doi:10.3969/j.issn.1000-436x.2014.05.007

基于 *K*-means 的云化分布式 BPEL 引擎放置机制

林荣恒，吴步丹，赵耀，杨放春
（北京邮电大学 网络与交换技术国家重点实验室，北京 100876）

摘 要：针对分布式 BPEL 引擎在云中的放置问题开展研究，提出了一种基于 *K*-means 的分布式 BPEL 引擎放置机制，该机制将 BPEL 引擎放置问题模型化为相关最优化数学模型，并且将该模型映射到 *K*-means 算法进行求解。该机制还讨论了算法在不同网络拓扑随机图、树形网络拓扑的应用。最后利用统计软件 R 进行了相关实验仿真，仿真结果显示该放置机制可优化服务调用所占用的带宽资源。

关键词：BPEL；服务引擎；*K*-means；分布式计算；云计算
中图分类号：TP393.09 **文献标识码**：A **文章编号**：1000-436X(2014)05-0049-08

Approach for distributed BPEL engine placement using *K*-means

LIN Rong-heng, WU Bu-dan, ZHAO Yao, YANG Fang-chun
(State Key Lab of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: Aiming to solve the distributed BPEL engine placing problem in cloud, a *K*-means based distributed BPEL engine placing algorithm was proposed. The algorithm transforms the BPEL engine placing model into some optimization model in mathematics, and the optimization problem is solved by *K*-means algorithm. How to apply the algorithm in different network topologies was also discussed, such as random graph and tree network. In the end, statistical software R was used as experiment tool to evaluate the algorithm. Results show that the proposed method can provide a more optimized bandwidth usage of combined BPEL service execution.

Key words: BPEL; service engine; *K*-means; distributed computing; cloud computing

1 引言

BPEL 引擎是 BPEL 脚本的执行环境，传统的 BPEL 执行引擎采用集中式执行的方式，暨 BPEL 进程处于同一个容器之中。这种典型的集中式执行形态，显然不适应当前分布式、云化环境的伸缩性要求。在云环境下，集中式的 BPEL 执行引擎很难随着规模增大而进行扩容的支持，一方面，BPEL 引擎调用相关 Web 服务所产生的网络流制约着引擎本身；另一方面，相关网络流也对云数据中心的产生一定的压力，制约着数据中心的吞吐量。

针对上述需求，分布式的 BPEL 引擎技术成为本领域的研究点之一，其基本思路是将组合的业务流程进行拆分，分布运行于各个引擎之上。分布式 BPEL 引擎技术存在着两大难点：一是流程的拆分，二是引擎的选择放置。现有研究主要集中在流程拆分，对引擎放置较少涉及。然而随着云计算、数据中心网络等的发展，大规模的服务组合和提供将成为未来的趋势之一。在大规模数据中心中的组合服务调用场景下，引擎的放置问题成为了提升网络利用率的关键一环。

本研究正基于上述考虑，对服务引擎的放置问题展开研究，提出了一种基于 *K*-means 考虑网络流量代价的引擎放置方案。

2 相关工作

工作流管理引擎为企业的信息共享以及协同提供了良好的支撑。典型的 BPEL 流程包含了多个

收稿日期：2013-09-05；修回日期：2014-02-23
基金项目：国家高技术研究发展计划（“863”计划）基金资助项目(2011AA01A102)；国家自然科学基金资助项目(61003067)
Foundation Items: The National High Technology Research and Development Program of China (863 Program)(2011AA01A102); The National Natural Science Foundation of China (61003067)

需要调用的服务，传统的方式将 BPEL 置于同一的引擎范围内执行。这种集中式执行对 BPEL 引擎自身性能要求较高，且存在单点故障。现有的研究中，一个组合流程需要由分布在网络各个位置的多个引擎协同完成。为此，工作流的分割及工作流引擎的放置成为研究的热点。

现有研究主要集中在工作流分割方面，OASIS 组织及相关研究机构首先提出了一种支持大规模分布式应用的工作流系统架构[1]。在 Wutke[2]的研究中，BPEL 流程采用手动的方式进行分割，并且在每个分割的 BPEL 流程上标注了相关部署描述。Daniel 和 Frank Layman[3]最初提出一种 BPEL 流程分布式执行方法，方法中涉及对基本 BPEL 流程的拆分。而后，Rainia 和 Frank Laymann[4]提出的分布式业务流程中对循环的 BPEL 片段的协调方法，解决了复杂 BPEL 结构分割的关键问题。与此同时，IBM 相关研究中心[5]开展了模型的分割、分割模型数据流约束下的编排等一系列研究。Tan[6]给出了一种基于模块化 Petri 网的静态和动态模型分割方法，验证了模型分割的正确性，并对模型分割的策略做了初步探讨。上述研究解决了 BPEL 的分割问题，但尚未涉及到工作流引擎的负载及放置问题。

Danesh[7]分别提出了基于分布式多引擎架构的网格工作流管理系统，其主要的贡献在于对整个架构的描述及其监控等因素。Pantazoglou[8]提出了名为 BPELcube 的架构，利用 P2P 的分布式结构解决引擎调用的性能问题，其贡献也主要在如何利用 P2P 结构完成 BPEL 流程的分布执行。国内研究机构对工作流拆分以及引擎也有一系列研究，如 Wu[9]针对 BPEL 流程提出了一种线性逻辑表示，这种表示方法可用于指导工作流的分割。乔晓强等[10]提出了一种分布式协调模型的服务协作方法，为服务的分布式拆分执行提供验证和保障。毕敬等[11]则提出了一种动态服务流程模型混合分割方法，可提高流程拆分后的吞吐率。张曼等[12]则针对 BPEL 流程并行执行中交叠模式进行分析。邓水光等[13]则较早在国内提出相关动态工作流建模的方法，对工作流执行、引擎等方面均有一定借鉴意义。

3 问题分析

在传统的集中式业务引擎条件下，一个流程的调用完全由中间节点进行分发控制，所有的调用来自中心节点。在分布式引擎的条件下，各个子流程被拆分到几个主要引擎。在网络中，这些引擎位置的选择直接影响着相关整个组合工作流执行过程中所占的带宽。如图 1 所示，细虚线标识了一种服务与引擎的分配方式。其中，A1、A2、A3、B1、B2、B3 为服务节点，Ae、Be 为执行引擎，R1、R2、R3 为相应的路由器。最初的一种分配方式为{A2,B2,A1}、{Ae,B1}，显然在该分配方案中，A2、Be 的通信经过了 R1、R2、R3 路由器。该分配方案未考虑 Ae 的存在，因此存在冗余的路由。粗虚线则标识了另一种分配方案{A2,Ae,A1}、{B1，Be}，该分配方案充分考虑了引擎与服务的位置关系，减少或避免了相关冗余路由。

显然，冗余的流量的占用来自于迂回路由，即服务与服务引擎间的距离过大，通过了过多的中间点，从而导致相关带宽的浪费。为此，如何寻找合适的引擎位置使得分布的流程占用最小的网络资源，是当前系统亟需解决的问题。

4 云化分布式 BPEL 引擎结构

4.1 体系结构

分布式 BPEL 引擎的总体结构如图 2 所示。

引擎管理模块是整个系统的核心，它负责将收到的流程进行拆分并分配到各个子引擎执行以及指导各个子引擎执行后的串接问题。它包括流程拆分、引擎分配、状态维护、引擎监控等多个子模块。本论文所涉及的引擎放置机制归属于本模块。

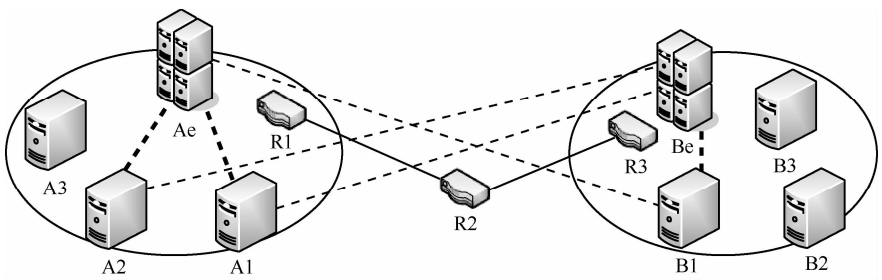


图 1 集中式分配与优化分布式分配对比

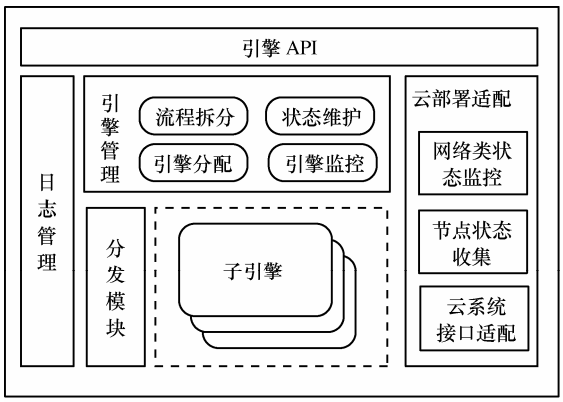


图 2　云化分布式 BPEL 引擎的总体架构

云部署适配模块是当前引擎部署到云平台的必要模块，它负责与云系统进行适配、收集各个节点的状态信息以及网络拓扑及状态等信息。这些信息将为引擎管理模块的决策提供支持。它包括网络类状态监控、节点状态收集、云系统适配等子模块。

分发模块负责将引擎管理节点与子引擎、子引擎与子引擎之间的通信维护。引擎管理与子引擎间主要传递管理信息，子引擎之间则主要传递流程中调用的返回信息。

子引擎是实际流程脚本片段的执行环境，它类似标准的流程引擎，但多了一个结果反馈模块。该模块负责将流程执行结果发送到分发模块。

在实际部署中，引擎管理与云部署适配形成管理节点，子引擎与分发管理形成实际业务节点。管理节点对过程进行拆分，拆分后分配到当前的业务节点。

4.2　分布式引擎模型

Frank Leynn 提出的流程拆分机制，可以为分布 BPEL 流程的分拆提供基础。本文的流程分拆均是在其基础上完成的。一般的流程分拆主要在典型的 Web 环境下，不考虑引擎与引擎之间的流量问题。在流程分拆的过程中，包括多个实体：服务引擎、流程片段、协调引擎。协调引擎则是为了保障拆分之后的流程具备原有流程的语义，协调引擎（协调器）通过 WS-Coordinate 以及 WS-Transaction 等协议保障分布流程的语义无二义性。一般流程的分拆仅是引擎级别的分拆，但保留其原有协调器，如图 3 所示。这种分拆方式一方面提供了分布的引擎，另一方面保留对协同的统一管理。

图 3 展示了 3 种部署方式：集中式部署、半分布式部署、全分布式部署。

1) 集中式部署方式（如图 3(a)所示），所有的协调、执行调度均在集中节点（EC）上进行。EC 节点既包含引擎的功能也包含协调器的功能。

2) 半分布式部署方式（如图 3(b)所示），首次将执行与协调进行分离。执行节点（E1、E2、E3）负责 BPEL 片段的调用，协调节点（C）则负责各个 BPEL 片段流程之间的数据、逻辑协调。

3) 全分布式部署方式（如图 3(c)所示），将半分布式部署方式中的协调节点(C)的功能分解到执行节点之中，形成新的执行与协调节点(EC1、EC2、EC3)。EC*x* 节点包含 BPEL 片段调用以及与本片段流程相关的协调任务。

集中式部署方式对其引擎与协调节点（EC）压力较大，显然不适应未来云环境下的需求。半分布式环境下的引擎节点（E1、E2、E3）处于分布状态，然而引擎间的协调工作仍由统一的协调节点(C)进行处理。本文为了简化相关建模过程，将协调节点(C)的功能拆解于各个引擎节点中，形成 EC1、EC2、EC3。依靠 EC1、EC2、EC3 的交互完成相关的协调工作。

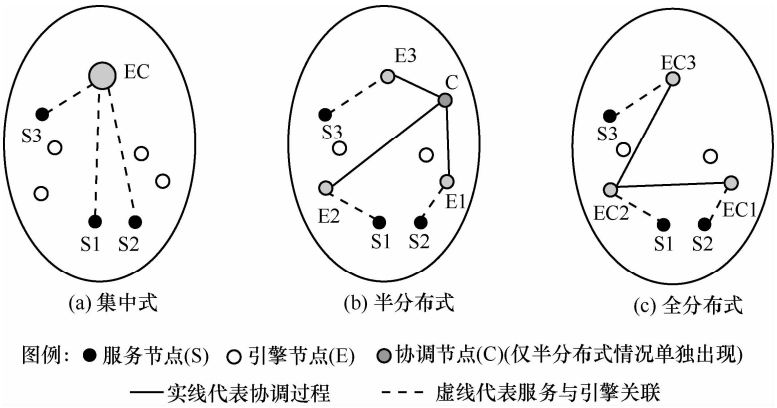


图 3　集中式、半分布式、完全分布式部署示意

5 基于 *K*-means 的引擎放置机制

5.1 问题数学建模

网络拓扑特别是数据中心网络包括多种类型典型的如胖树、B-Cube 等结构，Aoun 等[14]表明了对于不同类型的网络均可采用类似的网络定义方法进行分析。为此，本文首先忽略网络拓扑造成的差异，从通用网络进行分析，而后对具体的拓扑进行细化。问题转化为如何在网络中寻找合适的引擎位置，使引擎、被调用服务之间的调用距离、流量最小。

定义 1 BPEL 调用序列，$BP=a_1,a_2,a_3,\cdots,a_n$，为简便起见该执行流程的 a_i 隐性包含了循环等非线性流程。BP 级别仅包含顺序逻辑。

定义 2 网络的节点 $V=\{v_1,v_2,\cdots,v_n\}$，其中，网络的节点为 $v=\{id,c,t\}$，id 代表节点的唯一标识，c 为节点的能力，t 为节点的类型，目前的类型包括服务节点、引擎节点、其他节点 3 种类型。服务节点标识该节点具有 Web 服务提供能力，可被调用。引擎节点代表该节点为引擎节点可调用其他服务节点。其他节点则指非服务、引擎节点外的节点类型，可能包括管理节点、辅助节点等，由于与本文关系不大，为此统称为其他节点。

定义 3 网络边的集合 $E=\{e_1,e_2,\cdots,e_k\}$，其中，网络边为 $e_{x,y}=\{v_x,v_y,w\}$，v 代表起始和终结节点，w 代表权重，是距离及带宽所形成的权重，在本文中 w 与带宽成正比例。

定义 4 网络为 $G=(V,E)$，网络由相关的边及点构成。

定义 5 网络的路径 $p=(v_1,v_x,\cdots,v_y)$，标识网络中的一条路径，标记调用方向。BP 流程中的每个 a_i 对应的路径标记为 $p(a_i)$，针对特定的流程 a_i，其服务所在的位置设为 v_s^i，引擎所在位置 v_e^i，则 $p(a_i)$ 可表示为

$$p(a_i)=\begin{cases}\{(\mathrm{v}_s^i,\mathrm{v}_e^i),(\mathrm{v}_e^i,\mathrm{v}_e^{i+1})\}, i<sizeof(bp)\\ \{(\mathrm{v}_s^i,\mathrm{v}_e^i)\}, i=sizeof(bp)\end{cases} \tag{1}$$

$p(a_i)$ 在 $i<bp$ 的总流程个数时，其路径可由服务到引擎的路径，加上引擎与引擎的路径，而针对 $i=bp$ 流程总数时，即最后一个服务调用时，其路径为服务与引擎之间的路径。为此，总体 BPEL 流程的路径为 $p(bp)=\sum p(a_i)$。若已知流程的访问次数、频率、每次访问所占的带宽，根据上述关系可推出该流程占用网络的流量。

问题转化为，在给定访问数、访问所占带宽、访问频率的前提下，最小化 $p(bp)$ 所经过的路径的流量，即 $Minimize(flow(\sum p(a_i))$。

5.2 基于 *K*-means 的放置算法

显然，最终目标相当于针对给定 BP、G，寻找合适的引擎位置使得目标函数最小。

所建立的目标，相当于 $p(a_i)$ 的和最小。依照分治及贪心的思想，即每个 $p(a_i)$ 最短，每个服务引擎到其所涉及调用的服务之间的网络距离最短。如果针对特定的流程 a_i,其服务所在的位置设为 v_s^i，引擎所在位置 v_e^i，上述最小化目标可转化为

$$Minimize\sum_{a_i\in BP}\Big(\sum_{v_j\in paths(a_i)}\|(v_e^i,v_s^i)\|^2\Big)\times 2+\sum_{i=1}^{sizeof(BP)-1}\|v_e^i-v_e^{i+1}\|^2 \tag{2}$$

由于实际的操作中，引擎与引擎之间的路由属于引擎自身的开销，本处将重点考察引擎与服务之间的消耗。基于式(2)可转化为服务引擎与服务之间的消耗。

考虑到 v_e^i 可能的个数小于 v_s^i，因此，实际中必然多个 v_s 关联某个 v_e。为了方便起见，用 VE 标识 v_e^i 的集合，假设 v_e^i 的个数为 k，将 v_e^i 重新标记为 $u_{i'}$，则 $VE=\{u_1,\cdots,u_k\}$。如果将某个 v_e 即($u_{i'}$)所关联的 v_s 记做 S_e，将 $S_{i'}$ 定义为 $S_{i'}=S_e\cup\{u_{i'}\}$ 根据上述转化定义，可将式(2)转化为

$$\arg\min_S\sum_{i=1}^{k}\sum_{x_j\in S_i}\|x_j-u_i\|^2 \tag{3}$$

其中，重新将 i' 记作 i。

因此，问题域实质可以转化为，每个执行引擎及其所涉及的服务所在网络区域设置为某个聚类，聚类的中心点为执行引擎，服务到中心点的距离和最小。

针对上述公式的求解，可采用以下迭代公式

$$S_i^{(t)}=\{x_p:\|x_p-m_i^{(t)}\|\leqslant\|x_p-m_j^{(t)}\|,\forall 1\leqslant j\leqslant k\} \tag{4}$$

$$m_i^{(t+1)}=\frac{1}{|S_i^{(t)}|_{x_j\in S_i^{(t)}}}\sum_{x_j\in S_i^{(t)}}x_j \tag{5}$$

具体的算法过程描述如下。

1) 从 n 个数据对象任意选择 k 个对象作为初始聚类中心。(最终的聚类中心可作为协调节点备选点)。

2) 根据每个聚类对象的均值（中心对象），计算每个对象与这些中心对象的距离；并根据最小距离重新对相应对象进行划分。

3) 重新计算每个（有变化）聚类的均值（中心对象）;（此过程为聚类中心点的变迁，使得聚类中心朝着距离最小的趋势发展）。

4) 计算标准测度函数，当满足一定条件，如函数收敛时，则算法终止；如果条件不满足则回到步骤 2)。

上述算法，是基于 *K*-means 算法的一个典型过程。在典型的 *K*-means 算法中，一般采用欧式距离，本算法的距离则表示为

$$\| x_j - u_i \| = \| p(x_j, u_i) \| \tag{6}$$

其中，距离转化为是相应的路径的距离，并可根据需求增加权重影响。因此，针对随机图、树形结构的拓扑，上述式子均可进行选择。随机图中该距离表现为两点之间的最近路径。在树形结构图中，该路径同样是最近路径。树形结构最近路径的选择方法可通过最近的公共父节点的算法。

具体算法伪代码如下。

K-means 聚类算法

//输入参数：*k* 为聚类个数，*p* 为相关随机概率，*times* 为最大迭代次数，*m* 为种群大小

/**作为输入，用户需明确聚类个数 *k*，产生随机图的基础概率值 *p*，允许算法的最大迭代次数 *times* 以及当前生成随机图规模的种群数大小。聚类个数取决于引擎的个数，个数不小于 2 个；随机图的概率一般在 0.3~0.6 之间，迭代次数一般在 10 次以上，随机图种群大小取决于服务的个数，一般不小于 50**/

```
Function kmeanCluster    (k,p,times,m)
Begin
    Time=0;
    /**生成随机图矩阵, 基本算法通过 2 个高斯的随机函数分别对横坐标和纵坐标进行生成。**/
    matrixXY=generateGaussionMatrix(p,m);
    //初始化中心点集合,随机选择即可
    Centers=sampleCenters(matrixXY,k)
    /**计算初始聚类群，通过计算节点距离到各个中心距离，归属到最近的中心的类中。**/
    initClusters=initCluster(Centers,matrixXY)
    //计算每个中心到其成员的距离和集合
    initDist=dist (initClusters,Centers)
    //开始循环迭代
    While (true)    Begin
      //计算当前的虚拟中心位置
     imCenters=computeCenters(tempClusters);
     /**使用式(5)判断 imCenters 是否与 Centers 一样**/
        If    is Equal(imCenters,Centers)
            Break;
     //根据当前的中心计算新的 tempClusters
     tempClusters = initCluster(imCenters,matrixXY);
        /**对比前后生成的聚类，每个聚类成员到其中心的距离和，类似式(4)**/
        Tempdist=dist(imCenters,tempClusters)
        /**对比结果将距离较小的聚类的中心点保留在 Centers**/
        Centers=Compare(tempDist,initDist)
        //判断是否满足迭代次数
        Time++;
        If (Time==Times)    then break;
    End //while
/**返回与当前中心点集合最接近的实际节点集合**/
    Return    getSimilar(Centers)
End
```

根据上述算法过程，典型的时间复杂度为 $O(tKmn)$，其中，t 为迭代次数，K 为聚类的个数，m 为种群的大小，n 为维度。在本算法中，K 的个数取决于服务引擎的个数，m 取决于服务个数，n 一般为 2 维。t 的取值范围一般在 10~30 以内。由于服务引擎与服务之间的位置并非实时改变，因此本时间复杂度可满足系统实际需求。

6　实验仿真与系统应用

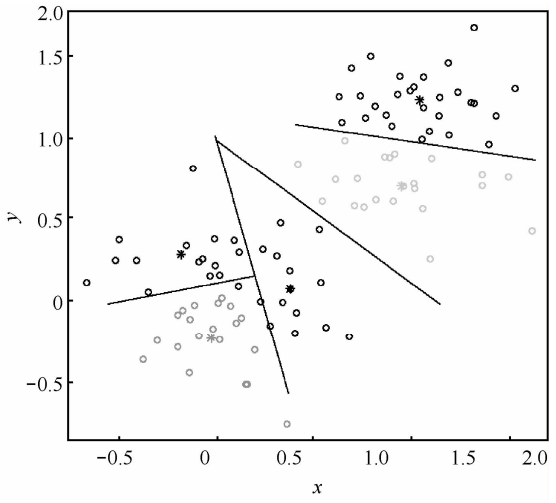
6.1　实验仿真

为了进行验证，采用 R 语言在仿真平台中创建相关的仿真环境，其中，主要以随机图拓扑为主。随机图的“随机”二字体现在边的分布上。一个随机图实际上是将给定的点之间随机地连上边而形成的一种图。假设将一些“纽扣”散落在

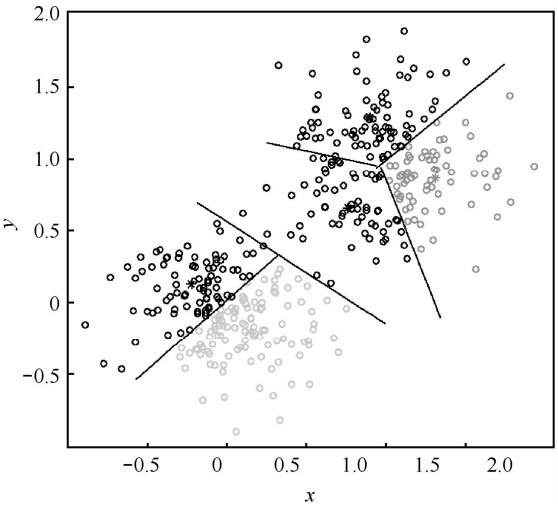
地上，并且不断随机地将 2 个“纽扣”之间系上一条线，这样就得到一个随机图的例子。其中，边的产生可以依赖于不同的随机方式，这样就产生了不同的随机图模型。

随机图一般定义为 $G_{\text{random}} = G'(n,p)$，其中，对于图的边是以独立概率 $p:(0<p<1)$ 进行连接。一个具备 m 条边的随机图的概率为 $p^m(1-p)^{N-m}$。本实验中，首先确定图的规模即节点个数，顺序创建每个节点，而后根据以两两节点以概率 p 互联，生成所需的随机图。为了简便计算，边的权重均设为 1。依照随机图，根据距离转换公式，节点之间的距离可转化为节点之间最短路径的距离。

在 k=5，p=0.3，K-means 迭代次数 25 的时候，且样本规模为 100 和 400，如图 4 所示。



(a) 100 样本规模的聚类结果



(b) 400 样本规模的聚类结果

图 4　100、400 样本的聚类结果展示

为了验证结果的有效性，将样本为 100，k 为 5 的实验环境，重复运行 100 次，并与相关随机选择算法选择对比。其中，随机选择算法以随机选 20 位为一个聚类。

图 5 给出各个聚类内部的距离平方和，图中虚线部分代表随机算法所得到的距离平方和，实线代表本文所用算法的结果。通过该图可明显发现本文所用算法的平方和远小于随机算法，说明在各个类别内部点到聚类中心的距离平方和较小。

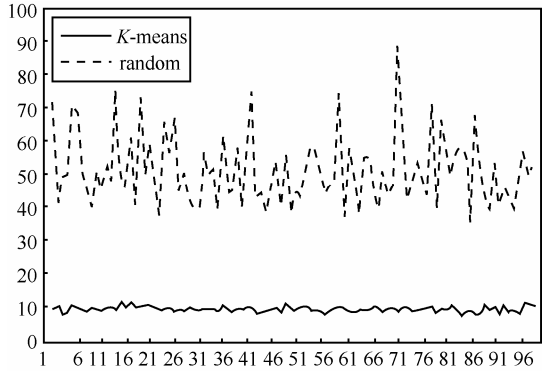


图 5　各个群体内的距离平方和比较

图 6 给出群体间的距离平方和（聚类中心之间），同样虚线代表随机算法，实线为本文算法。与图 5 相反，实线大于虚线。该结果说明本文算法各聚类之间区分度较好，交叠较少。

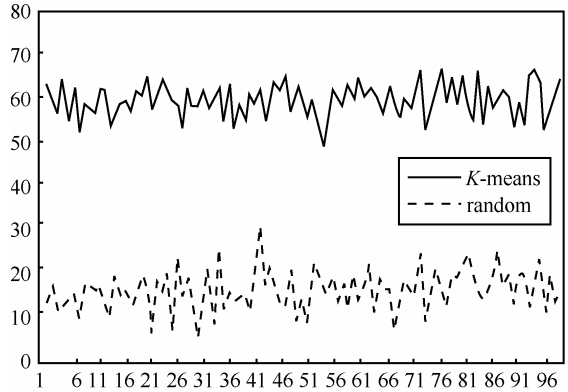


图 6　各个群体之间的距离平方和比较

图 7 则是在图 5、图 6 的基础上，根据式（2）所得出的。该图反映实际网络调用占用网络的距离和。反馈到实际网络，即调用的距离较小，从而占用的带宽也相对小。在实验中，若此时具备一定的访问量，图 7 的访问距离将与占用带宽成正比（带宽=所有链路上的访问量×每次访问带宽，显然若

访问距离长了，从网络整体上的带宽占用将提升），即不采用本算法的所占的带宽将远大于采用本算法的带宽。

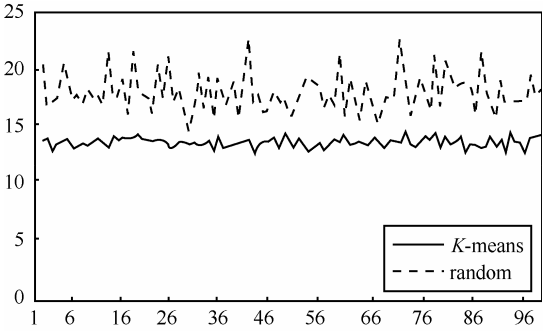


图 7　总调用距离对比

实验结果表明，本算法一定程度解决分布式 BPEL 引擎的位置选择、社区划分等问题，从而解决分布式服务调用过程中的带宽占用问题。由于现有的研究主要集中在如何进行流程拆分，因此本文的对比采用了与随机算法的对比。在今后的工作中，本课题组将延续本工作的基础，增加与其他可能工作的对比及算法优化。

6.2　系统应用的场景说明

分布式引擎管理系统可直接运行于 IaaS 之上，典型的如图 8 所示。其工作过程一般包含如下步骤。

1) 通过系统自身的适配接口可直接向 I 云申请多个虚拟机用于引擎的放置。

2) I 云返回虚拟机资源，分布式引擎管理将运行 *k*-means 算法计算引擎位置。

3) 分布式引擎管理可发起引擎放置选择的操作，将引擎放置于所需位置。

4) 若现有虚拟机均不满足放置位置，分布式引擎管理可再请求 I 云分配新的虚拟机。

可见，分布式引擎管理可作为云运营商的一个基础设施部署于 I 云之上，用于提供分布式执行引擎。*K*-means 引擎放置算法仅在每次引擎需要增加的时候被运行，因此其时间复杂度与服务执行无关，因此 $O(tKmn)$的时间复杂度可被接受。同时，由于 *K*-means 引擎放置机制是在分布式引擎管理系统中运行，因此其所占 CPU 与实际业务系统无关。但考虑实际情况，引擎位置的调整一般在业务量较小的时段进行，若引擎迁移过程也可采用 I 云提供的实时迁移功能。

7　结束语

BPEL 工作流的云化分布执行是当前工作流技术的实际需求和发展方向之一。本研究立足于寻求分布式 BPEL 引擎位置选择的关键问题，将现有 BPEL 引擎选择的问题映射到聚类算法领域，并通过定义新的距离公式等完成 BPEL 引擎的位置选择。现有工作仍存在一些不足，如初始点的优化、防止算法的局部最优等问题，均有待未来工作的完善和加强。

参考文献：

[1] FARAHBOD R, GLÄSSER U, VAJIHOLLAHI M. Advancesin Theory and Practice[M]. Berlin Heidelberg: Springer, 2004.

[2] WUTKE D, MARTIN D, LEYMANN F. Model and infrastructure for decentralized workflow enactment[A]. Proceedings of the 2008 ACM Symposium on Applied Computing[C]. Fortaleza, Ceara, Brazil, 2008. 90-94.

[3] WUTKE D, MARTIN D, LEYMANN F. A method for partitioning BPEL processes for decentralized execution[A]. Proceedings of the 1st Central-European Workshop on Services and Their Composition[C]. Stuttgart, Germany, 2009.2-3.

[4] KHALAF R, LEYMANN F. Coordination for fragmented loops and scopes in a distributed business process[J]. Information Systems, 2012, 37(6): 593-610.

[5] NANDA M G, CHANDRA S, SARKAR V.Decentralizing execution of composite web services[J]. ACM Sigplan Notices, 2004, 39(10): 170-187.

[6] TAN W, FAN Y. Dynamic workflow model fragmentation for distributed execution[J]. Computers in Industry, 2007, 58(5): 381-391.

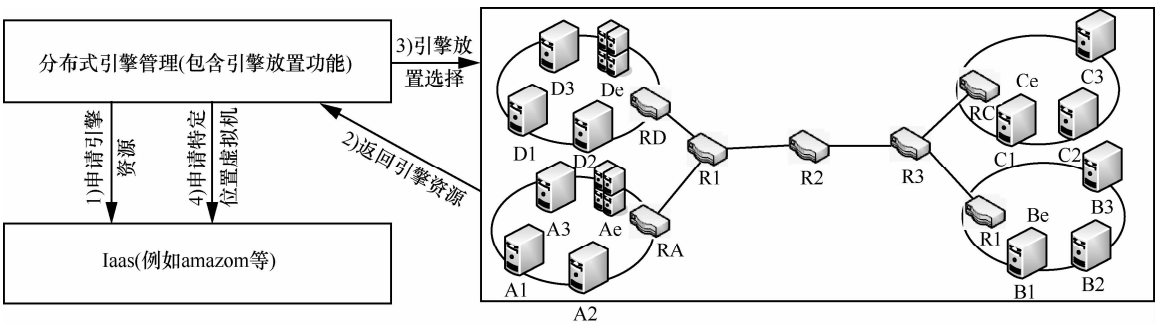


图 8　系统应用场景示意

[7] DANESH M H, RAAHEMI B, KAMALI S M A, *et al*. A distributed 2012 25th IEEE canadian conference on service oriented infrastructure for business process management in virtual organizations[A]. The 25th IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)[C]. Montreal, QC, Canada, 2012.1-6.

[8] PANTAZOGLOU M, POGKAS I, TSALGATIDOU A. Decentralized enactment of BPEL processes[J]. IEEE Transactions on Services Computing, 2013, 99(1):1-12.

[9] WU J, JIN L. A linear logic representation for BPEL process protocol[J]. Applied Mathematics & Information Sciences, 2011, 5(2): 25-31.

[10] 乔晓强, 魏峻, 黄涛. 基于分布式协调模型的服务协作方法研究[J]. 软件学报, 2009, 20(6): 1470-1486.
QIAO X Q, WEI J, HUANG T. Service collaboration approach based on decentralized mediation model[J]. Journal of Software, 2009, 20(6): 1470-1486.

[11] 毕敬, 朱志良. 动态服务流程模型混合分割方法及应用[J]. 东北大学学报(自然科学版), 2010, 31(5): 639-643.
BI J, ZHU Z L. Research and application of hybrid fragmentation method for dynamic service process model[J]. Journal of Northeastern University(Natural Science), 2010, 31(5): 639-643.

[12] 张曼, 段振华, 王小兵. BPEL 流程建模中的交叠模式分析与转换[J]. 软件学报, 2011, 22(11): 2684-2697.
ZHANG M, DUAN Z H, WANG X B. Analysis and transformation of overlapped patterns in BPEL process modeling[J]. Journal of Software, 2011, 22(11): 2684-2697.

[13] 邓水光, 俞镇, 吴朝晖. 动态工作流建模方法的研究与设计[J]. 计算机集成制造系统, 2004, 10(6):601-608.
DENG S G, YU Z, WU Z H. Research and design of dynamic workflow modeling method[J]. Computer Integrated Manufacturing Systems, 2004,10(6): 601-608.

[14] AOUN R, ABOSI C E, DOUMITH E A, *et al*. Towards an optimized abstracted topology design in cloud environment[J]. Future Generation Computer Systems, 2013, 29(1): 46-60.

作者简介：

林荣恒（1981-）,男，福建厦门人，博士，北京邮电大学讲师，主要研究方向为云计算、服务计算、业务流程及安全。

吴步丹（1982-），女，江西南昌人，博士，北京邮电大学副教授，主要研究方向为服务计算、需求工程、云计算。

赵耀（1979-），男，四川攀枝花人，博士，北京邮电大学副教授，主要研究方向为服务搜索、新一代通信网络。

杨放春（1957-），男，北京人，博士，北京邮电大学教授、博士生导师，主要研究方向为新一代通信网络、服务计算、云计算、车联网。