# 介绍

随着物联网和云计算的发展，越来越多的计算任务通过云端在数据中心上运行，导致数据中心的计算需求急速上升，其能耗也随之增加，据15年3月23日工信部联合国家能源局、国家机关事务管理局印发的《国家绿色数据中心试点工作方案》披露：我国数据中心发展迅猛，总量已超过40万个，年耗电量超过全社会用电量的1.5%，其中大多数数据中心的PUE（平均电能使用效率）仍普遍大于2.2，与国际先进水平相比有较大差距。所以提升数据中心的效率是当务之急，能产生很大的社会效益和经济效益。

本文是根据阿里巴巴2017年8月1313个节点构成的数据中心，通过研究这些节点24小时内的CPU利用率，内存利用率，再加上这段时间节点上运行的在线任务和离线任务及其状态。我们发现不同类型的节点上其CPU利用率和内存利用率的相关性有较大差别，这样我们就可以根据任务的不同对任务分到哪个节点进行更合理的划分。通过对节点失败特点的分析，我们可以通过更好的任务调配避免节点失败，这样就可以间接提升节点利用率。

工作负载特征：我们如何通过代表性的方式模拟各种生产工作量来调度阿里巴巴的工作负载。将工作负载分配给机器和CPU内核的新算法。 我们如何将工作量分配和重新调整到不同的机器和CPU，以获得更好的资源利用率和可接受的资源争用。在线服务和批处理作业调度程序合作：如何调整在线服务和批处理作业之间的资源分配，以提高批处理作业的吞吐量，同时保持可接受的服务质量和快速的在线服务故障恢复。

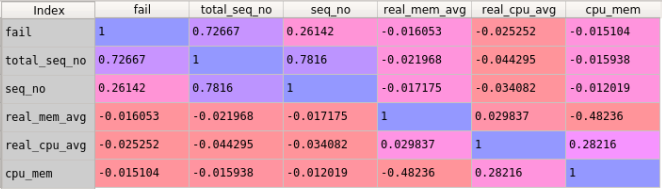
本文之后的章节如下。第二章是对节点上负载的分析。第三章是节点的分类和聚类。第四章是对如何优化数据中心效率的方法。第五章是总结。

# 2. 节点上负载的分析

要了解节点的运行状态和节点上出现的各种现象，我们首先要知道节点上运行的负载是什么类型的，根据阿里巴巴公布的数据显示，负载被分为两部分，分别是批处理任务和在线任务。批处理任务被细分成instance和task两部分来描述，而在线任务只分成instance，下面我们根据批处理任务和在线两部分对节点上运行的负载做一个分析。

## 2.1 节点上批处理任务的分析

用户以Job的形式提交批处理工作负载。作业包含多个任务，不同的任务执行不同的计算逻辑。 根据数据依赖性，任务形成一个有向无环图。 实例是批处理工作负载的最小调度单位。 任务中的所有实例都使用相同的资源请求执行完全相同的二进制文件，但输入数据不同。

由于最小的分配单元是instance，所以我们可以根据instance的特征对其进行分类，从batch\_instance.csv中提取出了一个instance使用的CPU数量，内存大小用kmeans方法对其做聚类计算得到instance的分类结果，依此结果对instance做更加合理的调度。

我们取出上图中可能和instance相关的一些特征，分别是instance失败，总的队列个数。Batch\_instance中失败的instance个数是209168个，总的instance是16094665，比例为1.29%

根据给出的数据集batch\_task.csv中每个任务的各项信息，如：cpu\_request，memory\_request，number\_of\_instance和把task按job分类，依此去分析哪些特征是和task的失败紧密相关的。

|  |  |  |  |
| --- | --- | --- | --- |
|  | Fail | Total | Rate |
| 1 | 209168 | 16094665 | 1.3% |
| 2 | 35276 | 147778 | 23.8% |
| 3 | 19134 | 35276 | 54.2% |
| 4 | 5599 | 19087 | 29.3% |
| 5 | 175 | 5112 | 3.4% |
| 6 | 57 | 172 | 33.1% |
| 7 | 20 | 55 | 36.4% |
| 8 | 7 | 19 | 36.8% |
| 9 | 1 | 2 | 50.0% |
| 10 | 0 | 1 | 0.0% |

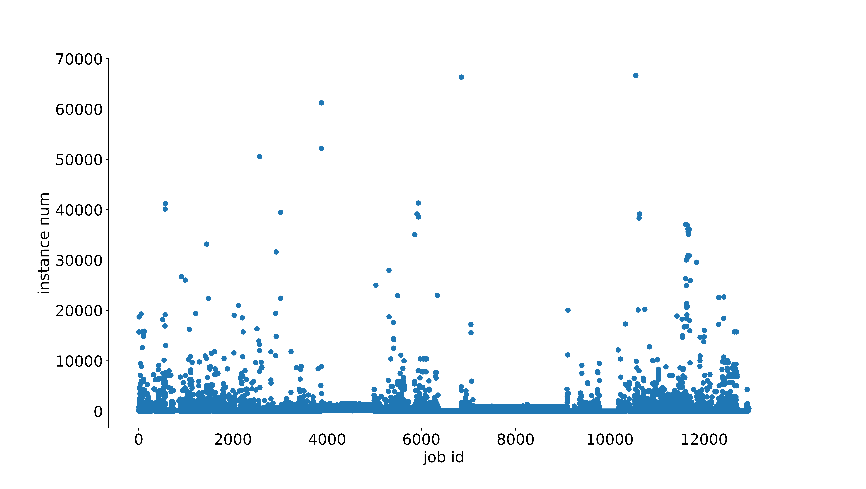
失效了又重新执行成功的instance的运行时间跟总体的instance的平均运行时间相比，大小？？

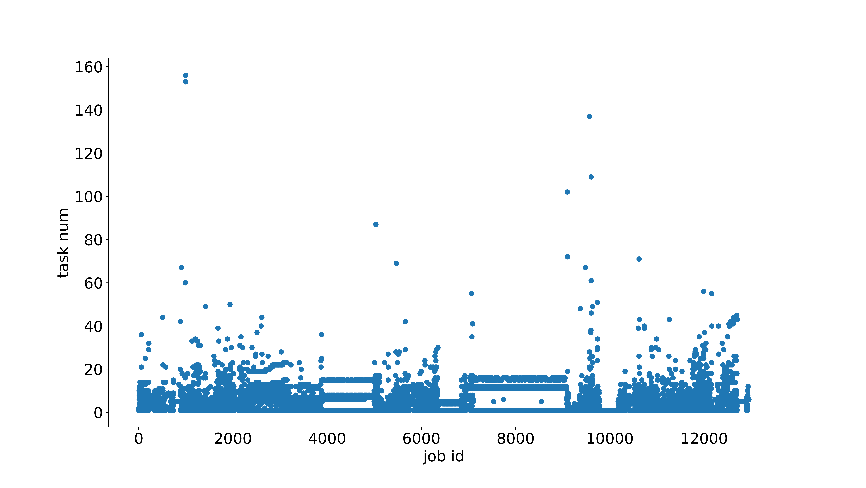
失效一次后也没有重新执行的instances的统计？第一次失效209168个instance，有209168-147778=61390个直接丢弃了。这些丢弃的统计？在哪些machineID上？属于哪些task？这些task有没有重启（taskID没变，但是执行起始时间戳变了）？如何判断一个instance重启了？id是不是变了？

上表揭示了阿里巴巴数据中心处理批处理任务时的instance重试的数据，第一列是一个instance运行的次数，第二列是运行第几次失败的概率，第三列是总的运行次数，第四列的是两者的比例。

从batch\_task中找到一个失败的任务如task id为153的这个任务，再去batch\_instance中找到所有这个任务的**实例**，发现总共401个task只有一个instance是失败状态，而且这个instance并没有再次运行，导致了整体任务的失败，在这个instance已经确定是失败的情况下，这个task后面的instance又继续运行造成了资源的浪费。我们数据集的整个task数量是80552个，其中失败的是个数是1126个。

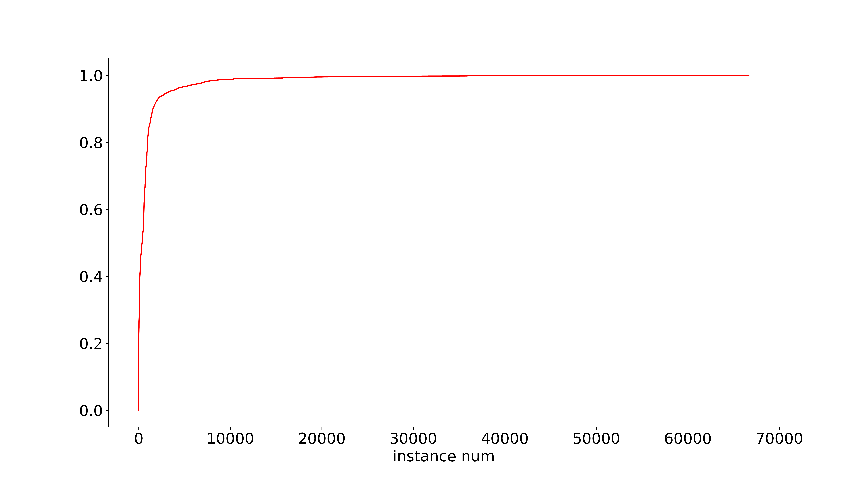
我们已有的数据，我们知道对于不同的Job运行的instance和task的数目不同，为了得出每个Job上各自运行了多少instance和task,我们对已有数据进行了统计。统计结果如图12，图13所示。

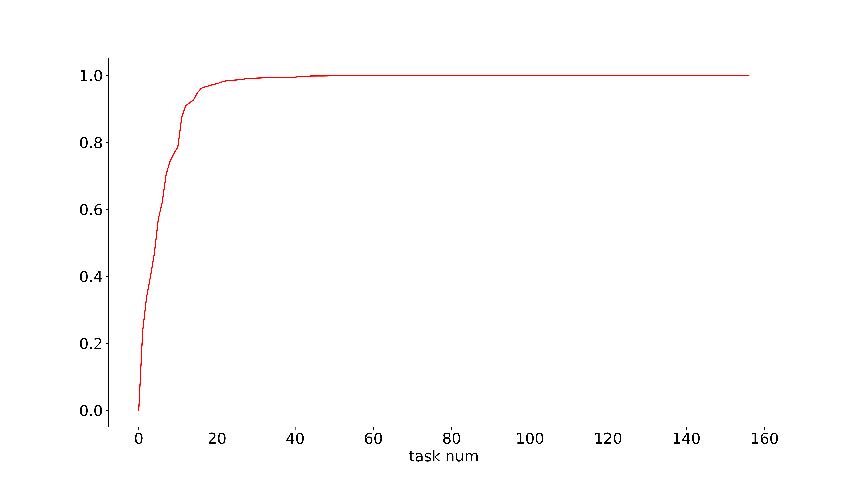




每个taskID对应的instance数目？？

为进一步分析每个Job的instance number和task number的分布情况，我们做出了其分布的CDF图，如图14，图15所示。





从图12可以得出，对于大多数的Job，其运行的instance number集中在0-10000。图14也说明了这一情况，当instance\_number在0-10000时，概率曲线非常陡峭，当达到20000时，其概率已经趋近1。

对于Job运行的task number，同instance number类似，从图13可以得出，对于大多数的Job，其运行的task number集中在0-20。从图15可以看出，当task\_num在0-20时，概率曲线非常陡峭，当达到40时，其概率也趋紧1。

因为每个Job上都运行了一定数目的instance和task，所以我们分析了每个Job运行不同数目instance/task时的运行时间，结果如图16,图17所示。从图16可以看出当Job运行的instance number在0-1000时，所需时间为一个增长的状态，而当instance number大于1000时，随着number的增加，运行时间明显下降。

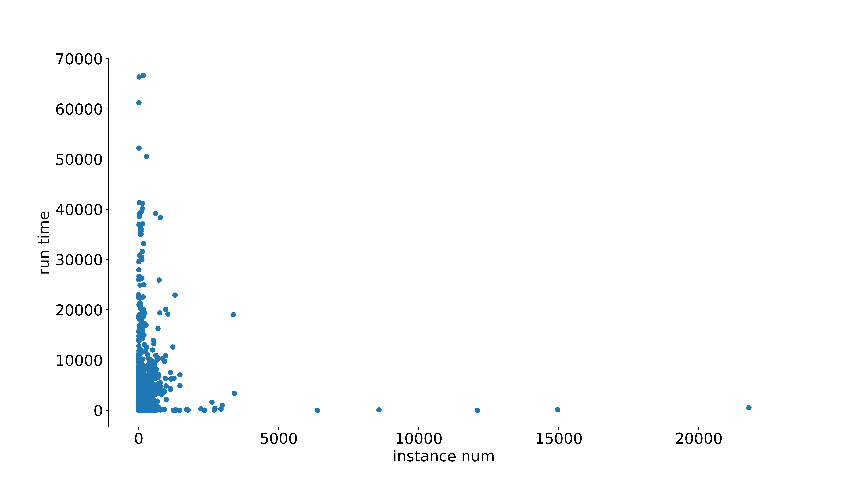


Figure 16 不同instance number运行时间

从图17我么可以看出，当Job的task number在0-5时，Job的运行时间比较长，而在其他number时，Job的运行时间都集中在0-2000s之间。

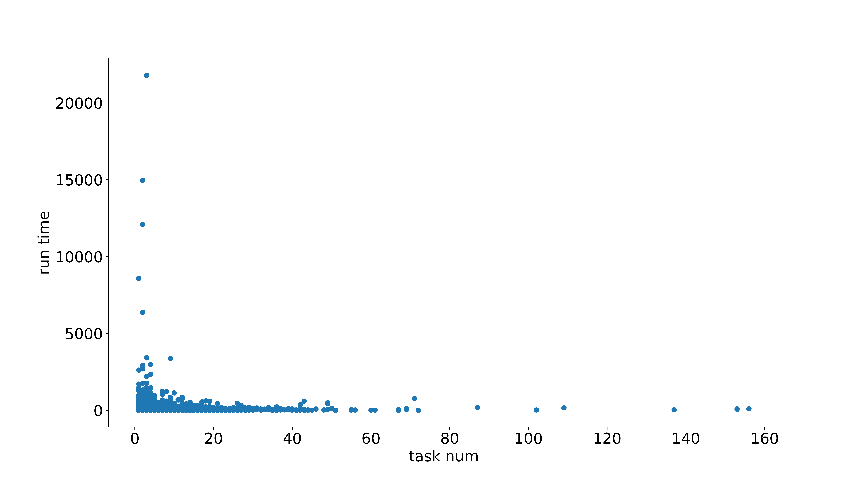


Figure 17 不同task number运行时间

最后我们分析了每个Job的完成时间，分析结果如图18所示。

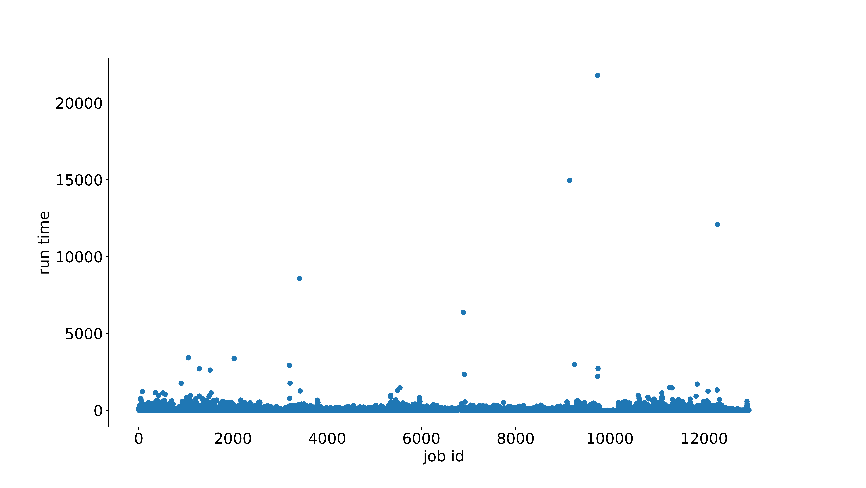


Figure 18 不同job的完成时间

把大部分的run time比较集中的job绘制出来（分段500以下、1000以下、2000、）

同样，我们对Job随时间变化的失效情况也进行了分析。图5描述了随时间变化，失效Job的分布。图6描述了随时间的变化，失效Job个数的分布。

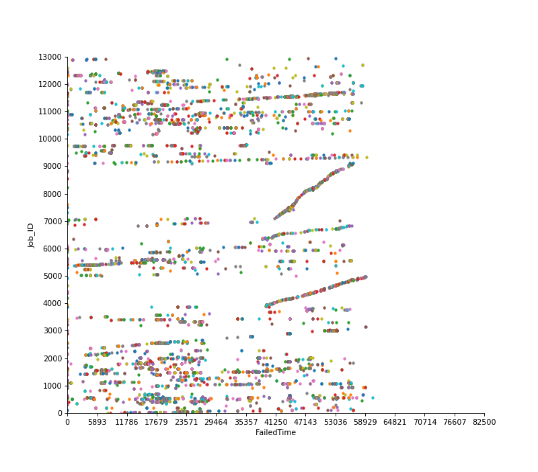


Figure 5 不同时间点失效Job的分布

没有失效的jobid的分布？在哪些machineID上？

随时间集中失效的几条线的jobid、machineid提出来看一下？

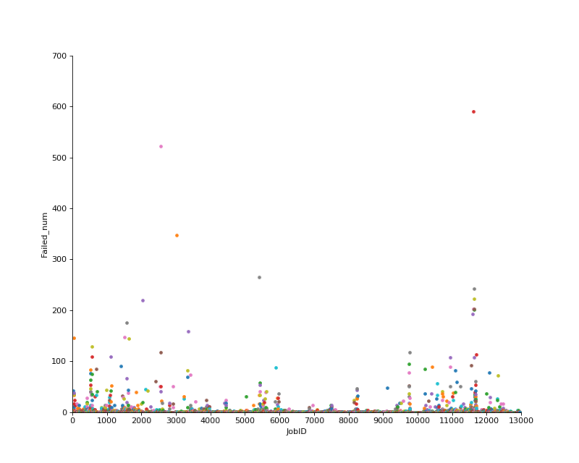


Figure 6 不同时间点失效Job次数的分布

## 2.2 在线任务分析

# 3. 节点的分类和聚类

为了对节点进行分类，我们对各节点随时间变化的cpu和memory利用率进行了分析，并做出了节点cpu\_usage和memory\_usage随时间的变化曲线。根据这些曲线特征，我们将所有的节点分成10类。下面将对这10类节点进行详细的分析。

## 3.1 不同类型节点的描述

3.1.1 cpu\_usage/memory\_usage均属于正常波动。1113个AB。例1号节点，其cpu\_usage/memory\_usage随时间变化曲线如下所示。

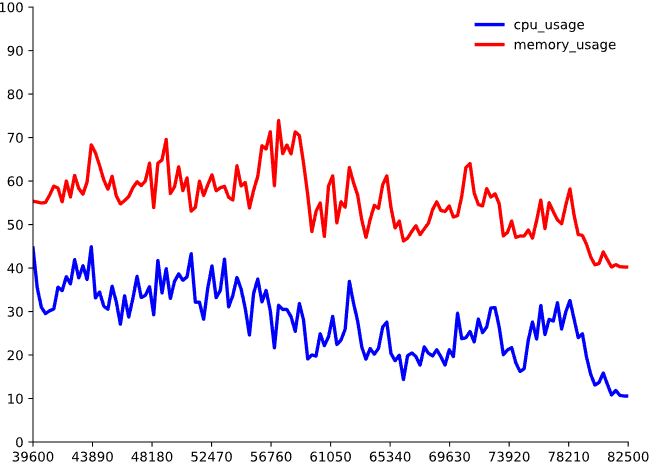


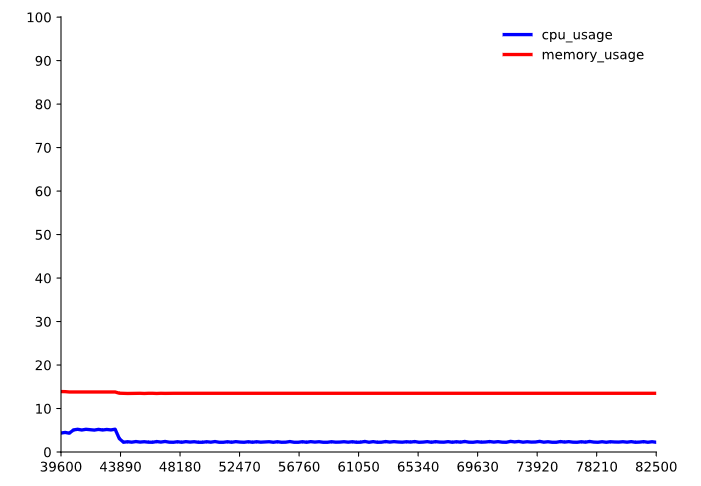
图 1

3.1.2 cpu\_usage/memory\_usage在52000s后均波动较小。154个C。例401号节点，其cpu\_usage/memory\_usage随时间变化曲线如下所示。

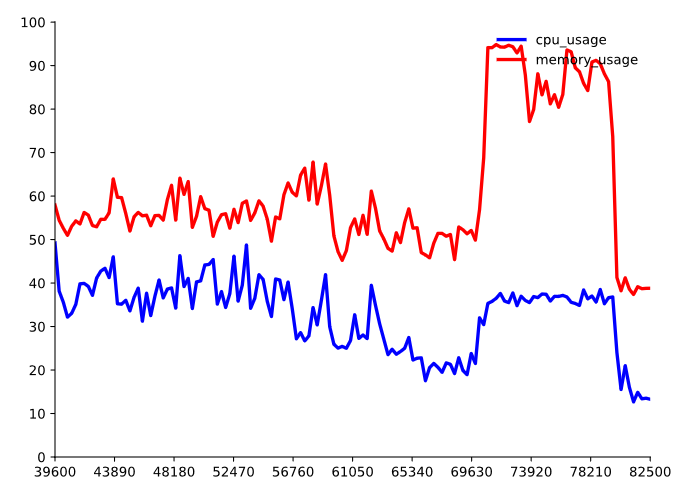
对对对


3.1.3 cpu\_usage/memory\_usage数据缺失。3个D

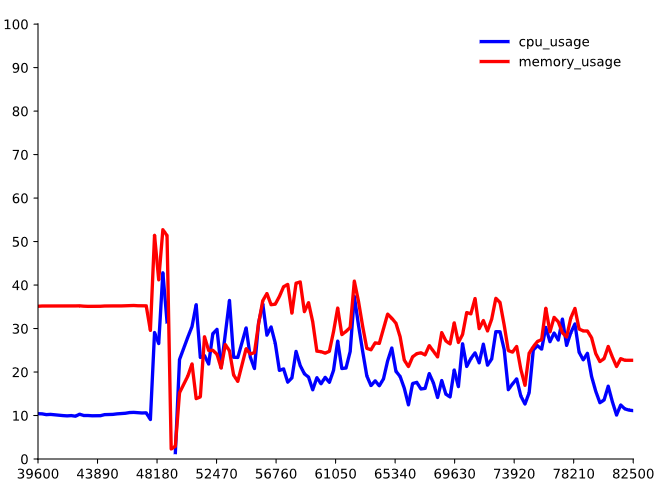
3.1.4 cpu\_usage/memory\_usage全程无波动。17个E。例1075号节点，其cpu\_usage/memory\_usage随时间变化曲线如下所示。



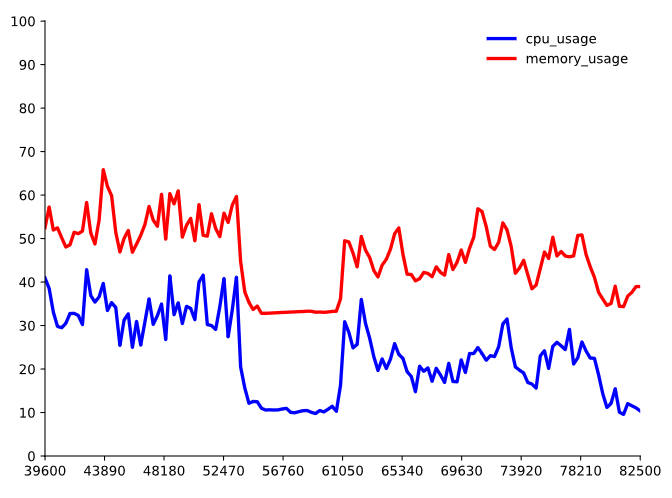
3.1.5 cpu\_usage/memory\_usage在70000s左右时出现大概10000s的尖峰。21个FGH。例232号节点，其cpu\_usage/memory\_usage随时间变化曲线如下所示。



3.1.6 cpu\_usage/memory\_usage在前10000s波动较小。1个I。例618号节点，其cpu\_usage/memory\_usage随时间变化曲线如下所示。



3.1.7 cpu\_usage/memory\_usage在52000s左右出现一段5000s的下降之后水平。4个G。例683号节点，其cpu\_usage/memory\_usage随时间变化曲线如下所示。



## 3.2不同类型节点cpu\_usage/memory\_usage分析

为了分析各类节点cpu\_usage和memory\_usage的特点，我们计算了各类节点cpu\_usage和memory\_usage的平均值,中位值和方差，具体数字如表2所示，根据表2，我们画出了各类节点cpu和memory平均利用率的柱形图，如图11（橙色和绿色柱形图）以供我们进行分析。

Table 2 不同类型节点cpu\_usage和memory\_usage

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 类型 | 占比 | cpu\_\_mean | mem\_usage  \_mean | cpu\_usage  \_median | mem\_usage  \_median | cpu\_usage  \_std | mem\_usage  \_std |
| A | XXX% | 27.4 | 50.6 | 27.2 | 52.6 | 9.8 | 12.4 |
| B |  | 22.8 | 58.6 | 22.1 | 58.6 | 4.6 | 9.6 |
| C |  | 21.1 | 39.5 | 15.1 | 35.7 | 13.3 | 11.6 |
| D |  | 0 | 0 | 0 | 0 | 0 | 0 |
| E |  | 9.26 | 28.0 | 9.7 | 30.5 | 6.6 | 12.8 |
| F |  | 31.5 | 61.6 | 33.2 | 57.0 | 9.5 | 15.9 |
| G |  | 26.9 | 45.1 | 24.0 | 44.4 | 14.9 | 16.4 |
| H |  | 20.2 | 34.8 | 19.1 | 31.3 | 11.5 | 14.5 |
| I |  | 19.7 | 29.6 | 19.0 | 29.5 | 7.6 | 7.4 |
| J |  | 24.7 | 46.2 | 24.4 | 46.3 | 9.1 | 8.4 |

我们对III节C部分的结果进一步进行了统计分析，根据machine cpu利用率和memory利用率的相关系数，我们将machine分为3类：

1. cpu利用率和memory利用率高度相关的节点：当machine cpu利用率和memory利用率的相关系数α>=0.85，我们将该类节点称为高度相关的节点。即该类节点的cpu利用率和memory利用率的变化方向和趋势密切相关。
2. cpu利用率和memory利用率负相关的节点：当machine cpu利用率和memory利用率的相关系数α<0，我们将该类节点称为负相关的节点。即该类节点的cpu利用率和memory利用率的变化趋势相反。
3. cpu利用率和memory利用率低度相关的节点：当machine cpu利用率和memory利用率的相关系数0<α<0.85，我们将该类节点称为低度相关的节点。即该类节点cpu利用率和memory利用率虽然存在联系，但联系的紧密程度比较弱。

为了更加方便的对节点进行分析，我们记录了每个节点的所属类型，并且计算了1313个节点中各类节点的个数，其中a类节点有198个，b类节点有6个，c类节点有1109个。

根据3类节点的各自特点，为了提供工作负载整合机会，我们主要分析a类节点。

从a类节点的统计结果，我们可以发现节点并非是单独工作的，节点88-127，节点261-296，节点830-906为分别为3个集群。集群内的节点运行同类型的job，因为a类节点cpu和memory的利用率是高度相关的，所以在同一个集群内，整合节点后的cpu，memory的使用率是可以计算的，根据计算结果，在资源足够使用的情况下，将节点进行整合，从而最小化节点使用数目。

除此外，根据以上的分类结果，在分配不同类型的job时，我们可以根据节点的特点做出一个更加合理的分配决策。例如，当job类型为计算型或者内存型时，尽可能的不将其分配给a类节点，从而在cpu和memory之间做出一个折中。

## 3.3 节点CPU和内存的关系

这节我们将针对节点的资源利用情况开展研究，从而在各类资源取得均衡。我们根据节点的cpu和memory的利用率，分析了二者之间的关系。

* + 1. CPU\_usage与memory\_usage的相关系数

根据节点的cpu和memory的利用率，我们计算了cpu利用率与memory利用率的相关系数，来评估节点资源利用之间的关系。计算结果如图12所示。从图可以看出，所求的相关参数在[-1,1]之间，节点cpu和memory的利用率既有正相关也可有负相关，但大多数的节点分布在0.6-1之间，整体上看，CPU和memory的变化是正相关的。

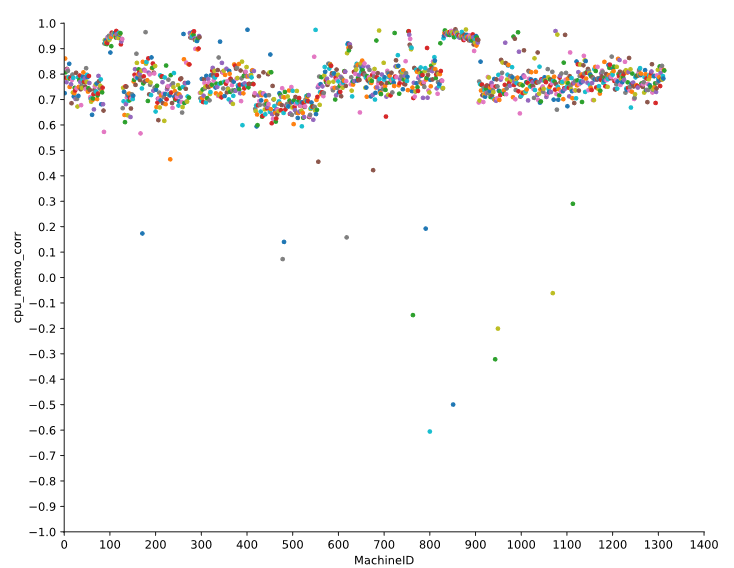


Figure 12 各节点cpu利用率与memory利用率的相关参数

相关系数比较高的节点上的job、task、instance再分类统计？

3.3.2 B部分所介绍10类节点的cpu\_usage和memory\_usage的差值

为进一步分析各类型节点cpu\_usage和memory\_usage之间存在的关系，我们计算了二者之间的差值，并根据计算的结果，得出了各类型节点cpu\_usage和memory\_usage差值的特征。

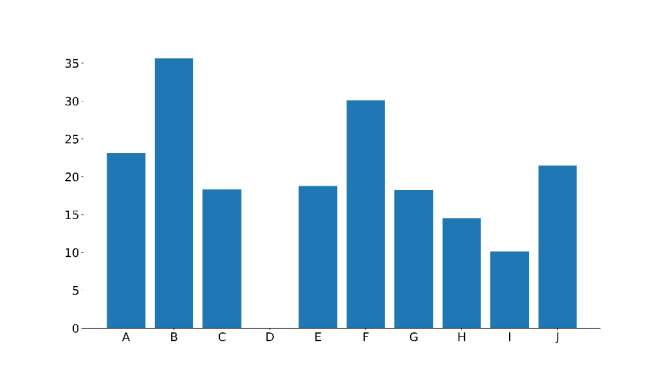


Figure13 各类节点cpu利用率和memory利用率的差值

## 3.4 失效节点分析

这一节我们对上两节的结果进行分析，我们可以发现对于某一些节点（例如：679,680,341,673）的失效次数相比其它的节点，失效次数明显更多。而只有少数节点（例如：167，478，1075）完全没有失效。为了分析原因，我们研究了失效节点和未失效节点的cpu和momery 的capacity、利用率以及在其上运行的Job类型。为了确认节点失效是由上述哪种原因造成的，我们进一步对失效Job的分布进行了分析。同失效节点分布类似，失效job分布是不均匀的，部分Job（例如：11624，2567,3012）的失效次数远大于其他的一些Job，对于失效次数比较多的Job，我们可以对该类Job的特点进行记录，分析其失效的原因。经上述分析，我们猜测节点在运行那些失效次数较多的Job时更容易失效，而在运行较不易失效的Job时可能由于节点自身的cpu和memory的限制而成为失效节点。

为了更加准确的评估节点失效的时间，从而预估节点的维护时间，我们计算了每个节点的MTBF（Mean Time Between Failure）。具体情况如图7所示。

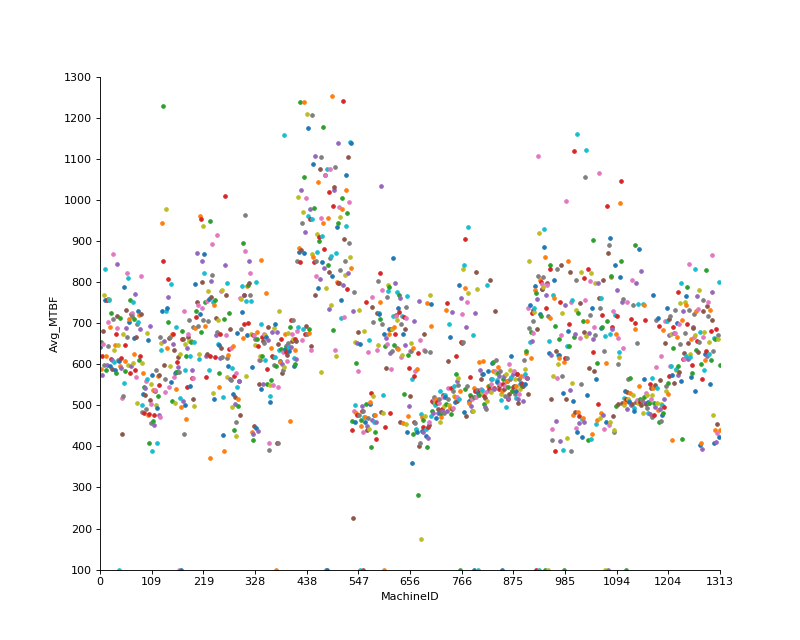


Figure 7 不同节点的MTBF

从图7我们可以发现，大多数的节点的MTBF集中在2000s~4000s之间，少数节点在4000s~7000s之间，还有极少数的节点低于2000s。

## 3.5失效节点预测和改善

根据以上对失效节点分布情况的描述以及分析，我们可以根据节点id，该节点上运行Job类型以及在哪个时间开始运行来预测节点大致的状态，若预测的状态是Failed，我们可以将Job迁移到别的节点上或者等待下一个时间点再运行，以降低失效的可能性。除此之外，我们还可以根据每个节点的MTBF以及上一个失效时刻来预测下一个失效时刻，从而制定最佳的一个维修计划，延长节点连续运行的时间。

为了降低节点失效的可能性，我们还可对失效率比较高的Job类型进行分析，确定其失效的可能原因，节点在运行该job时，避免可能的失效原因，最大限度的降低失效可能性。针对那些失效率低的节点，可将在其他节点运行的Job迁移到这些节点上面，减少其他节点的失效次数。

我们将MTBF比较小的节点称为高频率故障节点，针对这些节点，根据它们的特征，可改善其维修技术，以提高它们的MTBF。根据每个节点的MTBF，我们也可合理分配维修工作，从而更好的对节点进行维护。