百度大规模检索系统
容器配额调度策略调优

百度共享技术平台部 耿关辉

关于我

耿关辉

2009年加入百度

容量管理方向技术负责人

百度大商业体系运维技术负责人

在海量业务集群的容量建模、容量测量、资源规划、性能及成本优化方面有多年的实施经验

话题背景：**混布集群的碎片问题**

海量集群

支撑每日千亿请求，覆盖6亿用户的上百个产品。数万服务器、上百万在线service容器 的大规模虚拟化服务集群

- 物理集群的资源分配率 < 60%
- 容器内资源的利用率 > 90%

不同service容器应该如何设置尺寸及分布？

相关工作和研究

相关工作

- 索引分片问题：索引切分、压缩、缓存技术，更多关注在数据处理层面
- IDC成本优化：能耗优化、任务混布
  （Google， *Acm Transactions on Computer Systems ，2016*）
- 容器分布算法： multi-dimensional vector bin packing

常见的配额策略

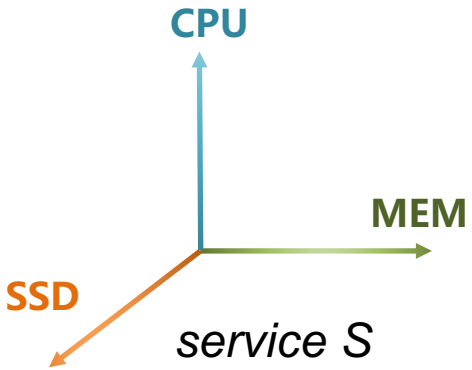
- 统一为标准尺寸（大规模系统上并不总是可达）
- 按比例设置容器资源（borg，k8s）
- 交由用户定制（EC2等公有云，Mesos，随机、或根据经验）

容器配额调度的问题定义

目标：确定分配给每个service实例的资源、实例个数、分布，使得 所用机器的资源总量最少

简化场景：

- CPU：与负载相关，总量恒定
- MEM，SSD：由service本身特性决定

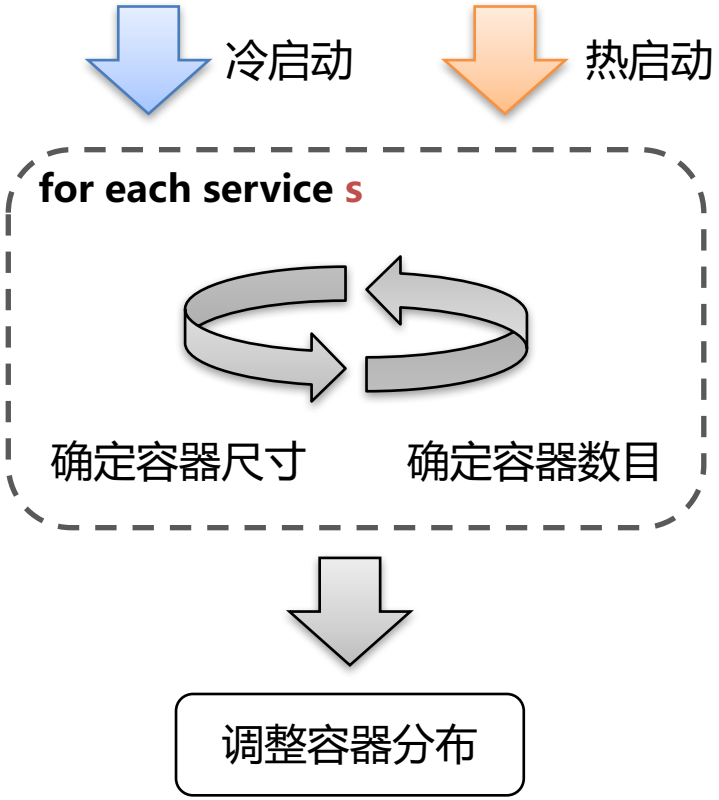


Trade-off：

✓ 增加实例数：降低单实例CPU消耗，有利 compact (小item更容易pack紧凑)

✓ 实例过多：MEM、SSD成为瓶颈，也会影响compact

容器配额调度的问题定义



实例数及约束

S	service 集合
n_s	service s 的实例个数, $s \in S$
n_s^{min}	n_s 的下限值
n_s^{max}	n_s 的上限值

$$n_s^{min} \leq n_s \leq n_s^{max}$$

X_s	service s 的实例集合

容器尺寸

$R_c(x)$	实例 x 的 cpu 需求
$R_m(x)$	实例 x 的 memory 需求
$R_s(x)$	实例 x 的 ssd 需求
$F_s(n_s)$	service s 的实例数与单实例 cpu 的映射函数

容量模型

M	机器集合
$C_c(m)$	机器 m 的 cpu 容量
$C_m(m)$	机器 m 的 memory 容量
$C_s(m)$	机器 m 的 ssd 容量

容器部署（调度）方案对配额调度效果的影响

- **Greedy 算法：First Fit Decreasing (FFD) bin packing**
  - 核心思想：根据「size」 排序容器，然后按大→小顺序放置，依次尝试机器，直到找到一台符合所有约束条件的机器执行
  - size定义：多种资源维度乘积，加权和

$$w(x) = a_c \cdot R_c(x) + a_m \cdot R_m(x) + a_s \cdot R_s(x)$$
$$w(x) = R_c(x) \cdot R_m(x) \cdot R_s(x)$$

- **Bin-Centric：考虑资源需求和机器剩余容量之间的耦合度**
  - 核心思想：每次装满一台机器，每次放置选择剩余容器中的「最佳」一个
  - 「最佳」：能置入该机器剩余容量的最大容器
- **Local search：在已有的部署方案基础上进一步优化**
  - 核心思想：尽量将较空的机器上的容器挪到其它机器上，从而减少所用机器数量

获得service的容量模型

容量压测+回归计算

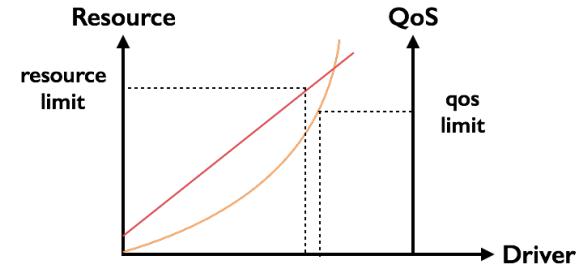
- 容量极限来自于Resource饱和或QoS撞线
- 资源利用率与Driver线性正相关
- Driver增长同时带来QoS退化

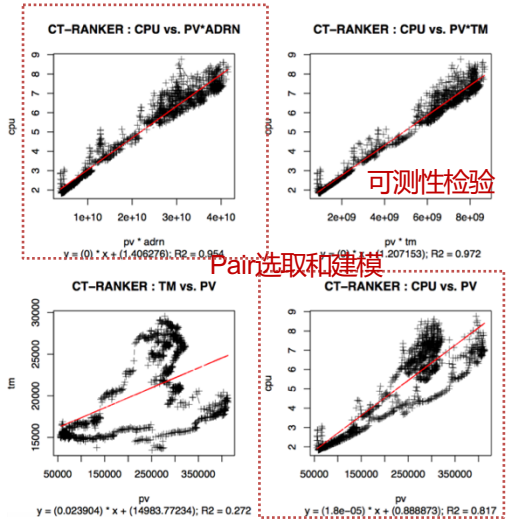
建立多组容量模型，根据Resource/QoS限制计算service容量

关键点

Key-driver与Key-resource选取

APP性能拐点





几种容器配额方案设计： MinNum

1. 对每个 service s ∈ S，取能够取到的最小实例数（最大容器尺寸）
2. 使用FFD或Bin-Centric算法进行初始部署
3. 再用local search进行优化

Algorithm MinNum

1: **for** each service $s \in S$ **do**
2: Set $n_s = n_s^{min}$
3: **end for**
4: Place instances using FFD or Bin-Centric
5: Optimize using Local Search (Algorithm 3)

优点：

简单，高效

所占用总资源最少 (不是分配最少)

局限：

适用于 冷启动

大尺寸实例，影响compact

几种容器配额方案设计： FixQuota

1. FixQuota目标是使各维资源使用均衡，从而间接地提高资源使用率
2. 假设 IDC 中所有机器的 cpu 总容量与 memory 总容量的比例为 ρ
3. 尽量让每个 service 的单实例分配的 cpu 和 memory 接近 ρ

Algorithm FixQuota

1: **for** each service $s \in S$ **do**

2: Set $n_s = \arg\min_{n_s^{min} \le k \le n_s^{max}} \left| F_s(k)/R_m(s) - \rho \right|$

3: **end for**

4: Place instances using FFD or Bin-Centric

5: Optimize using Local Search (Algorithm 3)

优点：

部署问题退化为一维

有利于compact

局限：

适用于 冷启动

机型异构及容器尺寸大小

分布显著影响性能

几种容器配额方案设计： SearchOnInit

1. 机器按照剩余资源排序，从最空闲开始遍历处理
2. 依次尝试将机器上的每个实例删除： ①压缩冗余 ②按比例调大容器的CPU配额
3. 用local search进行优化

Algorithm SearchOnInit

1: **while** runing time is not up **do**
2: Sort machines in increasing order according to remaining capacity, denote by $m_1, \ldots, m_n$
3: **for** each $1 \leq i \leq n$ **do**
4: Try to remove the instances on machine m_i
5: **end for**
6: Local Search (Algorithm 3)
7: **end while**

优点：

可用于 热启动

易于实施

局限：

受初始分布影响

配额调度实验方案

参考指标：

- *CORR(X, Y)：*多个资源维度之间相关性 $CORR(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var[X]Var[Y]}}$
- *Dominant Ratio (DR)：判别某种资源维度为主要约束*
- *对比ConsOnInit：*直接用 local search 进行 compact，反映当前状态的优劣程度
- *理论极限*： 总分配资源量的下限值，反映各个算法与最优解的距离

Product1：idc1~5； Product2：idc6~7

	P1-idc1	P1-idc2	P1-idc3	P1-idc4	P1-idc5	P2-idc6	P2-idc7
machines	1850	1221	2024	1547	2014	2522	4520
services	78	64	81	60	123	4933	10853
containers	2416	2763	3062	2063	2938	12997	25239

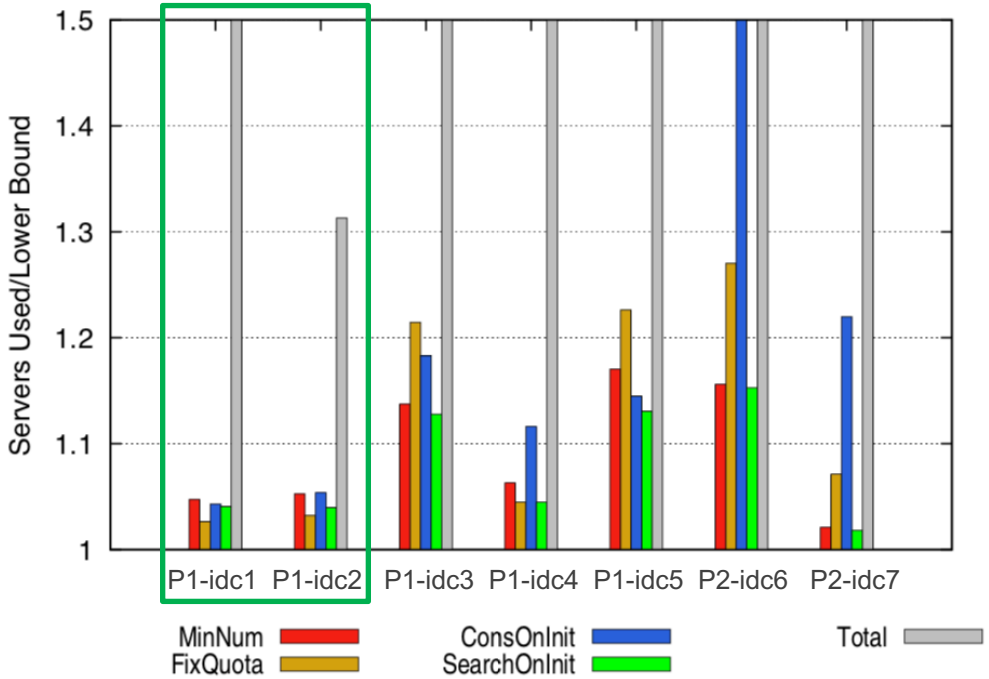
实验结果

各方案在各个 IDC 的表现：

- 纵轴表示各个方案所用的机器数与理论极限的比值

  数值越小则越优

- Total 表示当前机房总机器数和理论极限的比值



配额调度实验结果分析：IDC 1

	MinNum	FixQuota	ConsOnInit	SearchOnInit
Instance Number	1748	3627	3073	2339
$CORR(cpu, mem)$	0.4	0.90	0.66	0.68
$DR(cpu : mem)$	0.88	0.94	0.78	0.86

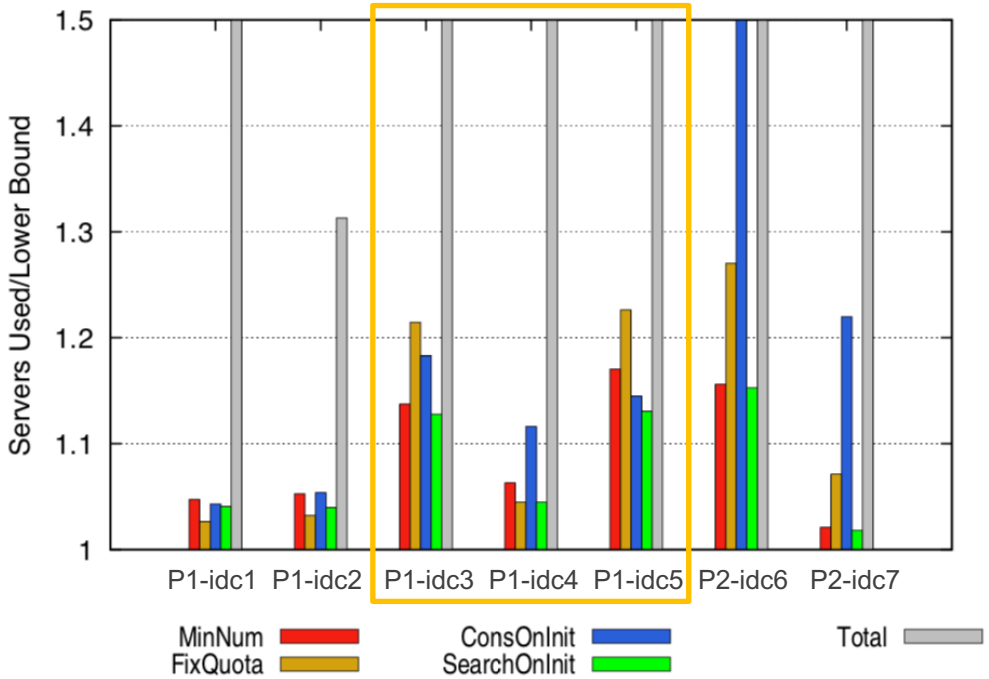
分析：

- 问题退化为一维资源约束
    1. FixQuota 有很高的资源相关系数
    2. MinNum和SearchOnInit的CPU主导约束明显
- 无机型异构

实验结果

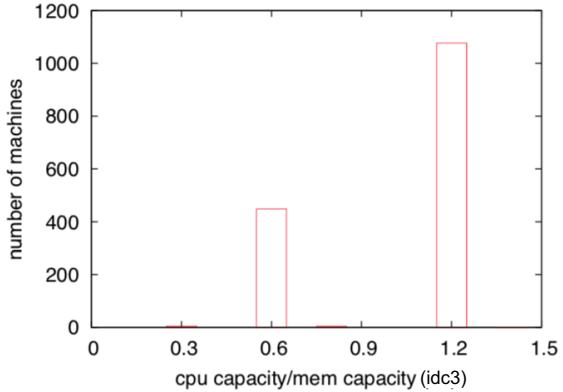
各方案在各个 IDC 的表现：

- 纵轴表示各个方案所用的机器数与理论极限的比值
- Total 表示当前机房总机器数和理论极限的比值

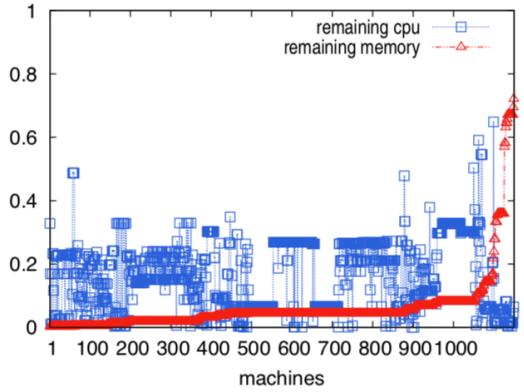


配额调度实验结果分析：IDC 3

	MinNum	FixQuota	ConsOnInit	SearchOnInit
Instance Number	1765	2083	2063	1853
$CORR(cpu, mem)$	0.72	0.88	0.68	0.76
$DR(cpu : mem)$	0.77	0.83	0.73	0.76



机器CPU与MEM的比值分布



FixQuota方案下的资源剩余

分析：

1. 机器异构较严重
   - FixQuota平均效果最差
2. 容器原始CPU配额偏大
   - 影响compact效果

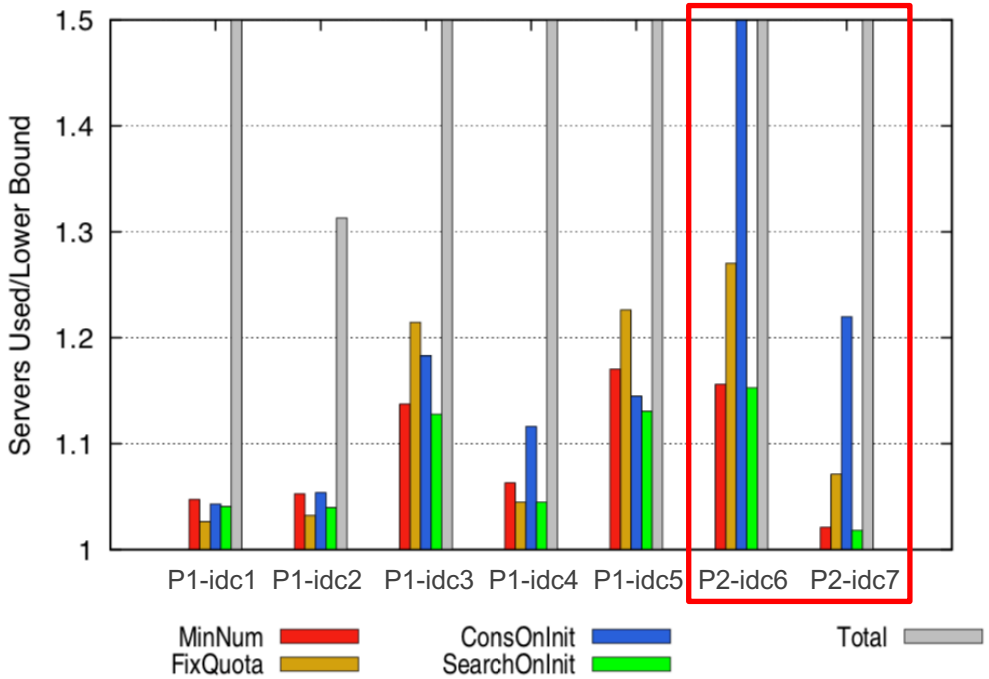
建议：

- 缩减单容器CPU配额
- 消除机器异构

实验结果

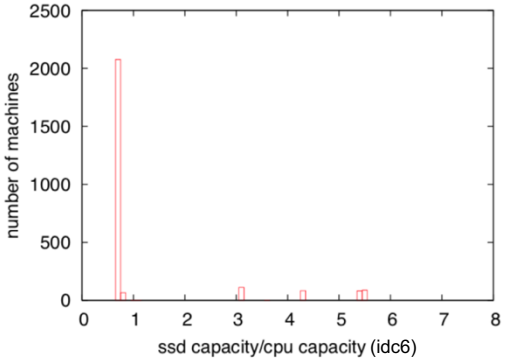
各方案在各个 IDC 的表现：

- 纵轴表示各个方案所用的机器数与理论极限的比值
- Total 表示当前机房总机器数和理论极限的比值

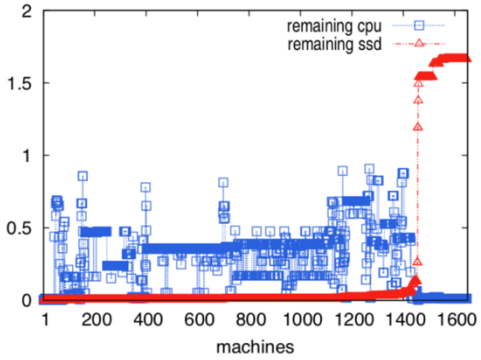


配额调度实验结果分析：IDC 6

	MinNum	FixQuota	ConsOnInit	SearchOnInit
Instance Number	8387	11234	12997	8584
$CORR(cpu, mem)$	0.36	0.84	0.74	0.42
$CORR(cpu, ssd)$	0.32	0.78	0.68	0.39
$CORR(mem, ssd)$	0.60	0.69	0.62	0.59
$DR(cpu : mem)$	0.2	0.26	0.04	0.22
$DR(cpu : ssd)$	0.24	0.22	0.15	0.25
$DR(mem : ssd)$	0.14	0.22	0.15	0.15



机器SSD与CPU的比值分布



FixQuota方案下的资源剩余

分析：

1. SSD为主要资源约束
   - Service自身特性决定
   - 产生容器数少的方案更优
2. 机器异构比较严重
   - 绝大多数机器有30%的CPU浪费

建议：

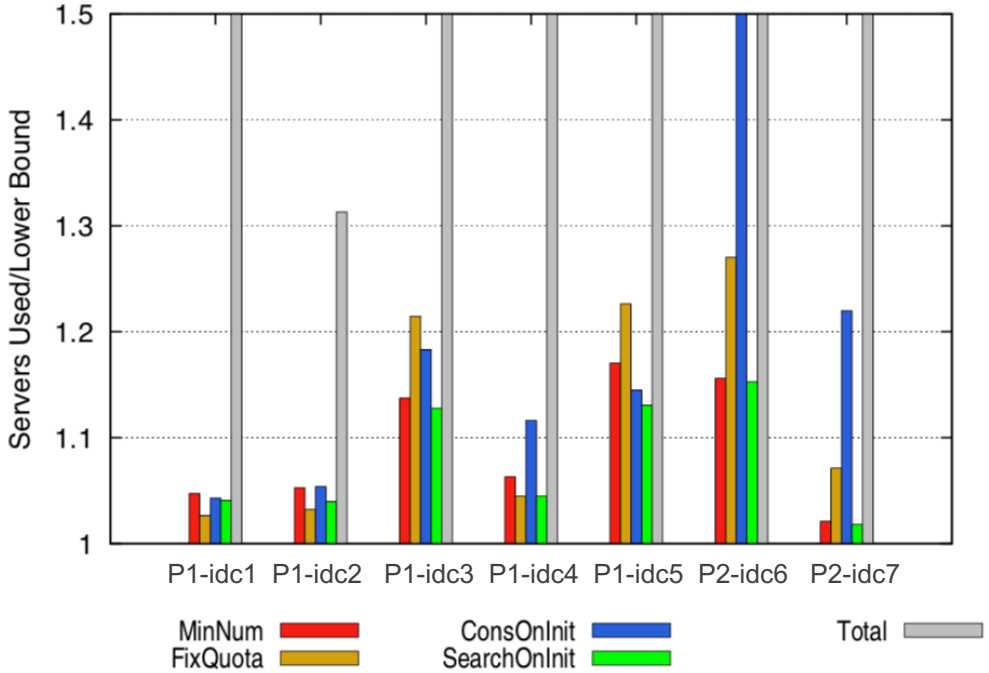
- 减少容器个数，提高单容器CPU配额
- 消除机器异构
- 调整数据分片，减少单容器SSD需求

实验结果

各方案在各个 IDC 的表现：

- 纵轴表示各个方案所用的机器数与理论极限的比值
- Total 表示当前机房总机器数和理论极限的比值

SearchOnInit 方案平均更优



实施方案及效果

- 仿真器，作方案选择
- 实际落地方案
  - Product 1：FixQuota & SearchOnInit
  - Product 2：SearchOnInit
- 效果：相比实施前节约 数千台 服务器

方案总结

方案	特点	注意事项
MinNum	部署新 service 简单高效 容器数最少，SSD和MEMORY资源消耗最少	可能产生过多大容器，影响compact
FixQuota	CPU 和 MEMORY 相关系数高，利于compact	CPU不是资源瓶颈时可能产生过多容器，而影响性能 IDC机器异构会显著影响性能
SearchOnInit	类似 MinNum，容器数少 不会产生过多大容器 热启动，更容易实施 只考虑减实例而没有增加实例过程	初始布局或者容器较大可能会对算法性能产生一定影响

谢谢！