化，同时考虑初始状态，尽量避免实例移动
    - TODO：这里改用NextFit初始化实例部署效果应该更好，性能应该也不会下降
- 初始化部署流程
  - 根据初始状态部署好实例
  - 对机器按照配置和当前得分从高到低排序，部署目标机器集合为前N个机器（N目前为指定的机器数量）
  - 对所有不在部署目标机器集合上的实例排序
    - App冲突数量大于某个值比如16，冲突数高的排在前面。
    - 如果App冲突数都比较小，那么平均资源消耗比例总和大的排在前面
  - 再使用FirstFitDecreasing策略将实例部署到目标机器集合上
- 该步骤一般耗时1-30秒内，不再在统计结果中单独呈现

算法－步骤1－在线实例部署

高低分机器混合迭代过程

- 高低分机器混合迭代过程
    - 对所有部署机器按照得分排序
    - 按照一高一低的搭配从中随机选择一批机器（这里随机选择是为了避免陷入局部最优，选择一批机器是为了并发以提高性能）
    - 对上一步选择的机器两两分组并行穷举出最优状态。对于无法完全穷举的，限制穷举次数
    - 不断重复上述步骤到一定次数
    - 之前的版本会在这里每隔一定次数做一次最优机器数量调整，详见后文（目前为手工指定，代码中仅包含部分探测代码）
- 算法分析
    - 该步骤为整个算法时间最长的步骤，其时间可以通过迭代步数，迭代效果，迭代时间来控制。
    - 为了得到高分同时尽可能快地结束，比赛期间我手工指定了比较大的迭代步数，通过saveload来节省时间。
    - 穷举方案可以进一步优化，如果实例数量比较大，无法穷举，可以前面一部分大的实例随机放置，剩余的实例穷举。该方案应该是可行的，我已试验过完全随机放置的方案效果还可以，比赛期间没有进一步尝试。

算法－步骤1－在线实例部署

最优机器数量探测

- 思路分析
  - 在线实例调度时，最开始迭代少量次数，评估需要增加的机器数量。加入新的机器，重新迭代更多的次数，再重新评估需要增加的机器数量，如此循环
  - 如何评估需要增加的机器数量?
    - 对部署机器中超过一定CPU分值的进行统计，通过公式计算出在线实例部署需要增加的机器数量，代码中的公式如下
      - h1=超过1.01的机器数，ScaleRatioH1=400（大约为4/(1.01-1)）
      - h2=超过1.1的机器数，ScaleRatioH2＝40（大约为4/(1.1-1)）
      - h3=超过1.2的机器数，ScaleRatioH3＝20（大约为4/(1.2-1)）
      - 部署实例需要增加的机器数量＝(h1-h2)/ScaleRatioH1 + (h2-h3)/ScaleRatioH2 + h3/ScaleRatioH3
    - 在线实例部署的最后一轮（设定迭代次数），计算离线任务单独部署（不考虑离线任务的顺序，直接用BestFit放入）需要的机器数量，增加这个数量一半的机器，对在线调度再进行一轮迭代（待改进）
- 一般情况下自动调整7-8次后，机器数量会比较稳定且效果还可以
- 自动探测的时间大约会比手工设定高3-5倍，且总得分估计会高200左右，目前为手工设定机器数量。
- （复赛早期7000分以上是使用自动探测算法的，之后改为手工指定。目前估计如果改为自动探测，总分在6400-6800左右）

算法－步骤2－在线实例迁移

思路分析

- 思路分析
  - 同配置机器重新映射，将大的实例的初始机器直接映射到目标机器上，避免移动
  - 空闲机器可以用作临时中转，但是如果在最终轮之前将空闲机器中的实例迁出，那么这一轮中的空闲机器仍然没法利用，所以空闲机器中中转的实例留在最终轮直接迁到目标机器上
  - 第一轮尝试将实例迁移到目标机器，如果失败则尝试迁移到空闲机器
  - 第二轮移动当中对每个不能直接移到目标机器的实例，下一轮如果按照最终状态，该实例保持不动是否会引起冲突，如果冲突就尝试将其移走，移动的目标机器要同时满足本轮和下一轮的约束
  - 第三轮能做的只有将剩下的实例都部署到目标机器上
- 进一步改进的地方
  - TODO：第一轮移动当中，对每个不能直接移动到目标机器的实例且没有迁移到空闲机器的实例，如果下一轮可以移动到目标机器，那么保持当前位置不变，并且占住下一轮的目标位置的坑以及下一轮当前位置的坑。然后再重新考虑所有剩下的实例，下一轮如果按照最终状态，该实例保持不动是否会引起冲突，如果冲突就尝试将其移走，移动的目标机器要同时满足本轮和下一轮的约束（目前的方案虽然已经足够，但还是要再优化一下）
  - 可以在在线实例调度阶段就固定住一大批实例，减少可能的移动需求。比如每台机器固定住一个大的实例以及一批小的实例
  - 可以在调度过程中尽量减少同配置机器间的实例移动

算法－步骤2－在线实例迁移

算法流程

- 准备阶段：通过交换同配置的机器，将较大的实例部署的机器映射到初始状态部署的机器，避免较大实例的迁移
- 第一轮
    - 对实例从大到小遍历（排序规则请参考在线实例部署－初始化部署），尝试将不在目标机器上的实例迁移到目标机器，如果失败则NextFit到空闲机器，并在原机器留下幽灵实例（这里目前实现的是FirstFit，目前的数据集不需要，待改进）
    - 再次对全部实例从大到小遍历，尝试将实例NextFit到空闲机器，使空闲机器饱和（本步骤暂未实现，目前的数据集不需要）
- 第二轮
    - 对实例从大到小遍历，尝试将不在空闲机器中的实例部署到目标机器
    - 复制最终状态镜像，再次对实例从大到小遍历
        - 尝试将不在目标机器且不在空闲机器上的实例添加到最终状态对应的相同机器上，若添加成功，该实例维持当前位置不变，同时将该实例添加到最终状态的镜像中
        - 若添加失败则再尝试NextFit到其他机器上（跳过与最终状态产生冲突的机器）并留下幽灵实例，同时将该实例添加到最终状态的镜像中
    - 清空所有幽灵实例
- 第三轮：将剩下的实例都移到目标机器上即可

算法－步骤3－离线任务部署

思路分析

- 思路分析
  - 如何处理离线任务的前后顺序？在前面的任务先部署
  - 任务如何部署到机器上？每个机器找到第一个可以放置的位置（FirstFit）
  - 如何从多个可部署的机器中选择最好的（BestFit）？选择时间最早的机器
  - 上面的步骤会出现一个问题，任务会倾向于堆积到较早的时间点上执行，导致CPU得分升高，如何解决？部署任务时可以控制CPU的使用上限，如果全部机器部署失败，再将CPU上限逐步上调，直到部署成功
  - 上一步的策略还有一个问题，可能任务被某些特殊情形较早地部署到了最大开始时间点上。考虑到机器数量以千计，另外CPU上调步长也比较大（大约92*0.005=0.46左右），这个问题相对影响会比较小。

算法－步骤3－离线任务部署

算法流程

- 算法流程
  - 对任务按照最早结束时间从早到晚排序
    - 首先保证父任务在前，子任务在后
    - 最早结束时间比较接近的，按照任务的面积＝CPU＊执行时间排序，大的在前
  - 设置CPU初始限制为0.5
    - 针对当前CPU限制，对每个机器FirstFit任务开始时间
    - 选择最早开始时间的机器（BestFit）
    - 如果所有机器均无法部署，上调CPU限制，继续尝试部署，直到CPU限制为1
- 算法分析
  - CPU初始限制可以优化为当前所有机器CPU均值相关的值

~~算法—步骤4—离线任务优化~~

- ~~算法分析~~

  - ~~该步骤是5号开始临时加上的，因为之前的步骤在执行一个多小时后便终止了。增加该步骤主要是为了用时间堆更高的分数（大约提高50-100分）~~

  - ~~主要思路是对每个机器找出最高的CPU时间点，将该时间点相关的任务移出并重新BestFit到其他位置或者其他机器当中，BestFit的目标是分数降的最多~~

  - ~~由于增加了该步骤，最优机器数量可以略微下调(100-200台)以提高分数~~

  - ~~目前该步骤执行时间较长，有很多待改进的地方。比如仅BestFit到CPU最低的部分机器，限制每个时间点每次迁移的任务数量以及对任务进行排序，不仅仅在每个机器的CPU最高点执行优化避免局部最优等~~

TODO

- 最优机器数量改为自动探测
    - 目前想到的最简单的方法是直接在几十台机器上同时运行不同的机器数配置，选最好的
- 优化实例部署时的高低分混合穷举策略，穷举次数限制会漏掉一些可能性，应改为一部分随机，一部分穷举。
- 优化实例部署时混合各机器组合的耗时相差较大，不能充分利用CPU并行计算，待优化
- 实例迁移优化，目前首轮迁移策略太粗糙，空闲机器的使用策略也很粗糙
- 完善实例调度循环的结束条件，可根据运行时间，迭代步骤，迭代效果等条件来判断。同时将数据E的输出特殊逻辑改掉。（这两个问题是等价的）
- 任务部署结束后的优化空间还很大，继续寻找之后的优化策略

复赛评审之后的文档及代码改动说明

- 本文档在提交评审的文档上修改
- 复赛提交评审之后继续对算法进行了少量改进，改进主要有3点
  - 在实例初始化部署时不再对所有实例进行FirstFit部署，改为只FirstFit不在目标部署机器上的实例。这样统一了所有数据集的初始化部署，不再对数据E做特殊处理。这个改动也会提升分数和性能，原因是FirstFit部署全部实例的话，初始状态会处在比较差的状态。。（代码在resource_management.go中的firstFitInstances函数中）
    - TODO：这里改用NextFit初始化实例部署效果应该更好，性能应该也不会下降
  - 在任务BestFit算法对每台机器检测最早可部署时间的循环中，将两个加法和一个乘法操作移到循环外面。任务部署的速度提升大约20-40%（代码在machine.go中的CanFirstFitJob函数中）
    - TODO：这里可以进一步把CanFirstFitJob函数逻辑直接实现在BestFit循环体中
  - 实例迁移的第一轮之前过度迁移了，将该部分代码删掉了，大幅度降低了迁移失败的可能性

结果简述

- 排行榜的成绩运行时间为1到2天，机型为MacBook（AB数据集）＋阿里云12核24G（CD数据集）

- 自动探测最优机器数量得分大概在6400－6800，时间消耗大约3-5倍，后面给出的是手工指定机器数量的得分

- 9月5号之后的版本为了追求更高的分数，任务部署结束后（一般在40分钟到80分钟），使用了无限循环迭代继续优化，耗时很长（约一天到两天），分数提升也有限（约50-100）

- 所以准备了两个版本的迭代结果，这两个版本代码是相同的，主要区别是机器数量不同。因为无限迭代后，略微调低机器数量（约100-200）能提高一点分数。

- 9月5号之后拥有了高配服务器，实例部署的轮数也调高了一倍（该因素大约提升50-100分）

- 本文中的数据为MacBook重新跑的数据（服务器实例已释放）

- 结果中黄色高亮的为复赛评审之后再改进后的数据

MacBook调度记录-A数据

黄色背景为提交评审后再优化后的数据，灰色背景为优化前的数据

数据集	机器数量	实例调度轮数	实例部署时间(秒)	实例迁移时间(秒)	任务部署时间(秒)	总时间(秒)	总时间(分)	最终得分
A	4700	0	29.913772	1.155898	652.946079	684.407725	11.41	17713.589336
			0.721993	0.656098	431.554351	433.394925	7.22	22196.577452
A	4700	128	199.269534	1.809333	511.434774	712.965997	11.88	7433.565920
			124.941329	0.645253	322.699621	448.727750	7.48	7541.406814
A	4700	256	305.530833	1.904631	478.359077	786.287006	13.10	5571.861500
			224.492663	0.661123	309.229758	534.827467	8.91	5515.185331
A	4700	512	524.285198	1.838116	484.112334	1010.714911	16.85	4982.964160
			422.453898	1.093199	314.089133	738.087036	12.30	4900.158281
A	4700	1024	827.130299	1.772108	505.607423	1334.881204	22.25	4888.079375
			636.843974	0.867670	294.338352	932.510741	15.54	4767.930950
A	4700	2048	1417.212085	1.588257	497.681490	1916.857025	31.95	4857.038284
			960.074717	1.136055	302.399852	1264.051298	21.06	4744.896294
A	4700	4096	2347.601492	1.811281	484.472356	2834.375666	47.24	4838.205156
			1709.013890	1.391486	283.077445	1993.971138	33.23	4741.197017

MacBook调度记录-B数据

黄色背景为提交评审后再优化后的数据，灰色背景为优化前的数据

数据集	机器数量	实例调度轮数	实例部署时间(秒)	实例迁移时间(秒)	任务部署时间(秒)	总时间(秒)	总时间(分)	最终得分
B	4800	0	34.116714	1.319207	435.421409	471.241250	7.85	17844.984522
			0.889532	0.589371	277.968850	279.826061	4.66	22158.915489
B	4800	128	208.388321	1.530756	361.513194	571.860925	9.53	7478.989039
			121.788417	0.667685	221.832427	344.766523	5.75	7714.558473
B	4800	256	318.915288	2.074480	308.097995	629.554010	10.49	5578.969039
			218.513139	0.722860	195.501663	415.150380	6.92	5640.356914
B	4800	512	529.401273	1.932764	308.581208	840.379605	14.00	5017.208109
			368.704748	0.808983	184.089481	553.897979	9.23	5001.027941
B	4800	1024	858.237347	2.086325	315.334757	1176.112546	19.60	4932.909585
			630.441923	0.823743	178.340741	809.995052	13.50	4848.031481
B	4800	2048	1499.544408	2.090855	336.948499	1839.149521	30.65	4907.384713
			1012.936596	1.035207	176.830422	1191.289321	19.85	4833.571229
B	4800	4096	2687.716556	1.671105	312.887420	3002.572144	50.04	4906.181764
			1512.138413	1.330751	187.840772	1701.656918	28.36	4831.469012

MacBook调度记录-C数据

黄色背景为提交评审后再优化后的数据，灰色背景为优化前的数据

0实例迭代无结果的原因是初始化直接使用了FFD导致迁移困难，已改进

数据集	机器数量	实例调度轮数	实例部署时间(秒)	实例迁移时间(秒)	任务部署时间(秒)	总时间(秒)	总时间(分)	最终得分
C	6300	0	无	无	无	无	无	无
			1.248961	0.716887	376.562317	378.990216	6.43	22469.208214
C	6300	128	75.638290	53.994838	458.120086	588.571129	9.81	10427.640961
			74.429778	1.563695	329.371271	405.862178	6.76	10614.820529
C	6300	256	109.838382	52.279229	462.568912	625.233841	10.42	8305.074517
			144.220781	4.610957	302.125320	451.313895	7.52	7952.478678
C	6300	512	190.789662	54.896719	500.773133	746.823501	12.45	7645.692353
			256.924707	12.687352	309.619058	579.609056	9.66	7121.564229
C	6300	1024	291.476140	51.351733	511.840203	855.154148	14.25	7408.832100
			411.795595	21.085966	356.547025	789.811424	13.16	6986.227240
C	6300	2048	469.079693	49.884753	445.239747	964.867274	16.08	7255.088509
			697.345183	23.749786	319.261609	1040.723015	17.34	6922.638375
C	6300	4096	764.368762	51.738775	465.776140	1282.433072	21.37	7128.414001
			1281.134939	26.731334	344.082595	1652.398035	27.54	6874.844770
C	6300	8192	1251.880627	47.983721	474.225162	1774.461935	29.57	7003.468028
			2250.703525	27.807161	325.481897	2604.387962	43.41	6840.139143
C	6300	16384	2408.991943	50.244930	458.500438	2918.329435	48.64	6887.808126
			4313.967309	29.717141	317.388778	4661.398486	77.69	6816.402538
C	6300	32768	4578.732932	45.397828	470.465107	5094.934036	84.92	6803.484118
			8051.174664	31.902389	360.216355	8443.635832	140.73	6794.191798

MacBook调度记录-D数据

黄色背景为提交评审后再优化后的数据，灰色背景为优化前的数据

0实例迭代无结果的原因是初始化直接使用了FFD导致迁移困难，已改进

数据集	机器数量	实例调度轮数	实例部署时间(秒)	实例迁移时间(秒)	任务部署时间(秒)	总时间(秒)	总时间(分)	最终得分
D	6000	0	无	无	无	无	无	无
			1.285967	0.644980	174.286340	176.667769	2.94	22237.952387
D	6000	128	72.073909	48.649827	216.052334	337.149208	5.62	10205.804937
			74.979949	1.590796	147.894143	224.857975	3.75	10373.828000
D	6000	256	118.146437	56.051192	221.689752	396.512785	6.61	8232.904285
			128.507738	4.528106	139.470453	272.954951	4.55	7809.867639
D	6000	512	196.260234	57.874421	227.247719	481.745849	8.03	7602.407402
			209.980861	11.863240	144.718965	366.924148	6.12	7074.453930
D	6000	1024	289.261848	54.450596	237.784217	582.199362	9.70	7357.884997
			366.081059	18.878481	144.406476	529.825881	8.83	6914.990564
D	6000	2048	424.241994	51.641796	225.072874	701.315843	11.69	7193.258079
			635.534264	23.037194	156.744585	815.669139	13.59	6840.907509
D	6000	4096	719.190785	50.537467	213.871917	984.250232	16.40	7042.623250
			1107.210494	26.491810	145.986376	1279.985121	21.33	6780.685138
D	6000	8192	1241.335821	53.127656	253.604467	1548.845381	25.81	6891.187526
			2078.878665	30.197798	154.468824	2263.880376	37.73	6732.429810
D	6000	16384	2286.826602	51.892938	208.251657	2547.495550	42.46	6750.059954
			3854.753150	28.798259	150.722056	4034.588875	67.24	6688.949444
D	6000	32768	4452.590432	47.325674	195.018099	4695.388831	78.25	6681.770625
			7313.410782	33.438854	151.558718	7498.726964	124.98	6652.869359

MacBook调度记录-E数据

黄色背景为提交评审后再优化后的数据，灰色背景为优化前的数据

数据集	机器数量	实例调度轮数	实例部署时间(秒)	实例迁移时间(秒)	任务部署时间(秒)	总时间(秒)	总时间(分)	最终得分
E	8000	0	1.691464	0.623488	无	2.816368	0.05	20418.456912
			0.110440	0.451061		1.505303	0.03	20418.456912
E	8000	128	51.406253	1.998590		54.266263	0.90	11907.327560
			59.802334	1.605809		62.336604	1.04	11993.259179
E	8000	256	91.165904	11.994638		103.980731	1.73	9771.865982
			102.643320	0.877256		104.377047	1.74	9809.537437
E	8000	512	168.518096	25.410305		194.936893	3.25	8909.990173
			180.485103	0.716940		182.036052	3.03	8908.624714
E	8000	1024	293.630226	37.009122		331.409148	5.52	8725.593702
			296.206628	0.704098		297.704070	4.96	8722.592967
E	8000	2048	499.213119	42.583885		542.689538	9.04	8628.884347
			515.256924	0.735319		516.813800	8.61	8625.253960
E	8000	4096	942.982509	49.536332		993.497561	16.56	8556.978685
			915.039633	0.547137		916.349368	15.27	8552.185787
E	8000	8192	1717.857115	52.278071		1770.918140	29.52	8499.219433
			1743.430475	0.697013		1744.901713	29.08	8495.853294
E	8000	16384	3314.600345	0.734303		3316.154382	55.27	8451.694764
E	8000	32768	6381.070349	0.683781		6382.470921	106.37	8415.028050
E	8000	135168	26785.897947	0.745624		26787.380682	446.47	8355.715742

9月5号之后手工调优机器数量记录（排行榜成绩）

总提升约100分,排行榜成绩为6254.52

下面是原始记录，服务器实例已释放，时间信息缺失（约1-2天），最终迭代前一般需要40-80分钟，之后是无限循环

A_8192表示A数据实例调度8192轮，因新买了12核服务器，实例迭代轮数较9月5号之前翻了一倍

红色为已放弃，绿色为当前在计算，黄色为最优结果，灰色为尚未进行

数据E的排行榜成绩为8448.281015，计算时间约5小时，比赛后期没有重新运行过

数据集	机器数量	实例部署得分	任务初步部署得分	迭代终止
A_8192	4200	4897.678876		
A_8192	4300	4767.856584		
A_8192	4400	4689.814810	5631.936317	5432.876545
A_8192	4500	4674.059846	5222.442870	
A_8192	4600	4661.360834	4939.535729	
A_8192	4650	4683.971828	4863.988346	
A_8192	4700	4714.400140	4830.088467	4771.051787
A_8192	4725			
B_8192	4200	4898.047266		
B_8192	4300	4772.035342		
B_8192	4400	4690.828378	5954.248145	5421.254886
B_8192	4500	4674.528984	5517.508075	5201.020846
B_8192	4600	4662.560520	5167.709408	5135.615933
B_8192	4650			
B_8192	4700	4714.443412	4966.464530	4803.619554
B_8192	4750	4753.549915	4908.842494	4848.368495

数据集	机器数量	实例部署得分	任务初步部署得分	迭代终止
C_32768	5600	6507.555002	7217.644984	7198.204350
C_32768	5700	6511.950722	7100.706595	7115.542470
C_32768	5800	6524.654437	7007.673792	6890.793342
C_32768	5900	6546.674066	6980.837402	
C_32768	6000	6570.264817	6870.808606	6656.862920
C_32768	6100	6605.161694	6843.107445	6703.762288
C_32768	6200	6634.391013	6817.917476	6798.291623
D_32768	5500	6498.501294	6926.043763	6646.373146
D_32768	5600	6495.446995	6860.069470	6832.451173
D_32768	5650	6510.518557	6814.709059	6787.652530
D_32768	5700	6515.505975	6790.080385	6590.440705
D_32768	5800	6530.252437	6739.008299	24-6602.529633
D_32768	5900	6547.483028	6700.323503	27-6680.865277

Thank you !