计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
51(Suppl.): 155-160, 2014

SpecGraph:基于并发更新的分布式实时图计算模型

景年强　薛继龙　曲　直　杨　智　代亚非
(北京大学信息科学技术学院计算机科学与技术系　北京　100871)
(jnq@net.pku.edu.cn)

SpecGraph: A Distributed Graph Processing System for Dynamic Result Based on Concurrent Speculative Execution

Jing Nianqiang, Xue Jilong, Qu Zhi, Yang Zhi, and Dai Yafei
(*Department of Computer Science and Technology, School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871)

Abstract　With the rapid development of Internet, more and more related applications need to do analysis and computations on large-scale graph data sets. Facing dynamically changing graph structure, people want to get the real-time results reflecting the latest graph structure. However, traditional graph processing systems mostly aim at static underline graph structure, so they cannot give the dynamic results. This paper proposes a new graph computing model called SpecGraph to meet the timeliness requirement by means of decoupled computing model, asynchronous execution engine and a mechanism of concurrent speculative update. By SpecGraph, we achieve a broader applicability as well as higher real-time performance.

Key words　graph processing system; distributed system; real-time computing; concurrent update; speculative execution

摘　要　随着互联网的快速发展,越来越多的应用需要在大规模图结构数据上作分析和计算,面对动态变化的图结构,人们希望能够实时地得到反映最新图结构的计算结果.传统的图处理系统都是面向静态图结构,不能满足动态图结构的实时性要求.已经提出的增量图计算模型,其算法适用范围受限,而且都是基于串行执行增量更新,当图结构变化比较迅速时,往往结果的实时性不够高.提出了一种新的基于并发更新的图计算模型SpecGraph,它通过解耦合的计算模型、异步执行引擎和基于推测执行的并发更新机制,达到更广的算法适用性和更高的实时性要求.SpecGraph通过解耦合的计算模型,使得顶点状态只依赖于接收到的邻居信息,为增量更新和并发更新提供了透明实现的可能;通过异步计算引擎,使得系统在增量更新时更加灵活,资源占用低,同时保证了并发的可执行性;通过基于推测执行的并发增量更新,SpecGraph能够达到更高的实时性要求.

关键词　图处理系统;分布式系统;实时计算;并发更新;推测执行

中图法分类号　TP393.0

图结构作为一种传统的数据结构,其建模能力强,应用十分广泛.近年来,随着互联网的不断发展,越来越多的信息和数据需要运用图结构进行存储和计算,例如在线社交网络的好友关系、论文库中的引用关系、搜索引擎中记录的网站链接关系以及维基百科中的知识图谱等.各互联网服务的提供商需要在这些图结构的数据基础上进行进一步运算与分析,得到需要的数据,例如网站排名(PageRank[1])、

收稿日期:2014-09-23

用户购买模式、热点信息等. 由于互联网上的这些图结构数据量巨大,单机往往无法运算,需要进行分布式运算.

针对大规模分布式图计算的需求,出现了以Pregel[2]为代表的图计算系统,这些系统能够在静态图结构上进行离线的计算和处理,即每次用户给出图结构和要运行的算法,系统经过运算给出本次处理的结果.

然而,随着广告、社交网络等互联网服务的高速发展,人们对于图计算系统有了更高的要求:1)基于运算的图数据大多并不是静态的,而是动态变化的,例如网站新建了链接,用户取消了关注,这时图结构已经发生了改变;2)朋友推荐、广告投放、热点检测等服务都需要应用最新的数据来计算. 基于以上2方面,人们对图计算系统的需求变为能够在图结构不断变化的情况下,持续给出反映最新数据的算法结果.

在传统图计算系统中,要得到图结构变化后的运算结果,我们只能重新提交图结构及算法,进行一次新的运算任务. 然而由于图结构规模较大,单次的基于图结构的运算需要较长时间,无法达到实时性要求. 针对动态实时图计算系统,目前已经有人提出采用增量更新的方式实现实时计算,如KineoGraph[3],IncGraph[4]等,然而其计算模型都是基于增量消息进行计算,模型表达能力有限,而且更新都是串行执行、实时性有限.

针对动态图结构变化的实时图计算需求,本文在已有研究基础上,提出了一种新的基于并发更新的实时图计算模型,它通过解耦合的计算模型、异步执行引擎,以及基于推测执行的并发增量更新,保证能够在动态变化的图结构上持续提供实时的计算结果. 基于这种新的图计算系统模型,我们设计了基于并发更新的大规模图处理系统SpecGraph.

为了能够保证实时性,SpecGraph采用了以下3个关键技术:

1) 解耦合的计算模型. 在面向顶点的编程模型中,SpecGraph规定顶点的状态只依赖于邻居顶点发来的消息,而需要发送的消息只与顶点状态有关. 这一限定使得实现并发更新的实现变得简单,同时又没有降低模型的表达能力.

2) 异步计算引擎. 在增量更新中,SpecGraph采用异步计算引擎,使增量计算只需局部资源,并且能够在不同顶点执行不同层面的任务,从而为并发更新提供了可能性.

3) 基于推测执行的并发更新. 利用异步计算引擎,SpecGraph采用推测执行,并发进行增量更新,一致性视图和回退机制保证了结果的正确性,通过最大限度地提高并发收益来提高系统计算结果的实时性. 在实际应用中,并发更新对于用户是透明的,用户只需实现一般增量计算的接口函数即可,并发更新完全是由系统根据图结构的变化自动完成的.

1 模型及API

本节将详细描述SpecGraph的计算模型及用户编程接口.

1.1 顶点环境闭包

SpecGarph采用与传统图计算系统相同的基于图顶点的计算模型,以顶点为中心进行编程,每个顶点并行的执行由用户自定义的程序将计算相关的信息保存在节点上,并通过消息的方式进行通信.

系统为顶点在计算的过程中维护一个环境闭包,包括与顶点相关的图结构、与顶点关联的计算信息以及其他顶点发给该顶点的最新消息. 算法的运行过程就是各个顶点闭包更新、通信的过程. 环境闭包具体定义如下:

定义1. 环境闭包 $closure(v)$. 顶点 v 的环境闭包 $closure(v)$,是一个三元组 $(G(v), status, message\_map)$. 其中,$G$ 是与顶点 v 相关的图结构信息,包括节点 v 本身及与 v 相连的所有边 e;状态 $status$ 是与该顶点相关的具体计算数据,随着计算过程不断进行更新;$message\_map$ 是一个从顶点到消息的映射 $\{vertex\ id: last\_message\}$,记录其他顶点最后一次发给该顶点的消息.

环境闭包简记为

$$CLS(v)=(G,S,M).$$

1.2 基本计算接口

SpecGraph提供的接口如图1所示,用户通过实现这些接口来定义自己的算法,接口都是以顶点为单位.

initiate();
初始化顶点环境闭包.

compute(G, *message_map*)—>*status*
通过图结构和 *message_map* 来更新顶点的状态 *status*.

generate(G, *status*)
通过顶点的状态计算需要发送给其他节点的消息.

trigger(*old_status*, *new_status*)
触发器,根据状态变化决定需要继续传播.

图1 SpecGraph基本用户接口

SpecGraph 的顶点计算流程如图 2 所示,顶点 v 在开始计算后首先进行初始化,然后进入消息监听状态,当收到消息时,便按照用户实现的接口函数进行计算,并发送消息,进行迭代.

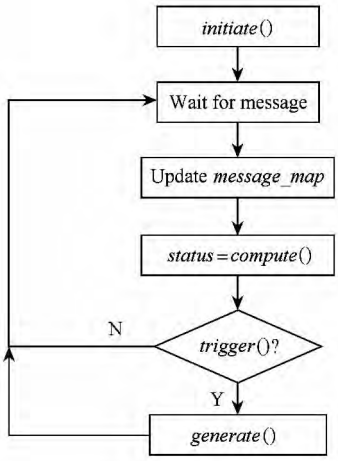

图 2 SpecGraph 的顶点计算流程

SpecGraph 提供的模型与传统的图计算系统有一定区别,传统图计算系统将 *compute* 和 *generate* 接口合到一起,只提供一个 $compute(G, old\_status, message\_map)$ 接口,用户利用旧的状态和最新接收到的消息计算出顶点的新状态,并发送相应的消息,而不关心相应的依赖关系. SpecGraph 提供的这种模型要求计算能够进一步解耦合,即顶点的状态 *status* 只依赖于最新接收到的消息,而与旧状态无关;顶点需要发送的消息则只依赖于顶点当前的状态,而与接收到的消息无关. 用符号表示就是:

$$S=compute(G,M);$$
$$M=generate(G,S).$$

1.3 增量更新

为了能够实现在动态图结构上的实时计算,SpecGraph 采用增量更新的模型,利用图结构的变化信息,在之前的计算结果基础上进行增量计算并进行传播,直到算法收敛.

为了实现增量更新,用户需要实现增量计算接口,按照顶点(边)的增减分别实现相应的增量算法,发送一些调整消息通知相应的顶点重新进行计算,然后系统进入一般计算状态,即各顶点在接收到消息后,根据自身的消息映射 M 更新顶点的状态 S,并通过 *trigger* 决定是否生成并发送更新消息,直到计算收敛.

回调接口描述如下:

on_vertex|edge_add|remove|change()

系统实现增量更新的流程如下:

1) 各计算节点更新图结构,记录图结构变化;
2) 根据图变化回调增量更新接口;
3) 触发相应节点进行更新计算;
4) 根据触发器决定是否传播更新;
5) 全部节点更新结束;
6) 输出新结果.

1.4 应用示例

SpecGraph 的解耦合模型为系统的增量更新及并发更新提供了很大的便利性和可操作性,然而这种模型也使得系统的适用性有了一定局限性,即算法必须能够表达成这种解耦合的模型. 根据我们的实验,常见的图算法以及统计信息都能够在该模型下实现,所以这种解耦合的模型并没有显著降低模型的表达能力. 以下是 3 种图中常用算法在 SpecGraph 接口下的实现示例,算法分别是网页排名 PageRank、单源最短路长度 SSSP(single source shortest path)[5] 以及求弱连通分量划分 WCC(weakly connected component)[6]:

算法 1. PageRank 计算接口实现.

```
compute(g, msg_map):
  /* 自身 rank 值来自邻居 rank 值之和 */
  sum=0;
  for (vid, message: msg_map.entry_set())
    sum+=message.value;
  end for
  return sum.
generate(g, status):
  send msg to nbrs(status|g.get_nbr_num()).
```

算法 2. SSSP 计算接口实现.

```
compute(g, msg_map):
  /* 最短距离取决于邻居的最短距离 */
  min_dist=max int;
  for (vid, msg: msg_map.entry_set())
    min_dist=min(min_dist, msg.value);
  end for
  return min_dist.
generate(g, status):
  send msg To nbrs(status+1).
```

算法 3. WCC 计算接口实现.

```
compute(g, msg_map):
  /* 同一个连通分量中,取最大的 id 作为编号 */
  ccNo=g.get_vertex_id();
  for (vid, msg: msg_map.entry_set())
    ccNo=max(ccNo, msg.value);
```

```
    end for
    return ccNo.
generate(g, status):
    send msg to nbrs(status).
```

算法 4. PageRank 增量接口实现.

```
/* 增加一条边,只需要使源顶点根据新的图结构重新分配 rank 值即可 */
on_edge_add(e):
    G.vertex[e.src].generate().
/* 删除一条边,此时源点对目的顶点的 rank 值贡献为 0,且源点 rank 值需重新分配 */
on_edge_remove(e):
    /* 模仿消息发出者为 src,使得在目的顶点可以更新对应的 message_map 项 */
    m=message(e.src,0);
    send_message(e.target,m);
    /* 源点重新分配 rank 值 */
    G.vertex[e.src].generate().
```

1.5 并发更新

并发更新是 SpecGraph 和其他增量图计算系统的最大区别,也是 SpecGraph 能够提高实时性的关键技术,然而这一特性在应用层是完全透明的,用户在进行接口编程的时候完全不需要考虑并发,只需要按照一般增量接口的实现即可,SpecGraph 会在有图结构变化时自动进行并行的增量更新.

SpecGraph 能够实现对用户透明的并发更新,得益于之前设计的解耦合计算模型.

2 基于推测执行的并发更新机制

本节详细介绍 SpecGraph 并发更新的实现机制.

2.1 异步执行引擎

在传统图计算系统中,大多采用整体同步并行计算模型(bulk synchronous parallel, BSP)[7]的同步计算引擎,即在计算过程中有超步(Superstep)这一概念,各个顶点在更新状态并发送消息完成后,要等待其他所有顶点也完成这一 Superstep 才能进行下一轮计算. 而异步执行引擎是只要当顶点收到了消息就进行状态更新并发送消息,而不管其他顶点状态,异步执行引擎的流程已经在图 2 中介绍过. SpecGraph 采用异步执行引擎,主要是针对增量更新过程,有以下 3 个原因:

1) 异步执行引擎占用资源少,不用像同步引擎一样每个 Superstep 都需要所有计算节点的参与才能完成.

2) 在增量更新过程中,各更新顶点的数据依赖少,在这种情况下异步引擎执行得更快.

3) 在异步引擎下,不同计算节点可以在同一时刻执行不同的任务,这是要实现并发更新的必要条件.

2.2 并发更新的动机和实现原理

每次更新结果必须基于一致性视图,即当我们在执行增量更新任务 $T0$ 的时候,如果 $T1$ 时刻图结构发生了变化,那么不能将这个变化应用到 $T0$ 中,否则得到错误的计算结果;即使通过算法设计使计算不会出错,等到系统计算完新的变化而无法及时更新计算结果,使得之前非实时的结果保持更长时间,所以这时需要重新提交一个新的增量任务 $T1$.

基于以上分析,$T1$ 的计算依赖于 $T0$ 的计算结果,那是否一定要在 $T0$ 更新完成后才能计算 $T1$ 呢,答案是否定的. 事实上,$T1$ 对 $T0$ 的依赖只限于 $T0$ 更新时涉及到的顶点,而对于其他顶点 $T1$ 依赖的是更早之前的计算结果,所以 $T0$ 没有涉及到的顶点 $T1$ 是可以不用等待直接计算的.

但是在没有执行完 $T0$ 和 $T1$ 之前,我们是无法知道它们各自涉及到的顶点,而且这种涉及范围无法预测,因为它跟具体算法相关. 采用推测执行的方法来实现并发:在 $T0$ 执行时,尝试并发的执行 $T1$,对于 $T1$ 涉及的顶点 v,如果已经被 $T0$ 涉及到,$T1$ 需要在其上等待 $T0$ 的完成;如果 $T0$ 尚未涉及 v,可以执行 $T1$ 的更新;如果在之后 $T0$ 更新涉及到 v,那么需要在 v 上对 $T1$ 进行回退,待执行完 $T0$ 后重新执行 $T1$.

由于增量更新涉及的顶点范围相较整个图结构很小,所以 2 次更新之间顶点冲突的概率很小,只要保证冲突处理的代价较小,就可以保证并发更新带来实时性的收益.

SpecGraph 的解耦合模型能够以较小的代价实现这种回退机制,模型中顶点的状态只依赖于其最新的 *message_map*,所以只要在更新的过程中保证面向各个视图的顶点 *message_map* 正确,就能保证运算结果的正确性,这便是 SpecGraph 并发机制实现的基本原理.

2.3 基于推测执行的并发机制设计

并发更新时,对任一顶点 v,SpecGraph 为 v 在每个图结构视图上维护一个版本的顶点环境闭包 $CLS(v)=(G,S,M)$. 初始化时,新环境闭包等于最新完成更新的顶点环境闭包. 对于顶点的每一个版

本,有一个标示位表示该版本是否失效,当版本失效时,通过该版本当前状态计算出已发送消息的顶点,向它们发出失效信息,以使这些顶点从 *message_map* 中去除之前接收到的消息;失效的版本只更新该版本的 *message_map*,而不进行计算接口的调用,即更新状态和发送消息.

以下描述 v 接收到新 *mesaage* 后的流程:

算法 5. 并发更新流程.

version=*message*.版本号;

标示所有大于 v 的环境闭包版本失效;

if 版本 *version* 有效

/* 即 *version* 是当前顶点上最早的更新 */

更新版本 *version* 的 *message_map*;

执行版本 *version* 的 *compute*();

执行 *generate*,发出版本为 *version* 的消息;

else

更新版本 *version* 的 *message_map*.

尽管更新可以并发地执行,但更新结果的提交(*commit*)必须按照任务提交的顺序来进行,在一次更新 *commit* 的时候,我们通过 *message_map* 自下而上(从旧版本到新版本)的传递实现推测执行中的回退机制.

顶点 v 在 *version* 为 i 的更新 *commit* 的时候,需要进行以下操作:

算法 6. 并发任务提交.

1) 对所有 *version*>i 的版本号 k:

更新 $M(k)$中所有消息版本号小于等于 i 的词项,令它等于 $M(i)$的相应词项;

2) 选择当前最小的 *version*,标记为有效,执行其 *compute* 和 *generate*.

3 系统架构

本节详细介绍 SpecGraph 的系统模块、各模块功能以及系统的工作流程.

3.1 系统架构

整个系统主要包括 4 个模块,由存储层、计算节点、视图管理器和任务管理器组成,如图 3 所示. 下面分别说明其功能:

1) 存储模块 Storage

系统的持久化存储层,一般是分布式文件系统,如 Hadoop 分布式文件系统(Hadoop distributed file system, HDFS)等. 存储模块主要负责静态图结构的存储、计算结果的保存等需要持久化的信息.

2) 计算模块 Worker

计算模块包含若干个计算节点 Worker,这些 Worker 组成一个计算集群,是真正执行任务的计算节点. Worker 上的图结构视图及任务执行分别由视图管理器和任务管理器协调管理.

3) 视图管理器 View Controller

对动态的图结构进行一致性视图的管理. 视图管理器接收来自系统输入的动态图结构变化,并将其发往 Worker,期间确保一致性快照,并负责通知 Job Controller 当前视图状态.

4) 任务管理器 Job Controller

负责任务的提交、管理,并监视、协调 Worker 的任务执行状况,控制系统的并发执行.

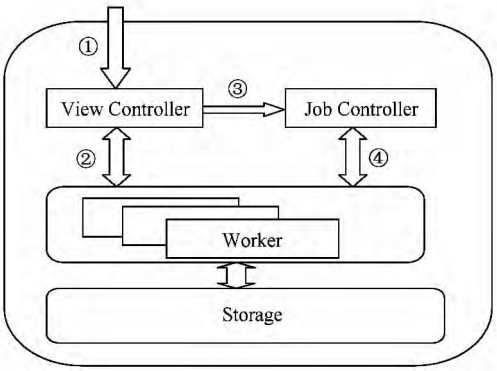

图 3 SpecGraph 系统架构

3.2 系统工作流程

按照图 3 的系统架构图上的流程编号来说明系统在进行更新时的整体流程:

① 从系统外部持续向 View Controller 输入动态图结构的变化信息;

② View Controller 将一段时间内的图结构变化作为一个快照发送至相应的 Worker,形成一个最新版本的图结构视图,当 Worker 中的视图更新好之后通知 View Controller;

③ 当所有 Worker 都更新好最新图结构视图后,View Controller 通知 Job Controller 这一消息,并告知最新视图的版本号;

④ 当 Job Controller 接收到 View Controller 的最新视图版本号后,需要根据当前 Worker 的负载和并发状态决定是否能够提交新的增量更新任务,如果能够提交新的更新任务,则提交从上次执行更新任务的视图版本到最新视图版本的增量更新任务,通知 Worker 回调增量更新接口. Worker 在任务执行过程中与 Job Controller 进行通信,通知当前负载、任务执行状况及并发更新冲突情况,方便 Job Controller 的协调工作.

4 相关工作

为了改进传统批处理框架如 MapReduce[8] 在图模型计算上的低效率,Google 于 2010 年推出了分布式图处理框架 Pregel,它采用基于顶点的语义模型和 BSP 计算模型,将图计算任务划分为若干个 Superstep,在每个 Superstep 中每个顶点进行本地计算,然后发送消息供其他顶点在下一个 Superstep 中计算使用,顶点之间通过 Superstep 的划分进行同步.受 Pregel 的启发,开源系统 Apache Giraph[9], Apache GPS[10] 等都采用类似的计算模型.卡内基梅隆大学(Carnegie Mellon University, CMU)开发的 GraphLab[11] 运用了不同的机制,顶点间通信通过多副本的状态同步而非消息传送进行,并且 GraphLab 有类似异步计算引擎的动态调度器和主要面向图模型的机器学习算法.

5 结束语

本文提出了一种新的基于并发更新的实时图计算模型 SpecGraph,能够实时地给出动态图结构上算法的运行结果,通过解耦合的计算模型、异步执行引擎、基于推测执行的并发增量更新,SpecGraph 能够提供更高的实时性.未来我们将进一步研究视图更新频率及更新的并发度对性能的影响,完善现有模型,对系统作进一步的改进和评测.

参 考 文 献

[1] Page L, Brin S, Motwani R, et al. The pagerank citation ranking: Bringing order to the web. Palo Alto, CA: Stanford InfoLab, 1999

[2] Malewicz G, Austern M H, Bik A J C, et al. Pregel: A system for large-scale graph processing //Proc of the 2010 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2010: 135-146

[3] Cheng R, Hong J, Kyrola A, et al. Kineograph: Taking the pulse of a fast-changing and connected world //Proc of the 7th ACM European Conf on Computer Systems. New York: ACM, 2012: 85-98

[4] 申林, 薛继龙, 曲直, 等. IncGraph: 支持实时计算的大规模增量图处理系统. 计算机科学与探索, 2013, 7(12): 1083-1092

[5] Wikipedia. Shortest path problem //The Free Encyclopedia. [2014-06-10]. http://en.wikipedia.org/w/index.php?title=Shortest_path_problem&oldid=629425850

[6] Wikipedia. Connected component //The Free Encyclopedia. [2014-06-10]. http://en.wikipedia.org/w/index.php?title=Connected_component_(graph_theory)&oldid=612040537

[7] Valiant L G. A bridging model for parallel computation. Communications of the ACM, 1990, 33(8): 103-111

[8] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Communications of the ACM, 2008, 51(1): 107-113

[9] Avery C. Giraph: Large-scale graph processing infrastruction on Hadoop. [2014-06-10]. http://giraph.apache.org

[10] The Apache Software Foundation. Apache Giraph. [2014-06-10]. http://giraph.apache.org

[11] Low Y, Bickson D, Gonzalez J, et al. Distributed GraphLab: A framework for machine learning and data mining in the cloud. Proceedings of the VLDB Endowment, 2012, 5(8): 716-727

景年强 男,1989 年生,硕士研究生,主要研究方向为计算机网络、分布式系统等.

薛继龙 男,1988 年生,博士研究生,主要研究方向为社会网络分析、分布式系统等.

曲 直 男,1989 年生,硕士研究生,主要研究方向为计算机网路、分布式系统等.

杨 智 男,1982 年生,助理教授,主要研究方向为社会网络分析、分布式系统等.

代亚非 女,1958 年生,教授,博士生导师,CCF 高级会员,主要研究方向为社会网络分析、分布式系统等.