一种数据流处理环境下的节点副本放置方法

丁维龙* 韩燕波
(北方工业大学云计算研究中心 北京 100144)

摘 要：物联网环境下的许多应用表现为传感数据的连续流式处理，且系统往往通过节点的副本技术保障可用性。但是，运行时副本的备份和放置存在内存和带宽等资源开销，产生处理的延迟。该文给出一种方法，根据运行时的资源消耗以贪心方式放置节点的副本，折中了系统的可用性和开销。实际系统的仿真实验表明，在相同的条件下，该方法相比传统的随机放置，能为系统提供更稳定的可用性。

关键词：物联网；数据流；可用性保障；副本放置；贪心算法
中图分类号： TP301 **文献标识码**：A **文章编号**：1009-5896(2014)07-1755-07
DOI: 10.3724/SP.J.1146.2013.01051

A Replica Placement Method during Data Stream Processing

Ding Wei-long Han Yan-bo
(*Research Center for Cloud Computing, North China University of Technology, Beijing* 100144*, China*)

Abstract: Many applications of Internet of Things (IoT) are performed by the continuous stream processing of the senor data and nodes' replicas are required to guarantee system availability. However, the replicas' backup and placement often bring the processing delay at run-time due to the consumption of resources such as memory and bandwidth. In this paper, a method is proposed as greedy fashion by the resources cost to place nodes' replicas, which could tradeoff between the availability and overheads of the system. Moreover, in a practical system, the extensive experiments show that the availability of the proposed method can be provided in a more stable manner than the traditional random placement under the same conditions.

Key words: Internet of Things (IoT); Data stream; Availability guarantee; Replica placement; Greedy algorithm

1 引言

物联网环境下的许多应用呈现数据的多源并发和汇聚处理的特征，为此近年来针对无边界、实时和连续及大规模传感数据的数据流处理(data stream processing)的相关研究发展迅速。本质上，数据流(data stream)是传感数据(sensor data)以追加递增(append-only)方式形成的无限集合[1–3]，相应地，低延迟、高吞吐量成为数据流处理的目标[4]。实时数据流处理在无线传感网络、金融、医疗、交通和军事[2,5]等关键领域，实现了监控与分析应用。为了提高性能，分布式数据流处理系统(Data Stream Processing System, DSPS)[6,7]，需要增加处理节点的数量[2,5]，但也同时增加了因为某一处理节点的失效导致系统整体受到影响的风险[5,8]。例如，在并发量大或数据高速环境下，系统接收快速到达的数据，一旦处理节点出现故障，一方面影响甚至中断下游的数据接收和处理，另一方面导致未能传递的数据迅速积累，影响上游节点的处理过程，甚至产生上游的连锁反应，降低系统的可用性。而事实上，云计算环境下的大规模系统，总是存在因为个别机器故障导致的节点失效。

可用性保障成为数据流处理系统的关键问题和内在需求[4,5]，而运行时节点的副本技术是其中的研究热点，保证在节点故障时无人工干预快速地恢复数据处理。节点的被动副本，是被冗余放置的节点状态的定期备份，相比完全冗余和在线的主动副本，由于可以调整备份间隔、放置位置和副本选择等策略，内存和带宽等开销可灵活，实现也更加复杂。基于被动副本的故障恢复，实际是通过延迟和资源开销[9]换取系统可用性，故存在着广泛的适用性。本文如无特殊说明，副本均指代被动副本。当前的相关研究，大多从副本备份的角度权衡可用性，然而给定系统中不同的副本放置策略，由于开销和延迟存在差异，同样影响着系统的可用性。

副本的随机放置是最常见的策略，广泛应用于当前的分布式数据流处理系统，如 Synergy[4,10]和

2013-07-17 收到，2014-01-26 改回
北京市属高等学校创新团队建设与教师职业发展计划 (IDHT20130502)，北京市教育委员会科技计划重点项目(KZ201310009009)和北京市教育委员会科技计划面上项目(KM201310009003)资助课题
*通信作者：丁维龙 dingweilong@ncut.edu.cn

CLASP[9,11,12]。文献[4]针对 Synergy 系统，给出根据副本放置改善系统可用性的开创性工作：通过定性和定量证明了随机放置副本的系统，其可靠性随节点的副本数增加而降低，也随机器数量增加而降低。这种反直观的结论源于如下事实：副本在随机放置时，增加机器或增加副本度，使得涉及副本放置的机器数量增多，从而因为某些机器的故障使得同一节点的所有副本不可用的概率增大，故系统宕机的概率增大。副本放置于指定位置是另一类常用策略，广泛应用于节点副本的上游备份，如 Borealis 系统[13]和 Sweeping Checkpointing 协议[9]。这种将副本放置于节点直接上游的方式，其优势在于消除了故障时选择副本的延迟，但却限制了节点副本度；同时，由于指定机器的宕机一定引发系统的不可用，故并不适合可用性需求较高的系统。

系统难于获取最优的副本放置策略，是因为副本放置与系统可用性存在固有矛盾：保证单个节点的可用倾向于其副本分布放置，而保障系统的可靠倾向于不同节点的副本集中放置。为此，本文给出一种方法，贪心地选择副本放置的机器，在给定的资源下折中副本放置阶段的延迟与带宽/内存开销，并理论上证明了对系统可用性的改善。同时，实际系统的仿真实验表明，该方法相比随机放置，在副本数、节点数、机器数和故障机器数相同的条件下，能为系统提供更稳定的可用性。

本文组织如下：第 2 节分析了研究背景，归纳了问题并阐述了优化开销与延迟的副本放置算法；第 3 节设计实验定量和定性评价本文工作；第 4 节为结束语。

2 优化开销与延迟的副本放置

2.1 问题分析

本文的研究背景来自某大型城市网格交通项目。该市在数千个关键路口部署的交通摄像头，每秒拍摄和识别一个机动车的车牌，并将拍摄的照片、识别的车牌和时间地点等 22 项属性打包为指定结构的数据项(tuple)，实时发送至云端的系统用于实时甄别违章车辆。在本文开发的系统中，数据以流的形式，被接收、校验和过滤后分发至在线的十余种交通业务计算，如假牌、黑名单、套牌和超速分析等，业务中最长的响应时间不能超过 3 s，是典型的实时计算需求。系统的节点或实现计算，如套牌计算节点、假牌计算节点，或提供基础功能，如通信节点、消息队列节点和存档节点。系统运行时定期备份节点的中间状态作为副本，避免丢失节点故障时的中间结果。但是，系统仍然存在不可用的可能：因为某些机器的宕机，某故障节点的所有副本均不可用于恢复。

系统中的某些节点能够容忍处理失效引起的输入数据的丢失或不准确（与业务计算相关），但整体过程的正确执行总倾向于所有节点功能正常可用。为了描述这类系统的特征，类比数据操作符(operator)的严格性[14]，本文给出严格系统的定义。本文的基本思想，就是通过合适副本放置策略，提高严格系统可用性。

定义 1 严格系统(strict system)。可用性依赖于所有组成节点的数据流处理系统，称为严格系统。

严格系统可用，当且仅当系统中所有节点可用，如下两个因素是必要的考虑。(1)机器的 CPU、内存和带宽资源。当资源给定时，上述资源的总量是确定的常量，而各个机器剩余的 CPU、内存和带宽资源是时变的，并可通过监控实时获取。(2)机器间传输的延迟。严格系统的可用意味着所有节点可用，为此副本倾向于在有限的机器放置[4]。

可见，这类问题一方面为保障系统的可用性，需优化放置所使用的机器；另一方面为保证处理的实时性，需优化资源开销，减少数据传输延迟。

2.2 问题归纳

综上分析，本文的优化目标是，最大化系统可用性和最小化放置副本的数据延迟。同时，本文存在如下几个假设。

(1)机器的故障导致节点不可用。事实上，除了机器的故障，程序异常等软件因素也可导致节点不可用，但本文关注副本在集群中的放置，故主要关注机器故障导致的不可用。

(2)为了叙述的方便，节点(node)默认是指节点的主副本，副本默认是指节点的被动副本，副本度(replica degree)是 1 个节点的副本数量加 1。

(3)节点的部署在系统启动时已经确定，其副本放置于内存；放置的基本原则是，节点与其副本之间，以及同一节点的副本之间，部署在不同的机器。

于是，本文的问题可归纳为如下的多目标规划。

$$\min \mathrm{Nu},\quad \mathrm{Nu}=\#\left\{v_k \left| \exists p_{i\_j}, \mathbf{Mp}\left(v_k, p_{i\_j}\right)=1\right.\right\} \tag{1}$$

$$\min \sum_{i=0}^{n-1}\sum_{j=1}^{d_i-1} \mathrm{ls}_{i\_j}\cdot l_{v_k,v_{k'}},\quad \mathbf{Mp}\left(v_k, p_{i\_0}\right)=1,\ \mathbf{Mp}\left(v_{k'}, p_{i\_j}\right)=1,\ j\in[1,\cdots,d_i-1] \tag{2}$$

s.t.

$$\mathbf{Mp}\left(v_{k'}, p_{i\_j}\right)=0,\ \text{当}\mathbf{Mp}\left(v_k, p_{i\_j}\right)=1,\ \forall p_{i\_j}, k\neq k' \tag{3}$$

$$\mathbf{Mp}(v_k, p_{i\_j'})=0,\ \text{当}\mathbf{Mp}(v_k, p_{i\_j})=1, \forall v_k, j\neq j' \tag{4}$$

$$\sum_{i=0}^{n-1}\mathrm{lp}_i+\sum_{i'=0,i'\neq i}^{n-1}\mathrm{ls}_{i'\_j'}<\mathrm{Lv}_k, \forall v_k, \mathbf{Mp}\left(v_k, p_{i\_0}\right)=1, \mathbf{Mp}\left(v_k, p_{i'\_j'}\right)=1 \tag{5}$$

$$\sum_{i=0}^{n-1} m_i + \sum_{i'=0,i'\neq i}^{n-1} \mathrm{ls}_{i'\_j'} < \mathrm{Mv}_k, \forall v_k, \mathbf{Mp}\left(v_k, p_{i\_0}\right)=1,$$

$$\mathbf{Mp}\left(v_k, p_{i'\_j'}\right) = 1 \tag{6}$$

默认$k,k' \in [0,\cdots,N-1]$, $i,i' \in [0,\cdots,n-1]$,

$$j,j' \in [0,\cdots,d_i-1] \tag{7}$$

表 1 列出了其中涉及的符号及语义。其中，机器数量 N、节点主副本数量 n、节点副本度 d_i是构造时常量；机器的总带宽 Lv_k和总内存 Mv_k，机器间的延迟 $l_{x,y}$是运行时常量；已部署节点和放置副本的机器数量 Nu、节点的 CPU 使用率 tp_i、节点之间接收和发送的数据量 $\mathrm{lp}_{i_\mathrm{in}}$和 $\mathrm{lp}_{i\_\mathrm{out}}$、放置副本的数据量 ls_{i_j}、副本放置矩阵 **Mp** 是运行时变量。其中，**Mp** 初始化时是零矩阵，标识了节点在机器的部署，随着副本放置的变化而动态调整。

表1 算法使用的符号及语义

符号	语义
N	机器的数量
n	节点(主副本)的数量
v_k	机器 k, $K\in[0,\cdots,N-1]$
Nu	使用的机器的数量，$\mathrm{Nu}\le N$
d_i	节点 i 的副本度(同一节点的副本数，包括主副本, $d_i\ge 1$)
p_{i_j}	节点 i 的副本 j, $i\in[0,\cdots,n-1]$, $j\in[0,\cdots,d_i-1]$，当 $j=0$ 时是主副本
Lv_k	机器 k 的总带宽
Mv_k	机器 k 的总内存
m_i	节点 i(的主副本 p_{i_0})占用的内存
tp_i	节点主副本 p_{i_0} 的 CPU 使用率
$\mathrm{lp}_{i\_\mathrm{in}}j$	主副本 p_{i_0} 从主副本 p_{j_0} 接收数据的数据量
$\mathrm{lp}_{i\_\mathrm{out}}j$	主副本 p_{i_0} 向主副本 p_{j_0} 发送的数据量
$\mathrm{lp}_{i\_\mathrm{in}}$	主副本 p_{i_0}接收来自其它主副本数据的数据量。 $\mathrm{lp}_{i\_\mathrm{in}} = \sum_{j=0,j\neq i}^{n-1} \mathrm{lp}_{i\_\mathrm{in}}j$
$\mathrm{lp}_{i\_\mathrm{out}}$	主副本 p_{i_0}向其它主副本发送数据的数据量。 $\mathrm{lp}_{i\_\mathrm{out}} = \sum_{j=0,j\neq i}^{n-1} \mathrm{lp}_{i\_\mathrm{out}}j$
ls_{i_j}	副本 p_{i_j}占用的带宽
$l_{x,y}$	机器 v_x和 v_y之间的延迟
lp_i	节点 i(主副本 p_{i_0})占用的带宽。 $\mathrm{lp}_i = \mathrm{lp}_{i\_\mathrm{in}} + \mathrm{lp}_{i\_\mathrm{out}} + \sum_{j=1}^{di-1} \mathrm{ls}_{i\_j}$
Mp	$N\times(n\times d_i)$的副本放置矩阵，其中元素 $\mathbf{Mp}(v_k,p_{i\_j})$ 取值为 0 或 1; 1 表示副本 p_{i_j} 放置于机器 v_k。

该规划有两个目标。如式(1)所示，最小化放置所使用机器的数量。该目标等价于最大化严格系统的可用性[4]。如式(2)所示，最小化副本放置导致的数据在网络中的传输延迟。这等价于最小化副本放置带来的性能的降低。同时，规划存在 5 组约束。如式(3)所示，任意副本只能放置于一个机器。如式(4)所示，任意机器只能放置节点的主副本或一个备份副本。如式(5)所示，放置于同一机器的所有副本所占用的带宽，不能超过该机器的总带宽。如式(6)所示，放置于同一机器的所有副本所占用的内存，不能超过该机器的总内存。事实上，由于副本是节点的备份，在计算中可认为，$\mathrm{ls}_{i\_j}\approx m_i, i\in[0,\cdots,n-1]$, $j\in[1,\cdots,d_i-1]$。如式(7)所示，相关下标默认的边界条件。

图 1 针对节点 p_{1_0}，给出了节点部署和副本放置的示例并标注了数据传输。其中，实线标注了节点间的数据传输；虚线标注了节点和副本间的数据传输。注意，副本状态同步的带宽开销，相比副本传输的数据量要小得多，可忽略不计，故图中节点与副本间的通信记作单向箭头。

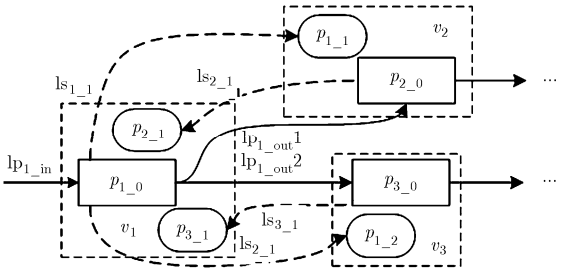


图 1 副本放置的示例

2.3 算法实现

多目标规划属于 NP 完全的问题难度[4]，无法在多项式时间内获得最优解。同时，在数据流环境下，求解最优往往意味着更大的延迟，导致与实时性需求的矛盾。为了实现上一小节归纳的规划，折中实时性与系统可用性和资源开销，这里给出如表 2 的 MRP(Multi-objective Replica Placement)算法选择放置副本的机器。该算法的基本思想是，排序资源满足条件的机器，在不违背放置原则的条件下，优先选择排名高的机器放置副本。可见，这是典型的贪心策略。

算法的输入有构造时常量(机器数量 N、节点主副本数量 n、节点副本度 d_i)和通过监控定时更新的运行时常量(机器的总带宽 Lv_k和总内存 Mv_k，机器间的延迟 $l_{x,y}$)。算法的输出是全局变量副本放置矩阵

表2 MRP算法的实现框架

输入：N, n, d_i, Lv_k, Mv_k和 $l_{x,y}$

输出：**Mp**

```
1   for(i=0; i<n–1; i++)
2     v_k = physical node hosts p_{i_0};
3     for (j=1; j< d_i–1; j++)
4       udset = ascendingSort(physical nodes allocating replicas);
5       unset = ascendingSort(physical nodes unallocating replicas);
6       while (exist v_k' in udset which meets constraints)
7         Mp(v_k', p_{i_j})=1;
8         return Mp;
9       end while
10      while (exist v_k'' in unset)
11        Mp(v_k'', p_{i_j})=1;
12        return Mp;
13      end while
14    end for
15  end for
16  return ERROR_Resource_Insufficient;
```

Mp。为了衡量放置所带来的处理延迟，本文给出了放置延迟指标：

$$\sum_{j=1}^{d_i-1}\mathrm{ls}_{i\_j}\cdot l_{v_k,v_{k'}}+\sum_{i'=0}^{n-1}\mathrm{ls}_{i'\_j'}\cdot l_{v_k,v_{k''}}$$

该指标的直观含义是副本备份和放置导致的延迟数据量，包括本机向远端放置副本和远端向本机放置副本导致的延迟数据量。

表 2 第 4~5 行通过 ascendingSort(·) 函数按照放置延迟指标递增排序，分别返回已放置过副本和未放置过副本的机器。这实际是对规划目标的式(2)的贪心求解。其中，$\mathbf{Mp}(v_k,\ p_{i\_0})=1$；$\mathbf{Mp}(v_{k'},\ p_{i\_j})=1,\ j\in[1,\cdots,d_i-1]$；$\mathbf{Mp}(v_{k''},\ p_{i'\_0})=0$，$\mathbf{Mp}(v_k,\ p_{i'\_j'})=1$，$i'\neq i$，$j'\in[1,\cdots,d_i-1]$，反映了规划条件的式(3)和式(4)。算法第 6~9 行，对副本 p_{i_j}, $i\neq0$，在符合资源约束条件下，首选已经放置过副本的放置延迟指标最小的机器，即复用机器降低 Nu。这实际是在满足规划条件的式(5)和(6)时，对规划目标的式(1)的逼近。表 2 第 10~13 行，当不存在符合资源约束的已放置过副本的机器时，选择未部署过副本的放置延迟指标最小的机器。表 2 第 16 行，如果不存在符合约束的机器，则返回资源不足的错误提示，此时需要增加机器。

值得指出的是，运行时变量在本文中是通过分析 Linux 系统的/proc 目录和调用 htop, dstat 和 Iperf 等工具实时获取的。例如，节点的 CPU、带宽(输入、输出和传输副本)、数据传输量，是通过上述监控工具得到的最近 1 s 的平均值；机器间延迟是 ping 命令结果在最近 1 min 的均值。

算法的时间复杂度为 $O\left(N\times n\times\max\limits_{0\le i\le n-1}d_i\right)$，与机器数量、节点数量和节点副本度有关。上述参数均为构造时的常量，所以算法的时间复杂度也是常量。本算法借鉴了文献[4]的研究思路，但目标和实现与原始工作存在显著的不同。(1)原始工作的算法不仅放置副本也部署节点，而本文算法只放置副本。事实上，确定节点部署位置后启动系统，在实践中更实际和可控。(2)原始工作假设所有节点的副本度相同，且资源开销是稳定可预测的构造时常量；而本文算法每个节点有各自的副本度，即 $d_i(i=1,\cdots,n)$ 不必相同，且资源开销是是监控获取的实时值。总之，由于副本是节点运行时的备份，通过监控资源的方法是更适合实际场景中副本的按需放置。

本文方法中，系统可根据算法输出的矩阵 **Mp**，指导完成副本在机器间传输和放置，在此不再详述。

3 实验与评价

实验环境采用 DELL PowerEdge 机架服务器，其单台配置为 AMD Opteron 16 核 CPU, 32 GB DDR2 RAM, 500 GB SATA 硬盘，安装 CentOS 5.5 x86 64 bit 操作系统。实验在本团队开发的 VINCA-CCS 系统上完成，该系统目前实际应用在 2.1 节提及的网格交通项目。本节配置 VINCA-CCS 实现了如图 2 所示的严格系统。

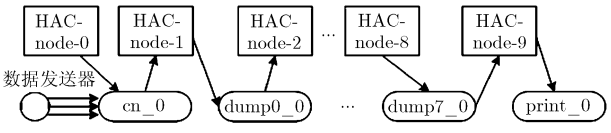


图2 实验的系统配置

针对该配置的系统相关说明如下。

(1)节点数 n=10: 1 个通信节点 cn_0, 8 个中间节点 (dump0_0~dump7_0), 1 个打印节点 print_0。10 个节点串行连接，实现在文件中输出数据发送器所发送的数据。通信节点 cn_0 是系统中第 1 个节点，向下游发送结构完整的数据项；dump 节点是用于调试的特殊节点，接收上游数据后不做额外处理直接向下游传递；打印节点 print_0 将接收的数据追加于指定的文件中。系统中节点的串行配置，使得数据最终的成功打印依赖于所有节点的

可用，故是严格系统。

(2)节点的副本度 d_i=3, $i=[0,\cdots,9]$，即每个节点默认存在两个副本。可供使用的机器数量 N=20。为了清晰示意，图 2 仅列出了节点的主副本。

(3)被动备份、副本放置和基于副本的节点故障恢复是通过 HAC-node[15−17]完成的。HAC-node 是本团队之前开发的一种端对端的连接件，提供了数据传输、缓存等基础服务和编程原语。10 个 HAC-node 分别配置在 10 个节点的直接上游，实现下游节点的副本备份和放置。注意：本节实验只关注副本放置对系统可用性的影响，故将 HAC-node 部署在不计入 N 的其它机器，并保证实验过程中始终可用。

(4)通过停止给定机器的网卡，仿真机器的故障，设置不可用机器的默认比例为 15%，并将故障机器的数量记作 H_N。即 N=20 时，随机选取 3 个机器仿真故障。这个比例取值参考了 Synergy 系统实际运行时不可用机器的统计值[4]。

(5)实验中系统可用的评价，可通过监控 print_0 节点的输出文件实现。系统正常运行时，输出文件的大小由于数据持续追加而连续增长；数据处理中断时，文件大小不再变化。实验以 0.1 s 为间隔(保证小于故障恢复的延迟)监控输出文件，判断系统的可用状态。为定量衡量系统的可用性，本节定义了可用比这个指标。可用比是在重复进行的实验中，系统正常运行的验次数占总实验次数的百分比。该指标的直观含义是，当给定数量的机器不可用时，系统能正常运行的概率。易于证明，可用比高是系统可靠的一个必要条件。

(6)HAC-node 同时实现了另外两种副本放置，用于对比本文放置方法的效果。(1)随机放置，将副本随机放置于可用的机器；(2)整群放置，即任何一个放置了副本的机器，将会放置所有节点的一个副本。即 $\forall v_m, v_m \in \{v_k \mid \exists p_{i\_j} \text{ on } v_k, j>0\}$, $\forall i' \in [0,\cdots,n-1]$, $\exists p_{i'\_j'}$ on $v_m, j'>0$，符号同 2.2 节的定义。整群放置保证了任何一个放置副本的机器均可以完成所有节点的恢复，只要存在一个这样的机器系统便可保证可用，是最大化系统可用的最优放置，但由于资源的限制在实践中并不实用[4]，在本节中实验作为最优解进行比较。

(7)本团队之前开发的数据发送器(data generator)[15−17]可以仿真发送数据流。在这个工具中，连接数量、每个连接的发送速度和数值分布，都是可以配置的。实验中，默认设置并发量为 1000、发送速度为每个连接每秒发送 1 个数据项。

接下来设计一组实验，仿真机器的故障，定量对比不同的副本放置对系统的可用性的影响。

实验 1　在初始配置基础上，配置节点副本度 d_i 从 1 调整至 5, $i \in [0,\cdots,9]$。每个 d_i 的配置下重复实验 100 次，统计 3 种副本放置的可用比。

实验 2　在初始配置基础上，调整节点 print_0 前连接的 dump 节点数量，使得系统中节点数量 n 从 5 增至 15。dump 节点在彼此相异的机器上部署，副本度均配置为 2。每个 n 的配置下重复实验 100 次，统计 3 种副本放置的可用比。

实验 3　在初始配置基础上，逐步增加实验所使用的机器，使得 N 从 10 调整至 20。每个 N 的配置下重复实验 100 次，统计 3 种副本放置的可用比。

实验 4　在初始配置基础上，设置故障机器数量 H_N，分别为 3, 5 和 8。每个 H_N 的配置下重复实验 100 次，统计 3 种副本放置的可用比。

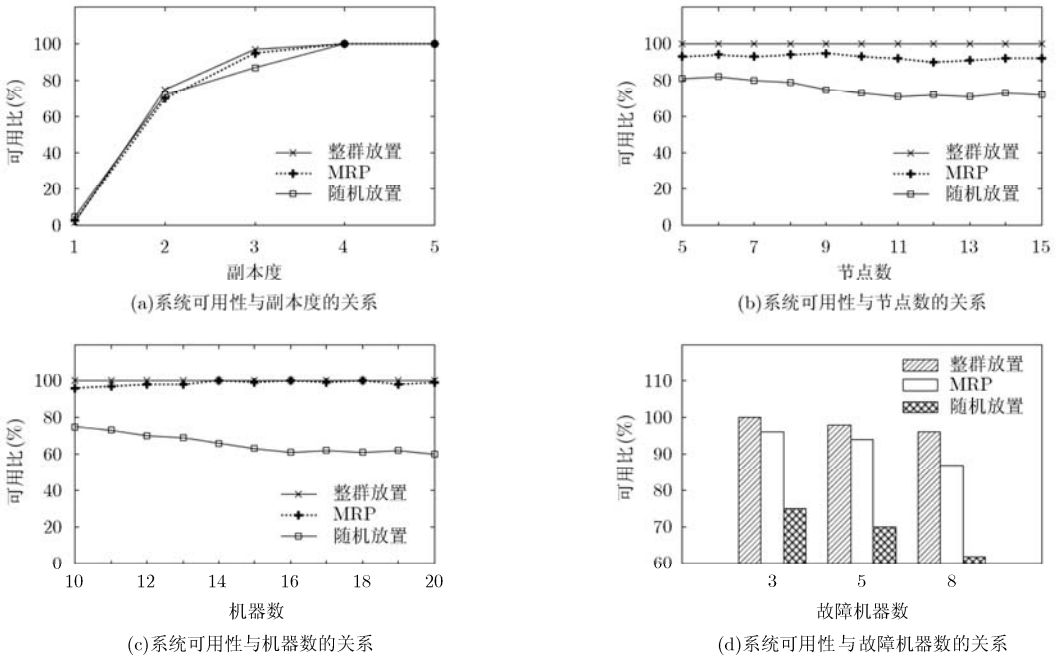
上述 4 个实验的结果如图 3(a)~图 3(d)所示，分析实验的结果，可以得到以下几个结论。

(1) 整群放置是最优方案。只要 $d_i \geq H_N$，该放置在上述 4 个实验的可用比一定是 100%。

(2)当副本度 d_i 增加时，MRP 放置和随机放置下的可用比均快速增加至 100%；在同一副本度下，MRP 相比较优，如图 3(a)。这是因为在随机放置下，副本度增加意味着副本在更多机器冗余的放置，某一节点及其副本刚好同时放置在给定故障的机器(H_N=3)上的概率增大，故系统的可用比减小。而在 MRP 放置下，在资源允许时不同节点的副本尽可能集中放置，在给定数量的机器故障时，系统不可用的概率基本不变，故可用比保持稳定。

(3)当主副本数量 n 增加时，可用比在 MRP 放置中是稳定的，而随机放置中稳中略降，如图 3(b)。在给定 d_i(=2) < H_N(=3)和 N(=20)的配置下，节点数增加使严格系统可用的条件更加苛刻。随机放置最大化副本在机器的均匀程度，增大 n 也增大了给定的故障机器放置了某个节点及其所有副本的概率。同理，MRP 放置在资源允许时不同节点的副本尽可能集中放置，在给定数量的机器故障时，可用比保持稳定。

(4)当机器数量 N 增加时，可用比在 MRP 放置下最终稳定在 100%，而在随机放置中稳中略降，如图 3(c)。MRP 放置首选已放置过副本的机器，故增加机器对放置没有影响，故可用比基本不变；而由于随机放置的随机性，使得系统涉及放置副本的机器的数量增多，因给定机器故障使同一节点所有副本不可用的概率增大，故可用比下降。



(a)系统可用性与副本度的关系 (b)系统可用性与节点数的关系

(c)系统可用性与机器数的关系 (d)系统可用性与故障机器数的关系

图 3 3 种副本放置算法下的可用比

(5)当故障机器的数量增加 H_N时，3 种放置下可用比均出现不同程度的下降；在同一条件下，MRP 放置相比随机放置较优，如图 3(d)。3 种放置因副本集中程度不同，可用比下降的幅度不同。相比最优的整群放置，MRP 放置虽然衰减较快，但是在 H_N =8 时可用比仍在 90%左右；而随机放置在同样条件下的可用比只有 60%左右。

综上，在相同条件下 MRP 放置相比随机放置能为系统保障更稳定的可用性；同时，由于 MRP 是考虑延迟以贪心的方式选择放置机器，故系统开销对处理延迟的影响一定是近似最优的，折中了副本放置影响的系统可用性和系统开销。

4 结束语

实时数据处理系统的副本放置策略，需要权衡系统开销与处理延迟，影响着系统的可用性。本文给出的一种贪心的副本放置方法，在给定的物理资源下折中了副本放置影响的系统可用性和系统开销。实际系统的仿真实验表明，该方法相比传统副本的随机放置，可保障系统实现更稳定的可用性。

参 考 文 献

[1] Rajaraman A and Ullman J. Mining of Massive Datasets[M]. Cambridge, United Kingdom: Cambridge University Press, 2011: 113–114.

[2] Balazinska M, Balakrishnan H, Madden S R, *et al.*. Fault-tolerance in the borealis distributed stream processing system[C]. Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, New York, USA, 2005: 13–24.

[3] Gama J. Data stream mining: the bounded rationality[J]. *Informatica*, 2013, 37(4): 21–25.

[4] Repantis T and Kalogeraki V. Replica placement for high availability in distributed stream processing systems[C]. Proceedings of the Second International Conference on Distributed Event-based Systems, Rome, Italy, 2008: 181–192.

[5] Hwang J H, Xing Y, Cetintemel U, *et al.*. A cooperative, self-configuring high-availability solution for stream processing[C]. 2007 IEEE 23rd International Conference on Data Engineering, Istanbul, Turkey, 2007: 176–185.

[6] Chandrasekaran S, Cooper O, Deshpande A, *et al.*. TelegraphCQ: continuous dataflow processing[C]. Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, 2003: 668.

[7] Gedik B. Generic windowing support for extensible stream processing systems[J]. *Software: Practice and Experience*, DOI: 10.1002/spe.2194.

[8] Hwang J H, Balazinska M, Rasin A, *et al.*. High-availability algorithms for distributed stream processing[C]. The 21st International Conference on Data Engineering, Tokyo, Japan, 2005: 779–790.

[9] Gu Y, Zhang Z, Ye F, *et al.*. An empirical study of high availability in stream processing systems[C]. Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware, Urbanna, Illinois, USA, 2009: 1–9.

[10] Repantis T, Gu X, and Kalogeraki V. Synergy: sharing-aware component composition for distributed stream processing systems[C]. Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware, Melbourne, Australia, 2006: 322–341.

[11] Branson M, Douglis F, Fawcett B, *et al.*. CLASP: collaborating, autonomous stream processing systems[C]. Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware, Newport Beach, California, USA, 2007: 348–367.

[12] Zhang Z, Gu Y, Ye F, *et al.*. A hybrid approach to high availability in stream processing systems[C]. IEEE 30th International Conference on Distributed Computing Systems (ICDCS2010) Genova, Italy, 2010: 138–148.

[13] Balazinska M, Balakrishnan H, Madden S R, *et al.*. Fault-tolerance in the borealis distributed stream processing system[J]. *ACM Transactions on Database Systems*, 2008, 33(1): 1–44.

[14] Yu H, Gibbons P B, and Nath S. Availability of multi-object operations[C]. Proceedings of the 3rd Conference on Networked Systems Design & Implementation, San Jose, CA, 2006: 211–224.

[15] Ding W, Han Y, Zhao Z, *et al.*. Stream-oriented availability services for endpoint-to-endpoint data transmission[C]. International Conference on Cloud and Service Computing (CSC 2012), Shanghai, China, 2012: 212–218.

[16] Ding W, Han Y, Wang J, *et al.*. Feature-based high availability mechanism for extreme aggregation tasks in real-time data stream processing[J]. *Journal of Internet Technology*, 2013, 14(2): 327–340.

[17] Ding W, Han Y, Wang J, *et al.*. Feature - based high - availability mechanism for quantile tasks in real - time data stream processing [J]. *Software: Practice and Experience*, DOI: 10.1002/spe.2244.

丁维龙：　男，1983 年生，博士，助理研究员，研究方向为实时数据处理和分布式系统.

韩燕波：　男，1962 年生，博士，研究员，研究方向为云计算、服务计算和数据集成.