软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(11):3241−3259 [doi: 10.13328/j.cnki.jos.005480]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于数据集分割的云工作流模型库并行检索方法*

黄 华[1,2], 彭 蓉[1], 冯在文[1]

[1](武汉大学 计算机学院,湖北 武汉 430072)

[2](景德镇陶瓷大学 信息工程学院,江西 景德镇 333403)

通讯作者: 彭蓉, E-mail: rongpeng@whu.edu.cn

摘 要: 在由多个行业云服务平台组成的集成服务平台中,随着行业云服务平台加盟数及各平台下租户数量的不断增多,其底层的云工作流模型库的规模也必将不断增大.当云工作流模型库的规模超大时,需要一种效率更高的并行检索方法去满足云工作流模型库高效检索的需求.鉴于此,采用均匀划分法或自动聚类法对大规模云工作流模型库进行合理的子集划分,并结合前期工作中已改进的基于图结构的流程检索算法,提出了基于数据集分割的大规模云工作流模型库并行检索方法.该方法主要包括4种流程并行检索算法:基于均匀划分模型集的静态并行检索算法、基于均匀划分模型集的动态并行检索算法、基于自动聚类模型集的静态并行检索算法和基于自动聚类模型集的动态并行检索算法.最后,在模拟生成的大规模流程集及真实的云工作流模型库中对这 4 种并行检索算法的检索效率进行了实验评估.

关键词: 云工作流;数据集分割;流程检索;并行检索

中图法分类号: TP311

中文引用格式: 黄华,彭蓉,冯在文.基于数据集分割的云工作流模型库并行检索方法.软件学报,2018,29(11):3241−3259. http://www.jos.org.cn/1000-9825/5480.htm

英文引用格式: Huang H, Peng R, Feng ZW. Parallel retrieval approach of cloud workflow model repositories based on data set partitioning. Ruan Jian Xue Bao/Journal of Software, 2018,29(11):3241−3259 (in Chinese). http://www.jos.org.cn/1000-9825/5480.htm

Parallel Retrieval Approach of Cloud Workflow Model Repositories Based on Data Set Partitioning

HUANG Hua[1,2], PENG Rong[1], FENG Zai-Wen[1]

[1](School of Computer Science, Wuhan University, Wuhan 430072, China)

[2](School of Information Engineering, Jingdezhen Ceramic Institute, Jingdezhen 333403, China)

Abstract: In the integrated service platform composed of multiple industry cloud service platforms, with the increasing of the number of cloud service platforms and theirs tenants, the scale of its underlying cloud workflow model repository will be increasing. When the scale

* 基金项目: 国家重点研发计划(2017YFB0503700, 2016YFB0501801); 国家自然科学基金(61170026, 61100017); 国家标准研究计划(2016BZYJ-WG7-001); 中央高校基本科研业务费专项资金(2012211020203, 2042014kf0237); 江西省重点研发计划(20171ACE50022); 江西省自然科学基金(20171BAB202011); 江西省教育厅科技项目(GJJ160906)

Foundation item: National Key Research and Development Program of China (2017YFB0503700, 2016YFB0501801); National Natural Science Foundation of China (61170026, 61100017); National Standard Research Program (2016BZYJ-WG7-001); Fundamental Research Funds for the Central Universities (2012211020203, 2042014kf0237); Key Research and Development Program of Jiangxi Province (20171ACE50022); Natural Science Foundation of Jiangxi Province (20171BAB202011); Science and Technology Research Project of Jiangxi Province Education Department (GJJ160906)

本文由面向智能制造的业务过程管理与服务技术专题特约编辑王建民教授、刘建勋教授推荐.

收稿时间: 2017-07-20; 修改时间: 2017-09-16; 采用时间: 2017-11-14; jos 在线出版时间: 2017-12-06

CNKI 网络优先出版: 2017-12-06 15:37:35, http://kns.cnki.net/kcms/detail/11.2560.TP.20171206.1537.027.html

© 中国科学院软件研究所 http://www.jos.org.cn

of the cloud workflow model repository is super large, the existing retrieval methods of large-scale process model repositories still can't meet the needs of efficient retrieval of cloud workflow model repositories, therefore, it is necessary to study a more efficient parallel retrieval method. To address this issue, this paper adopts two data partitioning modes, equipartition and clustering based partitioning, to divide large-scale cloud workflow model repositories into small pieces. Combined with the improved process retrieval algorithm proposed in authors' previous work, a series of data partitioning based process parallel retrieval approaches are put forward to accelerate the large-scale process retrieval. These approaches mainly include four kinds of process retrieval algorithms from static/dynamic parallel retrieval algorithm based on uniform/automatic clustering partitioning model sets. Finally, based on the large-scale simulation process model library and the actual cloud workflow model repository, experiments are conducted to evaluate the efficiency of four parallel retrieval algorithms.

Key words: cloud workflow; data set segmentation; process retrieval; parallel retrieval

伴随着云计算及流程管理技术的广泛应用,各种行业云服务平台的服务信息系统将积累大量的业务流程模型;而与此同时,一种由行业云服务平台联盟组成的科技服务集成平台也正在兴起[1].例如,由 2012 年国家科技支撑计划课题"高新技术产业集群科技服务平台研发与应用"(项目编号:2012BAH25F00)资助研发的高新区科技服务集成平台(简称"科集网",访问网址:http://www.tsip.com.cn/)自上线以来,已聚集多个行业云服务平台及大量服务提供商和服务产品(到目前为止,科集网已聚集 10 多个行业云服务平台,收录了 5 000 多个优秀服务商和 30 000 多个优质服务产品).在这些行业云服务平台中,仅陶瓷云服务平台(http://www. pasp.cn)就累积了 2 000 多个企业用户(租户)及 10 万多个业务流程模型.显然,随平台加盟数量及各行业云服务平台租户数的增多,整个高新区科技服务集成平台将累积海量流程模型,也即平台底层的云工作流模型库的规模将超大[2].虽然在前期工作[3]中,我们提出的流程检索方法能提高大规模云工作流模型库的检索效率,但是当云工作流模型库的规模变得超大时,文献[3]中提出的改进方法将满足不了流程高效检索的需求.这是因为单台计算机的串行检索能力毕竟有限,此时需要多台计算机或多个处理器进行"团队作战",也就是流程的并行检索.

鉴于此,本文将采用均匀划分法或自动聚类法对大规模云工作流模型库进行合理的子集划分,并结合前期工作中已改进的两阶段流程检索算法[3],提出基于数据集分割的大规模云工作流模型库并行检索方法,以进一步提高大规模云工作流模型库的检索效率.本文的贡献主要有以下 3 点.

- 通过均匀划分法或自动聚类法对大规模云工作流模型集进行合理的子集划分,得到均匀划分模型集及自动聚类模型集;
- 结合改进后的两阶段流程检索算法,基于两种模型划分集及静态、动态两种子集分配方式,提出了 4 种流程并行检索算法,并将其应用于大规模云工作流模型库的高效检索中;
- 在大规模模拟流程模型集及真实的云工作流模型库中做了大量验证实验,验证了方法的有效性.

本文第 1 节给出相关研究工作.第 2 节对相关基础知识进行介绍.第 3 节给出基于数据集分割的流程并行检索过程,包括流程模型子集划分算法及基于两种模型划分集及两种子集分配方式,提出了 4 种流程并行检索算法:基于均匀划分模型集的静态并行检索算法、基于均匀划分模型集的动态并行检索算法、基于自动聚类模型集的静态并行检索算法及基于自动聚类模型集的动态并行检索算法.第 4 节是在模拟生成的大规模流程模型集及真实的云工作流模型库中进行实验评估及结论分析,以验证本文所提算法的有效性.第 5 节是本文的总结及下一步工作展望.

1 相关研究

1.1 流程检索方法

由于流程检索的核心就是找到最为相似的流程并反馈给用户,所以大量流程检索方面的工作都是围绕流程相似度计算进行的[4−15].例如,文献[9]给出了一种基于流程模型的查询语言(APQL).文献[10]基于 BPEL 提出了一种业务过程模型检索语言 BP-QL,BP-QL 应用 BPEL 图(路径)进行流程的查询.文献[11]提出了基于图结构的业务流程模型检索框架,主要利用 BPMN-Q 语言匹配业务流程图和查询模型图的相似度,从而检索出相关的

© 中国科学院软件研究所 http://www.jos.org.cn

业务流程.文献[12]将业务流程模型转为业务流程图,然后对相似度检索中用到的图匹配算法进行了分析,并通过实验对各种算法的性能进行了比较验证.实验结果显示,从查准率和执行时间等综合指标来看,快速贪心算法优于其他算法.后来,文献[13]提出了一种基于最小深度优先搜索编码的新的流程检索方法,将图的编辑距离转为比较最小 DFS 码的字符串编辑距离,降低了流程模型图相似度计算的复杂度,从而提高了流程检索效率.文献[14]利用特征度量树来有效评估模型之间的相似度,从而根据查询的条件参考模型,将模型库中的模型分为相关的、无关的以及潜在相关的这 3 类模型,其中,只有潜在相关的模型必须利用基于图编辑距离的相似度度量方法与查询的条件模型进行比较,从而提高了流程检索效率.文献[15]先利用聚类技术,综合流程建模语言相似性、模型结构相似性对流程模型进行预处理,从而降低模型查询空间,提高检索效率.以上研究工作在查询过程中没有应用任何索引去过滤模型库,只是通过流程相似度计算或一些特殊的流程查询语言对流程模型库进行“硬”搜索(穷举查询).因此,这些方法只能应用于小规模流程模型库的检索,并不满足大规模流程模型库的高效检索需求.

为了提高流程检索的效率,一些研究者提出了许多基于索引的流程检索方法.例如,文献[16]在图结构的基础上提出一个基于特征的相似度查询算法,利用图数据库中的索引特征来过滤一些不需要进行相似度计算的图,从而提高流程检索的效率.文献[17]提出一种基于度量树的索引方法,这种索引结构主要利用对象特征值之间的距离来对检索空间进行划分,并利用图的编辑距离来计算流程模型之间的相似度,从而提高检索效率.文献[18]提出用多种索引去分别加速 4 种流程检索过程.文献[19]提出了一个基于特征网(FNet)的索引去加速流程检索的方法,实验显示,该方法比当时已有的流程检索方法快了近两个数量级.后来,文献[20]提出了基于节点索引(task index)的两阶段流程检索方法.在该方法中,第 1 阶段为过滤阶段:通过节点索引进行模型的筛选;第 2 阶段为精化阶段:应用 Ullman’s subgraph isomorphism 算法[21]对过滤后的候选模型集进行子图同构验证.实验表明,该方法能有效提高大规模流程模型库的检索效率,其检索效率也优于文献[19]中的检索方法.因此,在文献[3]中,文献[20]被作为我们前期工作扩展与比较的基准方法.

1.2 并行检索方法

并行计算能够利用多台计算机或者多个处理器的计算能力及存储资源来解决大规模问题.当前并行计算技术已相对比较成熟,基于并行检索技术的信息检索方法(也称为并行检索)也已有较多研究工作[22−31].然而,将并行检索技术应用于超大规模流程模型库高效检索的工作却很少.

并行检索可分为检索算法的并行化实现[32−35]及基于数据集分割的并行检索[25−27].为了快速应用我们在文献[3]中已改进的两阶段流程检索算法,以进一步提高大规模云工作流模型库的检索效率,本文的研究暂只考虑基于数据集分割的流程并行检索(在未来工作中,将对如何实现检索算法的并行化[23−35]进行深入研究).为了更好地理解本文提出的方法,下面将给出与本文研究方法相关的一些预备知识.

2 预备知识

本文主要研究基于数据集分割的大规模云工作流模型库并行检索方法,其主要涉及流程查询、并行检索、数据集分割等相关概念.本文研究扩展与比较的基准方法是前期工作文献[3]中提出的大规模流程检索方法,本文中,单条流程查询所涉及的流程模型、查询算法与步骤在文献[3]中有详细说明,因此,本文由于篇幅限制,将不再介绍流程查询的相关知识,下面主要介绍并行检索及数据集分割的相关基础知识.

2.1 并行检索

并行检索主要依赖并行计算技术.为了理解并行检索策略,下面给出信息检索的一般过程,如图 1 所示.

在图 1 中,用户提交一条原始查询,代理程序(broker)对原始查询进行处理(如查询的分析转换或格式化处理等),然后将处理后的查询发给检索程序,检索程序找到检索结果并进行处理(如排序)后返回给代理程序,代理程序经过必要的处理(如结果的归整、合并等)将最终结果返回给用户.

© 中国科学院软件研究所 http://www.jos.org.cn

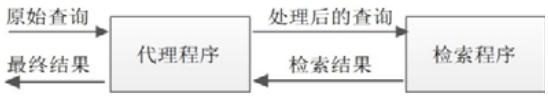

Fig.1 General process of information retrieval

图 1 信息检索的一般过程

从以上描述可以看出,信息检索有可挖掘的并行处理潜力.根据对象的不同,并行检索的实现方式总体上可分为以下两种[25].

1) 多条查询之间的并行处理

一个最自然的想法就是利用并行计算技术对多条查询的处理并行化,即每个处理器处理不同的查询,每个查询的处理之间相互独立,最多只对共享内存内的部分代码或者公有数据实行共享.这种方法也称为任务级的并行检索,它可以同时处理多个查询请求,从而提高检索的吞吐量.图 2 显示了 3 条不同查询在 3 个处理器上的并行处理过程.每条原始查询通过代理(也可同时运行多个代理程序,每个代理分别处理一条查询)发送到不同检索程序(每个处理器上运行一个检索程序)上去执行,每个检索程序的结果通过代理程序返回到不同查询的发起者.

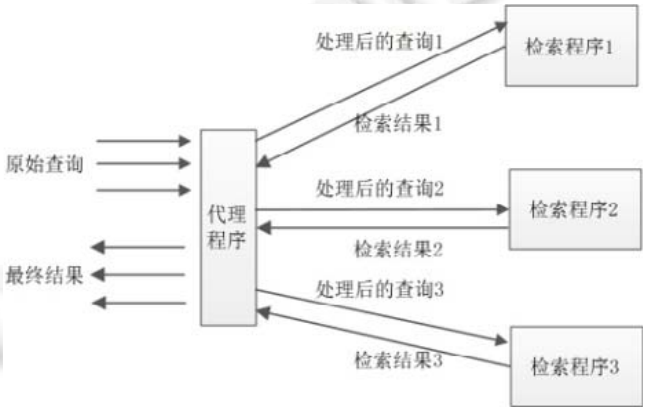

Fig.2 Parallel processing between queries

图 2 查询间的并行处理过程

查询间并行化策略是从一般检索升级到并行检索的最简单方法.简单地说,就是将检索系统复制多份(数据可以复制也可以不复制),每份分别处理不同的查询请求.

2) 单条查询内部的并行处理

即对单条查询的计算量进行分割,将分割后得到的多个子任务分配到多个处理器上的搜索进程上去执行.这种检索也称为进程级并行检索.将单条查询分成多个子任务的方法通常有两种.

- 一种是查询分割,它先将查询分解成多个子查询(如将一个多关键词查询分成多个单关键词查询),然后,对同一数据集用每个子查询分别进行检索,最后,将每个子查询的检索结果合并成最终结果;
- 另一种称为数据集分割,它事先将数据集分割成多个子集合,对同一条查询分别检索多个子集合数据,然后将每个子集合上的结果合并成最终结果.

图 3 给出了一个单条查询内部并行处理的示意图:原始查询发送给代理程序,代理程序将一条原始查询划分成多条子查询,每条子查询分别发送给一个搜索进程进行处理,各进程返回的子结果在代理上进行综合,得到最后的总结果返回给用户.

在进行多条查询之间的并行处理时,信息检索系统中的数据结构通常不需要改变;而对于单条查询内部的并行处理,则需要对原有串行信息检索的数据结构做相应的改变.在本文的研究中,应用并行检索策略对大规模

© 中国科学院软件研究所 http://www.jos.org.cn

云工作流模型库进行基于图结构/行为的高效检索时,是通过数据集分割的方式进行单条查询内部的并行处理.

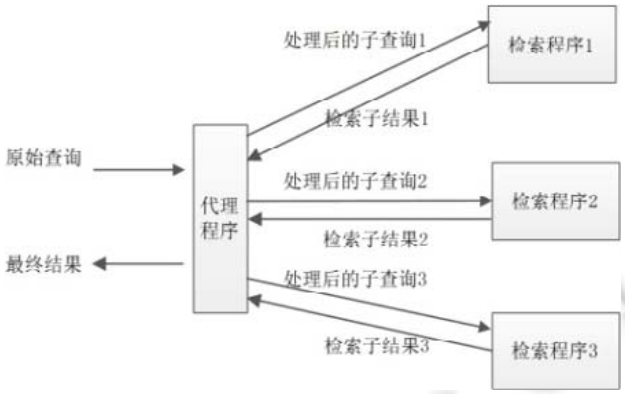

Fig.3 Parallel processing among a query

图 3 查询内部的并行处理过程

2.2 数据集分割

并行检索采用数据集分割时,有两种实现方法:逻辑倒排表分割方法和物理倒排表分割方法[27,32].这两者的数据集都在物理上分成多个子集合,但它们的不同之处在于:逻辑倒排表分割并不在物理上对倒排索引表进行分割,而是增加一个处理器分配表,整个倒排索引表则被多个处理器共享使用;而物理倒排表分割不仅将数据集分割,且同时也对倒排索引表进行物理分割,每个数据子集拥有自己独立的索引倒排结构,因此,它可以供不同的处理器单独调用,相对比较灵活[32].但是,由于独立的倒排表没有全局的统计信息(在进行检索时,通常需要全局的统计信息来计算文档和查询的匹配相似度),所以对每个子集进行检索时,必须要有另外的处理来获得全局信息.方法通常有两种:一种是采用复制的方法,即将全局信息复制多份分配到每个独立的索引倒排表上去;另一种是在每个子集合检索时分两步走,第 1 步获取全局信息表,第 2 步才进行检索.

前一种方法比较耗费空间,对索引表的更新也比较复杂;后一种方法需要较大的通信开销.总的来说,逻辑倒排表分割方法的通信开销很小,因此总体性能会高于物理倒排表分割,但是必须要对普通倒排表增加额外的数据结构来进行转换.而物理分割方法的灵活性比较强,每个文档子集可以单独检索[32].通过物理分割的方法,很容易将非并行的信息检索系统转换为并行的检索系统.

除了倒排表以外,信息检索中常用的其他数据结构(如签名文件 signature file、后缀队列 suffix array 等)在应用到并行检索时也需要进行改变.有兴趣的读者可以参考文献[36].

本文采用物理倒排表分割来实现大规模云工作流模型库的并行检索.下面将详细介绍基于数据集分割的流程并行检索过程.

3 基于数据集分割的流程并行检索过程

3.1 流程模型子集划分

基于数据集分割的流程并行检索算法实现的一个关键前提就是如何对大规模流程模型集进行合理的子集划分.假定有一个大规模流程模型集 R,为了应用并行检索算法,需要将 R 划分成 n 个子集:$R_1,R_2,\ldots,R_n$.一个合理的子集划分原则将同时满足以下 3 点.

1) 完整性条件:$\forall r\in R,\exists R_i$ 满足 $r\in R_i,i=1,2,\ldots,n$;

2) 可重构条件:$R=R_1\cup R_2\cup\ldots\cup R_n$;

3) 不相交条件:$R_i\cap R_j=\varnothing,i=1,2,\ldots,n,j=1,2,\ldots,n$ 且 $i\neq j$.

应用以上模型子集的合理划分原则,可以将大规模云工作流模型库中的流程模型集合理划分为 n 个模型

© 中国科学院软件研究所 http://www.jos.org.cn

子集的.对于一个大规模流程模型集 R,其模型集个数为$|R|$.假定要将 R 合理划分为 n 个模型子集,可以采用均匀划分法或者自动聚类法.

1) 均匀划分法

当采用均匀划分法时,可以根据模型 ID 依次创建 n 个模型子集,每一子集的模型数约为$|R|/n$.很显然,根据模型 ID 的均匀划分法得到的模型子集满足模型集的合理划分原则.本文中的云工作流模型数据存储在关系数据库中,可以通过创建视图的方法对流程模型数据集进行合理的子集划分.例如,将一个有 100 000 个模型的流程集划分为 10 个模型子集,可根据流程模型编号(假定模型编号是从 1 开始依次递增)选取流程子集,将模型编号在 1~10 000 间的流程子集创建为视图 1,模型编号在 10 001~20 000 间的流程子集创建为视图 2,…,依次类推,共创建 10 个视图.这 10 个视图就是对当前流程集进行合理划分的 10 个子集.

2) 自动聚类法

当采用自动聚类法时,应用基于流程相似度的模型聚类算法,根据设定的模型子集数 n 对大规模流程模型集进行聚类;然后,根据聚类结果将大规模流程模型集对应的索引表(如复合节点表、标签索引表)[3]分别分解成 n 个与模型子集相应的索引子表,并通过模型子集的物理倒排表来建立模型子集与其中心点模型及索引子表的关系,该物理倒排表数据结构如图 4 所示,而应用自动聚类方法对大规模流程模型集进行子集划分的具体实现过程如算法 1 所示.

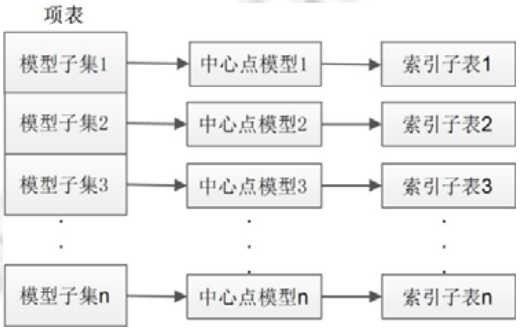

Fig.4 Diagram of data structure of physical inversion table

图 4 物理倒排表数据结构示意图

算法 1. 大规模流程模型集自动聚类算法.

输入:*R*; //模型集

V; //模型索引表

n; //模型子集数

输出:*RS*; //模型子集对象数组

RVS. //模型子集的物理倒排表

//基于流程相似度得到模型集 R 中各流程间的距离(流程间的距离计算)

```
1  for each p1 in R do
2  {   for each p2 in R do
3      {   ps=GetProcessSim(p1,p2);
4          add (p1,p2,1−ps) to ProcessDistanceSet
5      }
6  }
```

//应用 *k*-medoids 聚类算法根据设定的模型子集数对模型集 R 进行聚类(模型聚类)

```
7  RS=ProcessClustering(R,ProcessDistanceSet,n);
```

//根据聚类结果得到模型子集的物理倒排表(物理倒排表的生成)

© 中国科学院软件研究所 http://www.jos.org.cn

```
8   for each rs in RS do
9   {   po=GetCenterObject(rs,ProcessDistanceSet);
10      vs=GetIndexSubTable(V,rs)
11      add (po,vs) to RVS;
12  }
13  return RS, RVS;
```

算法 1 主要由 3 个部分组成:流程间的距离计算、模型聚类、物理倒排表的生成.其中,第 1 行~第 6 行用于基于流程相似度(如文献[37]中基于流程结构相似度)计算算法得到模型集 R 中各流程模型间的距离(也即流程间的距离计算);第 7 行应用 k-medoids 聚类算法根据设定的模型子集数对模型集 R 进行聚类.

函数 *ProcessClustering* 的实现过程如下.

```
ProcessClustering(R,D,n)
输入:R;   //模型集
      D;    //模型间的距离集
      n.   //模型子集数
输出:RS.   //模型子集对象数组
1   k=n;
2   M=getCenterPoint(R,k);   //从 R 中任选 k 个模型作为中心(medoids),并存于集合 M
3   while (k!=0)
4   {k=0;
5     RS=null;
6     for each p in M do
7     { C=GetMinDistanceObject(R,D,M,p);   //根据 D,M 将 R 中与模型 p 距离最近的所有模型存于聚类子
                                           集 C 中
8     add C to RS;   //将 C 增加到模型子集 RS
9     po=GetCenterObject(C,D);   //得到聚类子集 C 的新中心对象模型
10    if (po!=p)
11    {k++;
12      delete p from M;   //将 p 从集合 M 中删除
13      add po to M;   //将 po 增加到集合 M 中
14      }
15    }
16  }
17  return RS;
```

此外,算法 1 第 8 行~第 12 行是根据聚类结果得到模型子集的物理倒排表(也即物理倒排表的生成).根据算法 1 易知,通过聚类得到的模型子集满足上述给出的合理子集划分原则的 3 个条件:完整性条件、可重构条件、不相交条件.也就意味着,算法 1 可用于大规模云工作流模型库的子集划分,但由于算法 1 的时间复杂度约为 $O(u^2v^2)$,其中,u 表示模型集中的模型个数,v 表示模型集中每一模型平均节点数,而在大规模流程模型集中,u 的值通常在 100 000 以上,算法 1 将花费大量时间,因此,大规模流程模型集的子集划分都是事先以离线方式完成.

综上所述,不管是采用均匀划分法还是自动聚类法,都可以将超大规模流程模型集合理划分为指定数目(假定为 n)的模型子集.对于采用均匀划分法得到的 n 个模型子集,虽然每一模型子集的数目大致相同,但各模型子集中的各模型差异较大;而采用自动聚类法得到的 n 个模型子集,虽然各模型子集规模不一,但每一模型子集中各模型间的相似性却较大.因此,对于同一查询模型,两种方法得到的模型子集的检索代介也将存在很大区别.

© 中国科学院软件研究所 http://www.jos.org.cn

此外,对于 n 个模型子集,当用 m 个处理器进行并行检索时,n 个模型子集是初始静态还是实时动态的分配给各处理器进行检索,其最终的检索效率也将大不相同.鉴于此,结合文献[3]中改进后的两阶段流程检索算法,基于两种模型划分集及静态、动态两种子集分配方式,文中给出了 4 种流程并行检索算法:基于均匀划分模型集的静态并行检索算法、基于均匀划分模型集的动态并行检索算法、基于自动聚类模型集的静态并行检索算法及基于自动聚类模型集的动态并行检索算法.下面将分别介绍这 4 种算法.

3.2 基于均匀划分模型集的静态并行检索算法

对于通过均匀划分法已合理划分为 n 个模型子集的大规模流程模型集 R 及查询模型 q,假定要具有 m 个处理器的云服务器来提高 R 的基于图结构/行为的检索效率($m \leq n$).当采用静态方式来给各处理器分配检索用的模型子集时,可以将 n 个模型子集均分成 m 组,每一组对应一个处理器.其具体的实现过程如算法 2 所示.

算法 2. 基于均匀划分模型集的静态并行检索算法.

```
输入:RS;  //R 通过均匀划分法得到的模型子集数组
      q;  //查询模型
      n;  //模型子集个数
      m.  //处理器个数
输出:RqS.  //模型库 R 中覆盖 q 的模型集
1  ResultList=∅;  //定义一个全局检索结果集
2  k=0;  //定义一个表示模型子集编号的全局变量
3  MS=GetCPU(m);  //得到具有 m 个处理器的数组
4  d_num=n/m;  //得到 n 除以 m 的整数值
5  mr=n mod m;  //得到 n 除以 m 的余数值
//将 n 个模型子集均匀分配给 m 个处理器对象
6  for each cpu in MS do
7  {   for i=1 to d_num do
8     {cpu.addTaskData(RS[k++]);
9     }
10 }
11 if (mr!=0)
12 {   for i=1 to mr do
13    {   cpu=MS[i];
14        cpu.addTaskData(RS[k++]);
15    }
16 }
17 for each cpu in MS do   //对每一处理器执行查询模型 q 的任务
18 {   cpu.executeTask(QueryProcess(q));
19 }
//返回归总后的查询结果集
20 if (AllCPUTaskIsFinished()=true)
21    RqS=ResultList;
22 return RqS;
```

在算法 2 中:第 1 行是定义一个全局的检索结果集 *ResultList* 用于存储各处理器的检索结果,*ResultList* 的初始值为空;第 2 行是定义一个表示模型子集编号的全局变量 k;第 3 行用于获取具有 m 个处理器的处理器对象数组 *MS*;第 4 行~第 16 行用于将 n 个模型子集均匀分配给 m 个处理器对象;第 17 行~第 19 行用于对每一处理器

© 中国科学院软件研究所 http://www.jos.org.cn

执行检索查询模型 q 的任务:*QueryProcess*,其中,*QueryProcess* 将调用文献[3]中的算法(假定为算法 0,下同)进行流程检索;调用之前,需要在算法 0 中的返回检索结果 R_q 之前加上以下几行代码:

```
if (|Rq|>0)
{    lock(current_cpu);
add Rq to ResultList;
unlock(current_cpu);
}
```

以上代码用于将某一模型子集的非空检索结果集增加到全局检索结果集 *ResultList* 中.最后,第 20 行~第 22 行用于返回归总后的查询结果集.

3.3 基于均匀划分模型集的动态并行检索算法

对于通过均匀划分法已合理划分为 n 个模型子集的大规模流程模型集 R 及查询模型 q,假定要用具有 m 个处理器的云服务器来提高 R 的基于图结构/行为的检索效率($m \leqslant n$).当采用动态方式来给各处理器分配检索用的模型集时,可以先将 m 个模型子集分别分配给 m 个处理器;然后在处理过程中,等任一处理器查询完毕后,再从未检索过的模型子集中依次再分配一个给该处理器,直到所有的模型子集检索完毕,才返回归总后的最终检索结果.其具体的实现过程如算法 3 所示.

算法 3. 基于均匀划分模型集的动态并行检索算法.

```
输入:RS;  //R 通过均匀划分法得到的模型子集数组
      q;   //查询模型
      n;  //模型子集个数
      m.  //处理器个数
输出:RqS.  //模型库 R 中覆盖 q 的模型集
1  ResultList=∅;  //定义全局检索结果集
2  k=0;  //定义表示模型子集编号的全局变量
3  MS=GetCPU(m);  //得到具有 m 个处理器的处理器对象数组
4     for each cpu in MS do  //将 n 个模型子集的前 m 个分别分配给 m 个处理器对象
5     {
6        cpu.addTaskData(RS[k++]);
7        cpu.executeTask(QueryProcess(q));
8     }
9     while (AllCPUTaskIsFinished()!=true && k!=n)
10    {cpu=GetFinishedCPU(MS);
11       if (cpu!=null)
12       {
13          cpu.addTaskData(RS[k++]);
14          cpu.executeTask(QueryProcess(q));
15       }
16 }
//返回归总后的查询结果集
17 if (AllCPUTaskIsFinished()=true)
18    RqS=ResultList;
19 return RqS;
```

在算法 3 中:第 1 行是定义一个全局的检索结果集 *ResultList* 用于存储各处理器的检索结果,*ResultList* 的初

© 中国科学院软件研究所 http://www.jos.org.cn

始值为空;第 2 行是定义一个表示模型子集编号的全局变量 k;第 3 行用于获取具有 m 个处理器的处理器对象数组 MS;第 4 行~第 8 行用于将 n 个模型子集的前 m 个分别分配给 m 个处理器对象,在分配完成后,各处理器对象就开始执行检索查询模型 q 的任务:*QueryProcess*(*QueryProcess* 的实现过程同上);第 9 行~第 16 行用于监听处理器完成状况,等任一处理器查询完毕后,将再分配未被检索过的模型子集给当前空闲处理器进行处理,直到所有的模型子集检索完毕;最后,第 17 行~第 19 行用于返回归总后的查询结果集.

3.4 基于自动聚类模型集的静态并行检索算法

对于通过自动聚类法已合理划分为 n 个模型子集的大规模流程模型集 R 及查询模型 q,假定用具有 m 个处理器的云服务器来提高 R 的基于图结构/行为的检索效率($m \leqslant n$).由于通过基于流程结构相似性或流程行为相似性的聚类方法得到各模型子集的规模并不一致,并且对于同一查询模型,不同的模型子集与其相似的模型数也不相同,因此,同一查询模型在各模型子集中检索代价也存在较大差异,这种差异与模型集的整体特性及规模紧密相关.为了在并行检索过程中尽可能实现多处理器的负载均衡,需要对这种检索代价进行量化.由此,文中给出了针对自动聚类方法得到的模型子集的检索代价预估值的定义,如定义 1 所示.

定义 1(自动聚类模型集的检索代价预估值). 假定大规模流程模型集为 R,通过自动聚类法划分得到其中一个模型子集为 R',R'的中心点模型为 po,查询模型为 q,那么在 R'中查询 q 的检索代价预估值可表示为 $QC(R',q)$,其计算公式如公式(1)所示:

$$QC(R',q)=Sim(po,q)\times\frac{|R'|}{|R|} \tag{1}$$

其中,$|R|$,$|R'|$分别表示 R,R'中的模型个数,$Sim(po,q)$表示中心点模型 po 与查询模型 q 间的相似度值(该相似度值根据聚类时所应用的相似度计算算法来得到).

对于 n 个通过自动聚类法得到的模型子集,当采用静态方式为 m 个处理器分配相应的模型子集时,尽可能将 n 个模型子集总的检索代价进行均匀分配,以保证 m 个处理器的并行检索效率达到最优.为了更清楚地理解自动聚类模型子集的近似最优静态分配的过程,下面给出了模型子集等检索代价分配的形式化描述.

定义 2(自动聚类模型子集的等检索代价分配). 假定大规模流程模型集为 R,通过自动聚类法划分得到 n 个模型子集:$R_1,R_2,\ldots,R_n$($R_1,R_2,\ldots,R_n$ 满足模型子集合理划分的原则:1) $\forall r\in R,\exists R_i$ 满足 $r\in R_i,i=1,2,\ldots,n$;2) $R=R_1\cup R_2\cup\ldots\cup R_n$;3) $R_i\cap R_j=\varnothing,i=1,2,\ldots,n,j=1,2,\ldots,n$ 且 $i\neq j$).同时假定用于并行检索的处理器个数为 m,对于查询模型 q,当采用静态方式给 m 个处理器分配模型子集进行并行检索时,需要将这 n 个模型子集分成 m 组:$RM_1,RM_2,\ldots,RM_m$,其中,$RM_1,RM_2,\ldots,RM_m$ 必须满足:

1) $RM_i=\cup R_j,i=1,2,\ldots,m,j=1,2,\ldots,n$;
2) $R=RM_1\cup RM_2\cup\ldots\cup RM_m$;
3) $RM_i\cap RM_j=\varnothing,i=1,2,\ldots,m,j=1,2,\ldots,m$ 且 $i\neq j$.

对于任一模型子集组 $RM_i,i=1,2,\ldots,m$,它所有子集的总检索代价的预估值可表示为 $QC\_SUM(RM_i,q)$,其计算公式如公式(2)所示:

$$QC\_SUM(RM_i,q)=\sum_{\bigcup R_j\in RM_i}QC(R_j,q),\text{其中},i=1,2,...,m,\ j=1,2,...,n \tag{2}$$

为了提高流程并行检索效率,m 个处理器尽量做到均衡负载,此时,需要将 n 个模型子集按检索代价预估值均分到 m 个处理器中,也即自动聚类模型子集的等检索代价分配需满足公式(3):

$$QC\_SUM(RM_1,q)\approx QC\_SUM(RM_2,q)\approx\ldots\approx QC\_SUM(RM_m,q) \tag{3}$$

根据定义 1 及定义 2,采用启发式搜索算法可以实现自动聚类模型子集的等检索代价分配函数 *GetOptimalGroup*,其具体实现过程如下.

GetOptimalGroup(*RPS*,*m*)

输入:*RPS*; //(*rs*,*qc*)元组的集合,*rs* 表示模型子集对象,*qc* 表示查询模型 q 在模型子集 *rs* 中的检索代价预估值

© 中国科学院软件研究所 http://www.jos.org.cn

m. //处理器数(分组数)
输出:*RSM*. //最近似等检索代价分配的 *m* 组模型子集集合

```
1  qc_sum=GetQueryCostSum(RPS);  //得到各模型子集检索代价预估值的和
2  qc_aver=qc_sum/m;  //根据分组数 m 得到检索代价的均值
3  RPS=SortByDesc(RPS);  //对 RPS 按检索代价预估值进行降序排序
4  k=m;
5  while (k!=0 && RPS!=null)
6  { qc_group=0;
7    RC=null;
8    rps=GetFirstRps(RPS);  //得到 RPS 中第 1 个元组 rps:(rs,qc)
9    rs=GetRS(rps);  //得到 rps 中模型子集 rs
10   qc=GetQC(rps);  //得到 rps 中模型子集 rs 的检索代价预估值 qc
11   qc_group=qc_group+qc;
12   add rs to RC;  //将模型子集 rs 增加到一个模型子集分组集合 RC 中
13   delete rps from RPS;  //将 rps 从 RPS 中删除
14   while (qc_group<qc_aver)
15     { rps=GetNextRps(RPS);  //得到 RPS 中下一个元组 rps:(rs,qc)
16       if (rps!=null)
17       { qc=GetQC(rps);
18         if ((qc_group+qc)≤qc_aver||(qc_group+qc−qc_aver)≤qc_θ)  //qc_θ为最接近检索代价均值的
                                                                   距离阈值
19         { rs=GetRS(rps);
20           qc_group=qc_group+qc;
21           add rs to RC;
22           delete rps from RPS;
23         }
24       }
25       else
26       { break;}
27     }
28     add RC to RSM;
29     k−−;
30   }
31   while (RPS!=null)
32   {   rps=GetFirstRps(RPS);  //得到 RPS 中第 1 个元组 rps:(rs,qc)
33     rs=GetRS(rps);  //得到 rps 中模型子集 rs
34     RC=GetMinQC(RSM)  //得到 RSM 中检索代价预估值之和最小的模型子集集
35     delete RC from RSM;
36     add rs to RC;
37     add RC to RSM;
38     delete rps from RPS;
39   }
```

© 中国科学院软件研究所 http://www.jos.org.cn

40 return *RSM*;

综上所述,基于已得到的模型子集对象数组 *RS*、模型子集的物理倒排表 *RVS*,根据自动聚类模型子集的等检索代价分配,下面给出基于自动聚类模型集的静态并行检索的实现过程,如算法 4 所示.

算法 4. 基于自动聚类模型集的静态并行检索算法.

输入:*R*; //大规模流程模型集

RS; //*R* 通过自动聚类法得到模型子集数组

RVS; //模型子集的物理倒排表

q; //目标查询模型

m. //处理器个数

输出:R_qS. //模型库 *R* 中覆盖 *q* 的模型集

```
1  ResultList=∅;  //定义全局检索结果集
2  k=0;  //定义表示模型子集分组编号的全局变量
3  MS=GetCPU(m);  //得到具有 m 个处理器的数组
4  for each rs in RS do  //计算查询模型 q 与各模型子集的检索代价预估值
5  {   po=GetCenterPointProcess(rs,RVS);
6     qc=QC(rs,q);
7     add (rs,qc) to RPS;
8  }
9  RSM=GetOptimalGroup(RPS,m);  //将 n 个模型子集分成检索代价最接近的 m 组模型子集
10 for each cpu in MS do  //将 RSM 中 m 组模型子集分别分配给 m 个处理器
11 { k=k+1;
12 for each rs in RSM[k] do
13 { cpu.addTaskData(rs);}
14 }
15 for each cpu in MS do
16 { cpu.executeTask(QueryProcess(q));
17 } }
//返回归总后的查询结果集
18 if (AllCPUTaskIsFinished()=true )
19    RqS=ResultList;
20 return RqS;
```

在算法 4 中:第 1 行是定义一个全局的检索结果集 *ResultList* 用于存储各处理器的的检索结果,*ResultList* 的初始值为空;第 2 行定义一个全局变量 *k*;第 3 行用于获取一个具有 *m* 个处理器的处理器对象数组;第 4 行~第 8 行用于根据定义 1 计算查询模型 *q* 与各模型子集的检索代价预估值,并存于 *RPS* 集合中;第 9 行是根据 *RPS* 数组中每一模型子集的检索代价预估值,采用等检索代价的模型子集最优分配函数 *GetOptimalGroup*(*RPS*,*m*)将 *n* 个模型子集分成检索代价最接近的 *m* 组模型子集,存于集合 *RSM* 中;第 10 行~第 14 行用于将 *RSM* 中 *m* 组模型子集分别分配给 *m* 个处理器;第 15 行~第 17 行用于在 *m* 个处理器中执行检索任务 *QueryProcess*(*q*)(*QueryProcess* 的实现过程同上).最后,第 18 行~第 20 行用于返回归总后的查询结果集.

3.5 基于自动聚类模型集的动态并行检索算法

对于 *n* 个通过自动聚类法得到的模型子集,当采用动态方式来为 *m* 个处理器分配相应的模型子集时,每次尽可能将检索代价最高的模型子集分配给空闲的处理器,以保证 *m* 个处理器的并行检索效率达到最优.因此,基于已得到的模型子集对象数组 *RS*、模型子集的物理倒排表 *RVS*,根据自动聚类模型集的检索成本预估值,提出

© 中国科学院软件研究所 http://www.jos.org.cn

了基于自动聚类模型集的动态并行检索算法,如算法 5 所示.

算法 5. 基于自动聚类模型集的动态并行检索算法.

```
输入:R;  //大规模流程模型集
      RS;  //R 通过自动聚类法得到的模型子集对象数组
      RVS;  //模型子集的物理倒排表
      q;  //查询模型
      m.  //处理器个数
输出:RqS.  //模型库 R 中覆盖 q 的模型集
1  ResultList=∅;  //定义一个全局检索结果集
2  k=0;  //定义一个表示模型子集计数的全局变量
3  MS=GetCPU(m);  //得到具有 m 个处理器的处理器对象数组
4  for each rs in RS do  //计算查询模型 q 与各模型子集的检索代价预估值
5  {   po=GetCenterPointProcess(rs,RVS);
6      qc=QC(rs,po,q);
7      add (rs,qc) to RPS;
8  }
9  for each cpu in MS do  //将 m 个模型子集分别分配给 m 个处理器,并执行检索任务
10   {   rs=GetMaxQC(RS,RPS);
11     cpu.addTaskData(rs);
12     cpu.executeTask(QueryProcess(q));
13     k=k+1;
14   }
15 while (AllCPUTaskIsFinished()!=true && k!=n)
16 {   cpu=GetFinishedCPU(MS);
17   if (cpu!=null)
18   {   rs=GetMaxQC(RS,RPS);
19     cpu.addTaskData(rs);
20     cpu.executeTask(QueryProcess(q));
21     k=k+1;
22   }
23 }
   //返回归总后的查询结果集
24 if (AllCPUTaskIsFinished()=true )
25   RqS=ResultList;
26 return RqS;
```

在算法 5 中:第 1 行是定义一个全局的检索结果集 *ResultList* 用于存储各处理器的的检索结果,*ResultList* 的初始值为空;第 2 行定义一个全局变量 *k*;第 3 行用于获取一个具有 *m* 个处理器的处理器对象数组;第 4 行~第 8 行用于根据定义 1 计算查询模型 *q* 与各模型子集的检索代价预估值,并存于 *RPS* 集合中;第 9 行~第 14 行是根据 *RPS* 数组中模型子集的检索代价预估值,按从大到小的顺序将前面 *m* 个模型子集分别分配给 *m* 个处理器,并在分配完毕的同时,执行检索任务:*QueryProcess*(*q*)(*QueryProcess* 的实现过程同上);第 15 行~第 23 行用于监听处理器执行状况,并为每一空闲处理器分配一个没有被检索过的具有最大检索代价的模型子集,直到所有模型子集都检索完毕;最后,第 24 行~第 26 行用于返回归总后的查询结果集.

© 中国科学院软件研究所 http://www.jos.org.cn

综上所述,对于上述 4 个并行检索算法,最希望达到的是:n 个模型子集的检索任务量在 m 个处理器间平均分配,又不会为每个处理器引入额外的工作量.如果能成功达到该目标,在 m 个处理器进行并行检索的查询效率就是在单机(单处理器)环境下相应检索算法的 m 倍.但事实上不可能达到该目标,因为在多处理器的并行检索中会引入一些其他代价,如模型子集(静态/动态)分配给各处理器的时间消耗、各子处理器与主程序(主处理器)的通信开销等.这些代价必然削减并行检索算法的查询效率.但不管怎样,当采用一定数目的处理器进行并行检索时,可以显著提高大规模流程模型集的检索效率.下面将给出相关评估与验证实验.

4 实验与评估

为了验证和评估本文的方法,我们基于陶瓷云科技服务平台框架研发了一个云工作流管理平台(cloud workflow management platform,简称 CWMP),其访问网址是http://platform.pasp.cn,通过陶瓷云科技服务平台的统一登录界面可以进入云工作流管理平台的操作界面.该平台主要包括 5 个工具:模型自动生成工具、模型导入与转换工具、模型可视化建模工具、模型查询工具及结果显示工具.模型自动生成工具用于自动生成满足实验需求的模拟流程模型.模拟流程的所有节点标签都是通过预先设定的方式从 WordNet 词库自动选取;每一节点的类型、汇合属性及分支属性也是按一定的规则随机分配的.模型自动生成工具中,这些用于生成控制流的规则都基本来源于文献[38].模型导入与转换工具用来导入其他格式(如 Petri-net,YWAL,BPMN,EPC 等[39,40])的模型,并转换成文献[3]中 CNet 模型.所有的 CNet 模型可以通过模型可视化显示工具(http://cbpm.pasp.cn)进行展示;最后的实验结果及结果分析可以过结果显示工具进行查看.

为了验证和评估算法 2(用 PQ-DESA 表示)、算法 3(用 PQ-DEDA 表示)、算法 4(用 PQ-DCSA 表示)、算法 5(用 PQ-DCDA 表示)的效率,文中以文献[3]中的算法(假定为算法 0,用 TS-QTCS 表示)为比较基准,在云工作流管理平台中,基于模拟生成的大规模流程模型库及真实的云工作流模型库做了大量验证实验.在这些实验中,所有模型子集的索引,如复合节点索引及标签索引,都是通过 Apache Lucene(the Web site of apache lucene, http://lucene.apache.org)进行管理和维护;另外,实验用服务器共 5 台,型号为曙光 A840-G20.每一服务器都采用服务器专用芯片组,配有 4 颗 AMD Opteron6320 CPU(2.8GHz,8 核,16M 三级缓存),256GB 内存,并集成 Intel 四口千兆网卡.

4.1 合成数据集上的实验验证

为了分析算法 2~算法 5 的查询响应时间与模型集大小之间的关系,并比较这 4 种算法与算法 0 的查询效率,基于模型自动生成工具生成的 6 个大规模流程模型集(它们的模型数分别 100 000~600 000)做了一些评估实验(实验中设定模型子集数 n 为 30,处理器个数 m 为 10),其实验结果如图 5 所示.

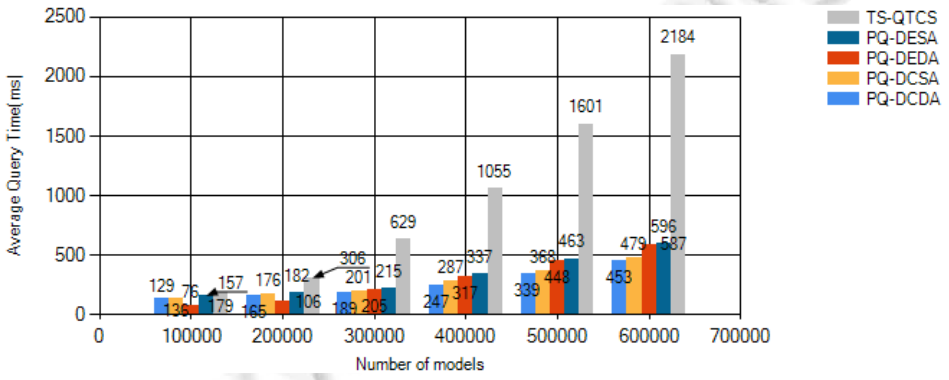

Fig.5 Comparion of query efficiency with the Algorithm 0 in the composite data set

图 5 在合成数据集上与算法 0 的查询效率比较

© 中国科学院软件研究所 http://www.jos.org.cn

在图 5 中,随流程模型集规模的增大,算法 2(PQ-DESA)、算法 3(PQ-DEDA)、算法 4(PQ-DCSA)、算法 5(PQ-DCDA)及算法 0(TS-QTCS)都要花费更多查询响应时间.同时,对于同一大小的模型集,算法 2~算法 5 的平均查询响应时间明显少于算法 0 的平均查询响应时间,并且随流程模型数量的增多,算法 2~算法 5 将节省更多的查询响应时间.

根据上述实验结果可以得出,在大规模流程模型库中进行流程检索时,应用基于数据集分割的并行检索方法能进一步提高流程检索效率.

此外,为了量化上述 4 个并行检索算法(算法 2~算法 5)与单机(单处理器)环境下相应检索算法(也称串行检索算法,如算法 0)查询效率的比较结果,并且对比这 4 个并行检索算法的效率.下面给出一个大规模流程模型集的并行检索加速比的概念,如定义 3 所示:

定义 3(并行检索加速比). 假定大规模流程模型集为 R,对于同一查询请求,用 T_{pq} 表示采用并行检索算法 PA 在 R 中的查询响应时间,用 T_{sq} 表示采用串行检索算法 SA 在 R 中的查询响应时间.那么与串行检索算法 SA 相比,并行检索算法 PA 在 R 中的并行检索加速比可表示为 $PQS(PA,SA)$,其计算公式如公式(4)所示.

$$PQS(PA,SA)=T_{sq}/T_{pq} \tag{4}$$

对于定义 3,在理想状态(不考滤并行检索中模型子集分配、处理器间的通信开销及检索结果归并等代价)下,用 m 个处理器进行并行检索时,可以认为 $T_{pq}=T_{sq}/m$,此时,根据公式(4)可以得到 $PQS(PA,SA)=m$,称为并行检索的线性加速比,也即理想加速比.但在实际的流程并行检索中,由于 m 个处理器的并行处理引入了其他检索代价(用 T_c 表示其他检索代价的时间消耗),此时,$T_{pq}=T_{sq}/m+T_c$.根据公式(4),可以得到 $PQS(PA,SA)$也可由公式(5)来计算.

$$PQS(PA,SA)=\frac{T_{sq}}{T_{sq}/m+T_c}=\frac{m}{1+m\times(T_c/T_{sq})} \tag{5}$$

在实际的流程并检索中,T_c 不可能等于 0.因此根据公式(5),可以得出基于数据集分割的流程并行检索不可能获得线性加速比.

如图 5 所示,可以得出,与算法 0 相比,算法 2~算法 5 对不同规模流程集的并行检索加速比的值见表 1.

Table 1 Parallel retrieval acceleration ratio of Algorithm 2~Algorithm 5 compared with Algorithm 0

表 1 算法 2~算法 5 与算法 0 的并行检索加速比

模型数	算法 2	算法 3	算法 4	算法 5
100 000	1.14	1.25	1.32	1.39
200 000	1.68	1.79	1.74	1.85
300 000	2.93	3.07	3.13	3.32
400 000	3.13	3.33	3.68	4.27
500 000	3.46	3.57	4.35	4.72
600 000	3.66	3.72	4.56	4.82

根据表 1 所示:随流程模型集规模的增大,与算法 0 相比,算法 2~算法 5 的并行检索加速比也在增加.这是因为根据公式(5),随流程模型集规模的增大,T_{sq} 比 T_c 的增长会相对快些,也即多处理器并行中的其他开销时间相对变少了.因此,并行检索加速比会随流程模型集规模的增大而增加.

此外,为了对比算法 2~算法 5 的检索效率,本文用每一并行算法在不同规模流程模集下与同一串行算法(算法 0)的并行加速比的平均值(也即平均并行检索加速比)来衡量它们之间的检索效率.根据表 1 可以得到:与算法 0 相比,算法 2~算法 5 的平均并行检索加速比见表 2.

Table 2 Average parallel retrievable acceleration ratio

表 2 平均并行检索加速比

串行算法	算法 2	算法 3	算法 4	算法 5
算法 0	2.67	2.79	3.13	3.40

如表 2 所示,与算法 0 相比,4 个并行检索算法的平均加速比从大到小的顺序为:算法 5、算法 4、算法 3、

© 中国科学院软件研究所 http://www.jos.org.cn

算法 2.这表明,在检索效率方面,基于自动聚类模型集的并行检索算法优于基于均匀划模型集的并行检索算法;而应用动态方式进行模型子集分配的并行检索算法优于应用静态方式进行模型子集分配的并行检索算法.

首先,基于均匀划分法模型集的算法在给处理器分配模型子集时,无论是静态分配还是动态分配,虽然都比较容易实现,但由于每一模型子集中各模型差异性较大(也即模型子集的模型特性多样化),对于同一查询模型,规模相同的各模型子集的检索代价也将各不相同;基于均匀划分法模型集的算法在分配时忽略它们之间检索代价的差异,这样难以保证多处理器并行检索的效率达到最优.而基于自动聚类模型集的算法,不管是静态分配还是动态分配,都通过检索代价的预估值(定义 1)来度量这种差异.在以静态方式进行模型子集分配时,按检索代价尽可能均匀分给各处理器;在以动态方式进行模型子集分配时,尽可能将检索代价高的模型子集先分给处理器进行处理,以减少处理器的空闲或等待时间,这样就可以充分发挥多处理器的并行性能,提高流程并行检索的效率.

其次,应用动态方式进行模型子集分配的算法也将明显优于应用静态方式进行模型子集分配的算法.这是因为静态分配方式虽可以减少各子处理器与主处理器(主程序)通信开销,但该方式可能会闲置部分处理器,造成处理器资源浪费,最终的检索时间会受限于任务最重、处理时间最长的处理器执行时间;而动态分配方式虽然会增加各子处理器与主处理器(主程序)通信成本,但当处理器间通信成本远小于大规模流程模型集检索的查询成本时,该方式会充分利用各处理器的并行检索能力,以提高大规模流程模型库检索的效率.

4.2 真实数据集上的实验验证

为了观察在确定的模型子集数下算法 2~算法 5 的查询响应时间与用于查询的处理器数目之间的关系,基于陶瓷云服务平台中的云工作流模型库做了相关比较实验(实验中设定模型子集数 n 为 30,处理器个数 m 分别取 4,6,8,10,12,14),其最后的实验结果如图 6 所示.

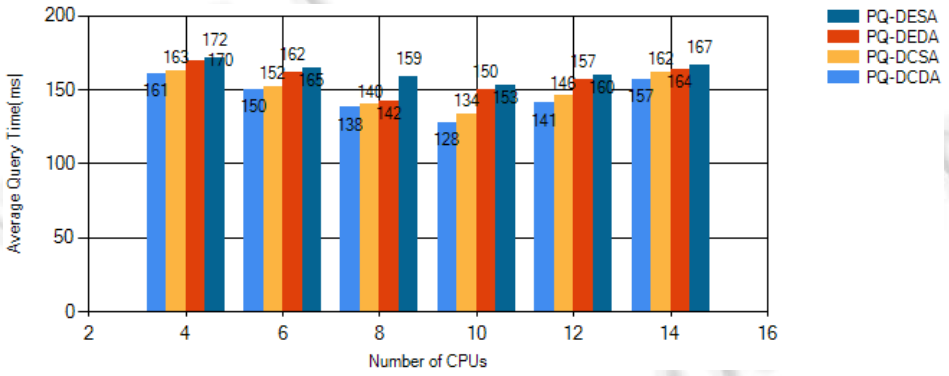

Fig.6 Relationship between the query response time and the number of parallel processors

图 6 查询响应时间与并行处理器数目之间的关系

如图 6 所示,在模型集规模及其合理划分的模型子集数确定的情况下,算法 2~算法 5 的查询响应时间总的趋势都是先随并行处理器的数目增多而减少,当并行处理器数达到一定数目后,再随并行处理器数目的增多而增加.这是因为当处理器达到一定数目后,并行处理器中的其他检索代价将花费越来越多时间,从而造成整个并行算法查询响应时间的增多.对于不同的算法,其查询响应时间达到最小值的处理器数目也并不一致,其中,算法 2、算法 4 及算法 5 的查询响应时间达到最小值的处理器数目比算法 3 要大,这表明算法 3 的查询响应时间对并行处理器数的变化最为敏感,算法 3 随并行处理器数的增多,最先达到查询响应时间的变化拐点.

此外,为了观察在确定的查询处理器数目下算法 2~算法 5 的查询响应时间与模型子集数之间的关系,本文基于陶瓷云服务平台中的云工作流模型库也做了相关比较实验(实验中设定处理器个数 m 为 10,模型子集数 n 分别取 10,20,30,40,50,60),其最后的实验结果如图 7 所示.

© 中国科学院软件研究所 http://www.jos.org.cn

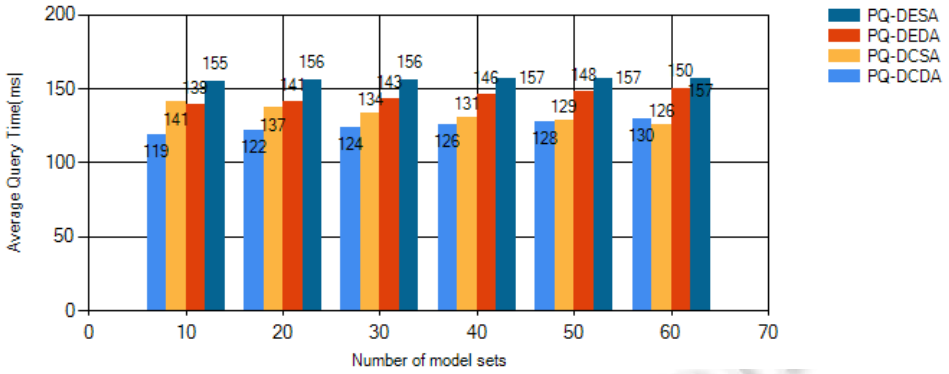

Fig.7 Relationship between the query response time and the number of model subsets

图 7 查询响应时间与模型子集数之间的关系

如图 7 所示,在模型集规模及并行处理器数目确定的情况下,算法 2 的查询响应时间受模型子集数变化的影响较少;算法 3、算法 5 的查询响应时间随模型子集数目增多而增加;而算法 4 的查询响应时间则随模型子集数目增多而减少.这是因为无论模型子集数是多少,算法 2 的模型子集分配结果都是按处理器数进行均分,最终的查询响应时间只受处理器数影响,与模型子集数无关;而算法 3、算法 5 在处理器数确定的情况下,其调度的通信开销与模型子集数紧密相关,模型子集数越多,处理器调度的通信开销越大,因此,算法 3、算法 5 的查询响应时间将会越多;对于算法 4,其检索代价的预估值随模型子集(聚类结果集)的增多将会获得更精确的估算值,也就可以更精确的将整个大规模流程模型集的总检索代价均分到确定数目的处理器上,这样可以使并行检索的性能达到最优化,算法 4 的查询响应时间将会减少.

综上所述,当设定合适数量的模型子集数及查询处理器数,基于数据集分割的流程并行检索算法都能进一步提高大规模云工作流模型库的检索效率.

5 结 论

为了提高大规模云工作流模型库检索的效率,本文提出了一个基于数据集分割的流程并行检索方法.该方法通过均匀划分法或自动聚类法对大规模流程模型库(集)进行模型子集的合理划分,然后应用前期工作中改进后的两阶段流程检索算法,基于两种模型划分集及两种子集分配方式提出了 4 种流程并行检索算法:基于均匀划分模型集的静态并行检索算法、基于均匀划分模型集的动态并行检索算法、基于自动聚类模型集的静态并行检索算法及基于自动聚类模型集的动态并行检索算法.为了评估我们所提出的方法,基于模拟生成的大规模流程模型库及真实的云工作流模型库上做了大量验证实验.实验结果表明,基于数据集分割的流程并行检索方法能进一步提高大规模云工作流模型库的检索效率.此外,在大规模云工作流模型库的检索中,基于自动聚类模型集的并行检索算法比基于均匀划分模型集的并行检索算法具有更好的检索效率;采用动态方式给处理器分配模型子集的并行检索算法比采用静态方式给处理器分配模型子集的并行检索算法具有更好的检索效率.

综上所述,本文提出的基于数据集分割的流程并行检索方法可以应用在大规模云工作流模型库的高效检索中.在下一步工作中,我们将应用 MapReduce 模型实现基于图结构/行为的流程检索算法的并行化,并将其应用在真正的分布式并行计算环境(如 Hadoop 生态系统)中,以进一步提高大规模云工作流模型库的检索效率.

致 谢 感谢清华大学软件学院的王建民老师、闻立杰老师、金涛老师与我分享他们所掌握的实验数据资源.

References:

[1] Baeyens T. BPM in the cloud. In: Proc. of the Int'l Conf. on Business Process Management. LNCS 8094, Berlin, Heidelberg: Springer-Verlag, 2013. 10–16.

© 中国科学院软件研究所 http://www.jos.org.cn

[2] Chai XZ, Cao J. Cloud computing oriented workflow technology. Journal of Chinese Computer Systems, 2012,33(1):90−95 (in Chinese with English abstract).

[3] Huang H, Peng R, Feng ZW. Efficient and exact query of large process model repositories in cloud workflow systems. IEEE Trans. on Services Computing, 2018,11(5):821−832. [doi: http://doi.ieee computersociety.org/10.1109/TSC.2015.2481409]

[4] Song W, Xia X, Jacobsen HA, *et al*. Efficient alignment between event logs and process models. IEEE Trans. on Services Computing, 2017,10(1):136−149.

[5] Song W, Jacobsen HA. Static and dynamic process change. IEEE Trans. on Services Computing, 2018,11(1):215−231. [doi: 10.1109/ TSC.2016.2536025]

[6] Zhang ZW, Cui GM, Zhao S. IFOA4WSC: A quick and effective algorithm for QoS-aware service composition. Int'l Journal of Web and Grid Services, 2016,12(1):81−108.

[7] Song W, Jacobsen HA, Ye C, *et al*. Process discovery from dependence-complete event logs. IEEE Trans. on Services Computing, 2016,9(5):714−727.

[8] Zhang ZW, Cui GM, Deng SG, He Q. Alliance-Aware service composition based on quotient space. In: Proc. of the IEEE Int'l Conf. on Web Services (ICWS). San Francisco, 2016. 340−347.

[9] ter Hofstede AHM, Ouyang C, La Rosa M, Song L, Wang JM, Polyvyanyy A. APQL: A process-model query language. Lecture Notes in Business Information Processing, 2013,159:23−38.

[10] Beeri C, Eyal A, Kamenkovich S, *et al*. Querying business processes with BP-QL. Information Systems, 2008,33(6):477−507.

[11] Sakr S, Awad A. A framework for querying graph-based business process models. In: Proc. of the 19th Int'l Conf. on World Wide Web. ACM Press, 2010. 1297−1300.

[12] Dijkman R, Dumas M, García-Bañuelos L. Graph matching algorithms for business process model similarity search. In: Proc. of the Int'l Conf. on Business Process Management. Springer-Verlag, 2009. 835−842.

[13] Cao B, Yin JW, Chen HR. Levenshtein distance based process retrieval method. Computer Integrated Manufacting System, 2012, 18(8):1766−1773 (in Chinese with English abstract).

[14] Yan Z, Dijkman R, Grefen P. Fast business process similarity search with feature-based similarity estimation. In: Proc. of the Int'l Conf. on the Move to Meaningful Internet Systems. Springer-Verlag, 2010. 60−77.

[15] Qiao M, Akkiraju R, Rembert AJ. Towards efficient business process clustering and retrieval: Combining language modeling and structure matching. In: Proc. of the Int'l Conf. on Business Process Management. Springer-Verlag, 2011. 199−214.

[16] Dumas M, García-Bañuelos L, Dijkman RM. Similarity search of business process models. IEEE Data Eng.Bull, 2009,32(3):23−28.

[17] Kunze M, Weske MM. Metric trees for efficient similarity search in large process model repositories. Lecture Notes in Business Information Processing, 2010,66:535−546.

[18] Jin T, Wang J, Wen L. Efficiently querying business process models with BeehiveZ. In: Proc. of the Demo Track of the 9th Conf. on Business Process Management. 2011.

[19] Yan Z, Dijkman R, Grefen P. FNet: An index for advanced business process-querying. LNCS, 2012,7481:246−261.

[20] Jin T, Wang J, Rosa ML, *et al*. Efficient querying of large process model repositories. Computers in Industry, 2013,64(1):41−49.

[21] Ullmann JR. An algorithm for subgraph isomorphism. Journal of the ACM, 1976,23(23):31−42.

[22] Ryeng NH, Vlachou A, Doulkeridis C, *et al*. Efficient distributed top-*k* query processing with caching. In: Proc. of the Int'l Conf. on Database Systems for Advanced Applications. Springer-Verlag, 2011. 280−295.

[23] Jiang H, Cheng J, Wang D, *et al*. A general framework for efficient continuous multidimensional top-*k* query processing in sensor networks. IEEE Trans. on Parallel & Distributed Systems, 2012,23(9):1668−1680.

[24] Wang X, Shen D, Yu G. Uncertain top-*k* query processing in distributed environments. In: Proc. of the Distributed & Parallel Databases, 2015. 1−23.

[25] Wang B, Zhang G, Sun J. Large-Scale distributed parallel information retrieval technology. Information Technology Express, 2005, (2):1−9. (in Chinese with English abstract).

[26] Peng D. Parallel information retrieval and correlative technology. Research and Exploration of Education in China, 2007,(1):84−86 (in Chinese with English abstract).

[27] Chen GL, Sun GZ, Xu Y, Lu M. Methodology of research on parallel algorithm. Chinese Joural of Computers, 2008,31(9): 1493−1502 (in Chinese with English abstract).

[28] Liu Y, Chen L, Jing N, Liu L. Parallel top-*k* spatial join query processing on massive spatial data. Joural of Computer Research and Development, 2011,48(S3):163−172 (in Chinese with English abstract).

© 中国科学院软件研究所 http://www.jos.org.cn

[29] Wu C. Parallel algorithm and optimization of top-*k* problems in information retrieval [Ph.D. Thesis]. Hefei: University of Science and Technology of China, 2011 (in Chinese with English abstract).

[30] Qi M. Spatial data retrieval and optimization research on shared memory parallel system [Ph.D. Thesis]. Hefei: University of Science and Technology of China, 2012 (in Chinese with English abstract).

[31] Zhu J. Large scale distributed parallel information retrieval technology. Computer CD Software and Application, 2013,(21): 305−306 (in Chinese with English abstract).

[32] Ge FJ. The study of text index construction for large-scale dynamic collection [MS. Thesis]. Harbin: Harbin Institute of Technology, 2008 (in Chinese with English abstract).

[33] Kaur S, Kaur U. A survey on various clustering techniques with *K*-means clustering algorithm in detail. Int'l Journal of Computer Science & Mobile Computing, 2013,2(4):155−159.

[34] Lv Z, *et al*. Parallel *k*-means clustering of remote sensing images based on MapReduce. In: Proc. of the Web Information Systems and Mining. 2010. 162−170.

[35] Surve AR, Paddune NS. A survey on hadoop assisted *K*-means clustering of hefty volume images. Int'l Journal on Computer Science & Engineering, 2014,6(3):113−117.

[36] Singhal A. Modern information retrieval: A brief overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2009,24(24):35−43.

[37] Dongen BV, Dijkman R, Mendling J. Measuring similarity between business process models. In: Proc. of the 20th Int'l Conf. on Advanced Information Systems Engineering. 2008. 405−420.

[38] Wynn MT, Verbeek HMW, Aalst WMPVD, *et al*. Reduction rules for YAWL workflows with cancellation regions and OR-joins. Information & Software Technology, 2009,51(6):1010−1020.

[39] Mendling J, Reijers HA, van der Aalst WMPVD. Seven process modeling guide-lines (7PMG). Information & Software Technology, 2010,52(2):127−136.

[40] van Dongen BF, de Medeiros AKA, Verbeek HMW, *et al*. The ProM framework: A new era in process mining tool support. Lecture Notes in Computer Science, 2005,3536:444−454.

附中文参考文献:

[2] 柴学智,曹健.面向云计算的工作流技术.小型微型计算机系统,2012,33(1):90−95.

[13] 曹斌,尹建伟,陈慧蕊.基于 Levenshtein 距离的流程检索方法.计算机集成制造系统,2012,18(8):1766−1773.

[25] 王斌,张刚,孙健.大规模分布式并行信息检索技术.信息技术快报,2005,(2):1−9.

[26] 彭达.并行信息检索及其相关技术.中国教育科研与探索, 2007,(1):84−86.

[27] 陈国良,孙广中,徐云,等.并行算法研究方法学.计算机学报,2008,31(9):1493−1502.

[28] 刘义,陈荦,景宁,等.海量空间数据的并行 Top-*k* 连接查询.计算机研究与发展,2011,48(S3):163−172.

[29] 吴超.信息检索中 top-*k* 问题的并行算法及优化研究[博士学位论文].合肥:中国科学技术大学,2011.

[30] 齐鸣.共享内存并行系统上空间数据检索及优化研究[博士学位论文].合肥:中国科学技术大学,2012.

[31] 朱江.大规模分布式并行信息检索技术.计算机光盘软件与应用,2013,(21):305−306.

[32] 葛付江.面向动态文档集的大规模文本索引构建技术的研究[硕士学位论文].哈尔滨:哈尔滨工业大学,2008.

黄华(1981－),男,湖南衡阳人,博士,副教授,CCF 专业会员,主要研究领域为服务计算,业务过程管理.

冯在文(1980－),男,博士,讲师,CCF 专业会员,主要研究领域为业务过程管理.

彭蓉(1975－),女,博士,教授,博士生导师,CCF 专业会员,主要研究领域为需求工程,软件工程,服务计算.

© 中国科学院软件研究所 http://www.jos.org.cn