**中图分类号：**

**论文编号：**



硕 士 学 位 论 文

**基于运行时间预测的并行图计算系统性能优化研究**

作者姓名 李东泽

学科专业 计算机技术

指导教师 樊文飞 教授

培养院系 计算机学院

**The Research on Performance Optimization of Parallel Graph Computing System based on Runtime Prediction**

A Dissertation Submitted for the Degree of Master

**Candidate：Li Dongze**

**Supervisor：Associate Prof. Fan Wenfei**

School of Computer Science and Engineering

Beihang University, Beijing, China

**中图分类号：**

**论文编号：**

硕 士 学 位 论 文

基于运行时间预测的并行图计算系统性能优化研究

作者姓名 李东泽 申请学位级别 工学硕士

指导教师姓名 樊文飞 职 称： 教授

学科专业 计算技术 研究方向： 分布式图计算

学习时间自 年 月 日 起至 年 月 日 止

论文提交日期 年 月 日 论文答辩日期 年 月 日

学位授予单位 北京航空航天大学 学位授予日期 年 月 日

关于学位论文的独创性声明

本人郑重声明：所呈交的论文是本人在指导教师指导下独立进行研究工作所取得的成果，论文中有关资料和数据是实事求是的。尽我所知，除文中已经加以标注和致谢外，本论文不包含其他人已经发表或撰写的研究成果，也不包含本人或他人为获得北京航空航天大学或其它教育机构的学位或学历证书而使用过的材料。与我一同工作的同志对研究所做的任何贡献均已在论文中作出了明确的说明。

若有不实之处，本人愿意承担相关法律责任。

学位论文作者签名： 日期： 年 月 日

学位论文使用授权书

本人完全同意北京航空航天大学有权使用本学位论文（包括但不限于其印刷版和电子版），使用方式包括但不限于：保留学位论文，按规定向国家有关部门（机构）送交学位论文，以学术交流为目的赠送和交换学位论文，允许学位论文被查阅、借阅和复印，将学位论文的全部或部分内容编入有关数据库进行检索，采用影印、缩印或其他复制手段保存学位论文。

保密学位论文在解密后的使用授权同上。

学位论文作者签名： 日期： 年 月 日

指导教师签名： 日期： 年 月 日

**摘 要**

随着社交网络、生物医学领域的兴起，基于大规模图数据的分析与数据挖掘技术逐渐适用于更多领域，而对大规模图数据处理的计算模型与计算效率的需求一直是各研究领域的热点问题之一。其中同步计算模型与异步计算模型是分布式图处理中经典的两种计算模型，这两种模型在顶点的计算顺序、调度层的结构、以及对计算机硬件和网络资源的利用率方面有着显著的差异，并且它们在不同的数据集、算法、甚至图计算执行过程中不同的阶段均表现出不同的优势，因此本文以结合两种计算模型并尝试达到分布式图并行处理在性能表现上的最优化为目的，提出了基于运行时间预测的自适应计算模型，主要有以下几个方面研究：1) 本文以图计算接收的消息量为切入点，通过对消息量的预测转化成基于对算法运行时间的预测，提出了基于运行时间预测的自适应计算模型，该模型会在图计算执行过程中持续的收集各计算节点的运行时信息，运行机器学习预测技术，对下一轮迭代计算的运行时间进行预测，从而计算得出期望接收的消息量，最终根据期望消息量与已接收消息量的比较，选择性的等待一段时间，目的是期望接收更多的消息量，执行更有质量的迭代计算，从而加速收敛。2) 我们选取了工业界与学术界常用的8个图应用算法，通过分析算法在PIE编程模型下的运行时行为，将算法分为持续活跃型、消息依赖型、和多阶段型三类。针对每一类别给出不同的图计算特征提取方案，并选用多种回归模型进行预测，比较分析预测结果。3) 本文在已有分布式图计算框架GRAPE+基础上进行修改扩展，实现了基于运行时间预测的自适应图计算系统GRAPE++，系统核心包括调度方式的重构、算法运行时信息的采集、以及模型的训练与预测的实现。此外，我们在“天河二号”服务器上对GRAPE++系统进行了详细的性能评测，通过实验结果显示，我们的系统计算性能优于同配置下的同步、异步、和传统自适应计算模型。

**关键词：**分布式图计算，自适应计算模式，算法运行时间预测，分布式系统

**Abstract**

Nowadays, with the diversification of social networking services, the email still maintains a large user base due to its good privacy and popularity.

**Key words:** email classfication, feature evaluation, email thread network, EmailRank

目 录

[摘 要 I](#_Toc25147461)

[**Abstract** II](#_Toc25147462)

[目 录 III](#_Toc25147463)

[图 目 VI](#_Toc25147464)

[表 目 VII](#_Toc25147465)

[第一章 绪论 9](#_Toc25147466)

[1.1 研究背景及意义 9](#_Toc25147467)

[1.2 国内外研究现状 10](#_Toc25147468)

[1.2.1 同步计算模型 11](#_Toc25147469)

[1.2.2 异步计算模型 14](#_Toc25147470)

[1.2.3 混合计算模型 15](#_Toc25147471)

[1.2.4 自适应计算模型 16](#_Toc25147472)

[1.3 研究目标和内容 17](#_Toc25147473)

[1.4 论文组织结构 18](#_Toc25147474)

[第二章 相关理论与技术研究 2](#_Toc25147475)

[2.1 图模型与图应用算法 2](#_Toc25147476)

[2.1.1 图模型 2](#_Toc25147477)

[2.1.2 图应用算法 3](#_Toc25147478)

[2.2 分布式图计算相关技术 5](#_Toc25147479)

[2.2.1 图计算粒度 6](#_Toc25147480)

[2.2.2 消息传递机制 9](#_Toc25147481)

[2.2.3 网络通信方式 10](#_Toc25147482)

[2.2.4 顶层消息优化技术 11](#_Toc25147483)

[2.2.5 图数据划分 11](#_Toc25147484)

[2.3 运行时间预测 12](#_Toc25147485)

[2.3.1 基于数学模型的预测 12](#_Toc25147486)

[2.3.2 基于机器学习技术的预测 12](#_Toc25147487)

[2.3.3 基于相似性的预测 13](#_Toc25147488)

[2.4 GRAPE+系统 13](#_Toc25147489)

[2.4.1 系统架构概况 14](#_Toc25147490)

[2.4.2 PIE编程模型 15](#_Toc25147491)

[2.4.3 图计算执行流程 16](#_Toc25147492)

[2.5 本章小结 16](#_Toc25147493)

[第三章 基于运行时间预测自适应计算模型 17](#_Toc25147494)

[3.1 RPAP计算模型总体结构 17](#_Toc25147495)

[3.2 RPAP计算模型的并行模式 18](#_Toc25147496)

[3.3 RPAP计算模型的消息传递机制 20](#_Toc25147497)

[3.4 RPAP计算模型执行流 21](#_Toc25147498)

[3.5 RPAP计算模型调度实例 25](#_Toc25147499)

[3.6 本章小节 27](#_Toc25147500)

[第四章 运行时间与消息达到速率预测 28](#_Toc25147501)

[4.1 算法分类 28](#_Toc25147502)

[4.1.1 算法的运行时行为分析 28](#_Toc25147503)

[4.1.2 持续活跃型算法 32](#_Toc25147504)

[4.1.3 消息依赖型算法 33](#_Toc25147505)

[4.1.4 多阶段型算法 35](#_Toc25147506)

[4.2 运行时间预测 37](#_Toc25147507)

[4.2.1 问题定义与图计算约束 37](#_Toc25147508)

[4.2.2 特征提取 37](#_Toc25147509)

[4.2.3 模型选取 39](#_Toc25147510)

[4.2.4 运行时间预测 42](#_Toc25147511)

[4.3 消息到达速率预测 44](#_Toc25147512)

[4.4 本章小节 44](#_Toc25147513)

[第五章 基于运行时间预测的图计算系统的设计与实现 45](#_Toc25147514)

[5.1 系统架构概况 45](#_Toc25147515)

[5.2 模块设计与实现 46](#_Toc25147516)

[5.2.1 图存储结构 46](#_Toc25147517)

[5.2.2 消息传递机制 47](#_Toc25147518)

[5.2.3 自适应调度控制 48](#_Toc25147519)

[5.2.4 容错方案与一致性保证 50](#_Toc25147520)

[5.3 系统性能评测 51](#_Toc25147521)

[5.3.1 环境配置 51](#_Toc25147522)

[5.3.2 实验结果 52](#_Toc25147523)

[5.4 本章小节 53](#_Toc25147524)

[总结与展望 54](#_Toc25147525)

[论文总结 54](#_Toc25147526)

[未来工作展望 54](#_Toc25147527)

[参考文献 55](#_Toc25147528)

[攻读硕士学位期间取得的学术成果 59](#_Toc25147529)

[致 谢 60](#_Toc25147530)

图 目

[图 1邮件信息过载问题是一项价值6500亿美金的经济累赘 **错误! 未定义书签。**](#_Toc531617249)

[图 2 ERIC邮件客户端的用户界面 **错误! 未定义书签。**](#_Toc531617250)

[图 3信任网络结构 **错误! 未定义书签。**](#_Toc531617251)

[图 4 EMIRT算法信任分数计算原理图 **错误! 未定义书签。**](#_Toc531617252)

[图 5重要邮件分类方法的架构 **错误! 未定义书签。**](#_Toc531617253)

[图 6邮件会话的两种网络结构 **错误! 未定义书签。**](#_Toc531617254)

[图 7 Webmagic系统架构 **错误! 未定义书签。**](#_Toc531617255)

[图 8 Lucene的架构和全文检索过程 **错误! 未定义书签。**](#_Toc531617256)

[图 9 Spring Data Neo4j的应用架构 **错误! 未定义书签。**](#_Toc531617257)

[图 10不同分类器实验效果对比 **错误! 未定义书签。**](#_Toc531617258)

[图 11基于特征方法与基准方法分类效果对比 **错误! 未定义书签。**](#_Toc531617259)

[图 12邮件的两种引用方式 **错误! 未定义书签。**](#_Toc531617260)

[图 13改进后邮件会话网络构建示意图 **错误! 未定义书签。**](#_Toc531617261)

[图 14邮件关系匹配流程图 **错误! 未定义书签。**](#_Toc531617262)

[图 15 Neo4j浏览器界面中的邮件会话网络 **错误! 未定义书签。**](#_Toc531617263)

[图 16改进和未改进会话网络对比图 **错误! 未定义书签。**](file:///C:\Users\jiang\Desktop\最终论文\毕业论文_蒋坤_SY1606301.docx#_Toc531617264)

[图 17某用户邮件关系网络中的重要节点识别 **错误! 未定义书签。**](#_Toc531617265)

[图 18 EmailRank和基于邮件特征方法实验结果对比 **错误! 未定义书签。**](#_Toc531617266)

[图 19原始Nylas-Mail客户端界面 **错误! 未定义书签。**](#_Toc531617267)

[图 20插件ThreadNetwork效果图 **错误! 未定义书签。**](#_Toc531617268)

[图 21 W3C会员影响力评价系统架构图 **错误! 未定义书签。**](#_Toc531617269)

[图 22分布式爬虫架构图 **错误! 未定义书签。**](#_Toc531617270)

[图 23 W3C会员影响力评价系统截图：影响力排名 **错误! 未定义书签。**](#_Toc531617271)

[图 24 W3C会员影响力评价系统截图：影响力对比 **错误! 未定义书签。**](#_Toc531617272)

表 目

[表 1节点中心度的度量方法 **错误! 未定义书签。**](#_Toc531563451)

[表 2四类邮件特征列表 **错误! 未定义书签。**](#_Toc531563452)

[表 3 SemanticDistance的算法描述 **错误! 未定义书签。**](#_Toc531563453)

[表 4邮件重要性特征重要权值 **错误! 未定义书签。**](#_Toc531563454)

[表 5改进和未改进会话网络下的方法性能对比 **错误! 未定义书签。**](#_Toc531563455)

[表 6 EmailRank和基于网络拓扑方法的实验结果对比 **错误! 未定义书签。**](#_Toc531563456)

# 绪论

## 研究背景及意义

在社交媒体、社交网络和即时通讯飞速发展的如今，随着“互联网+”以及各式互联网应用等新型产品，如抖音、猿辅导的兴起和云计算等新相关技术的飞速发展，人类获取数据的途径和规模正以前所未有的速度快速增长，与此同时，与大数据相关的技术变革与理论发展成为了当今世界的热点话题[1-5]。大数据在环境科学、超级计算、生物学、互联网经济学等领域以及通讯、航天、金融等行业普遍存在，并引起人们的广泛关注。而研究发现，在这些领域或行业中所涉及的数据通常具有某种关联性，从而产生了大图数据[6-10]。图作为计算机数据结构中一种基本且常用的数据类型，是表示实体与实体间关系的方法，一张图由若干顶点和顶点间所连接的边组成，其中根据连接边是否具有方向性又分为有向图与无向图。无论哪种表示方式，其在表达的语义甚至结构方面都比线性表和树更为复杂。同样，社会或科学领域中的许多应用场景都可以通过转化成图计算模型从而得到解决，与图相关的处理和应用几乎无所不在。比如，社会安全部门需要对整个社交网络进行精准的模式匹配[11]，从而增加寻找贩毒、犯罪集群的可能性；交通运输部门需要确定最优的运输路线；生物信息领域需要对蛋白质进行子图匹配，分析蛋白质间的相互作用从而开发出更有效的临床药物[12]。不仅如此，互联网行业中的用户账号匹配、社交网络分析、风险控制模型的训练都可以看作是大图数据的应用。因此，对大图数据的分析与计算具有很重要的现实意义。

早期传统的图数据管理与图分析计算技术通常只针对“小图”进行处理，整个计算过程中不需要过多甚至复杂的迭代过程，更不会产生大量需要信息通讯的消息，时间与空间的开销相对较低。但是，现实世界中实体规模的持续扩大，导致相应图模型的数据规模迅速增长，甚至有数十亿顶点以及上万亿条边的图数据，而这些只靠顶点和边数量组成的图数据结构还只是大图数据的冰山一角。很多应用中涉及的复杂图数据为了表达其现实语义，往往在顶点以及边上附加一定的属性信息，如表示人的节点通常含有姓名、电话、地址等相关属性，这些属性的存储与计算代价通常是点和边的数十倍乃至数百倍。而且，相比于图数据上的搜索查询，大图上的分析计算算法往往需要基于图的结构进行递归迭代操作，直至满足收敛条件或达到预设的最大迭代轮数，因此会产生巨大的通讯数据，并需要频繁的处理中间结果。此外，除了静态数据与离线计算外，很多情况下实时处理情景需要图数据是动态变化的，如淘宝“双十一”实时分析风控数据等，这种动态往往以流式载图作为扩图途径，不断改变着大图数据的规模和结构，因此面对如此大规模的静态和动态图数据，其底层存储、索引建立、统计分析、实时计算等处理的时间开销和空间开销远远超出了早期传统图处理的能力，因此如何解决大图数据的存储技术与查询计算技术，已经成为数据库领域需要迫切解决的问题，也是大数据领域极具挑战的工作。

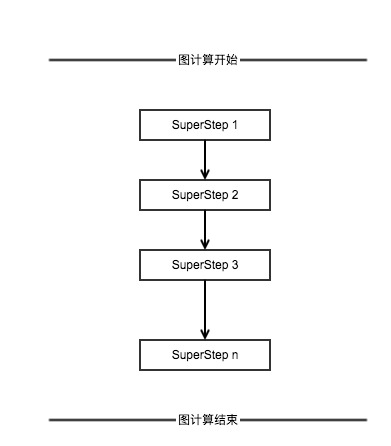
面对上述大图数据存储与计算的诸多问题，研究者们已经不满足于仅通过传统的方法探讨基本的图管理与分析问题，一些新兴的研究热点也在不断涌现，如图查询语言[13-14]、图计算框架等。对于这些研究方向，虽然早期也有一些相关的探讨，但随着计算机软硬件的以及云技术的不断发展，如何提出更准确高效的方法来解决这些问题，为图管理与分析工作带来了新的机遇与挑战。Parallel BGL[15]最为最早一代的并行图计算系统，其在经典的图应用算法基础上开发了并行版本，但此类系统大多数本身不具备一定的稳定性与容错性，并且部分图应用算法的正确性也得不到保证，为了解决稳定性的问题，一部分研究团队专门为特定的图计算应用开发了分布式图计算架构来加速图数据的并行计算，虽然针对性的架构处理可对特定图应用算法进行深度的代码优化以达到更好的处理性能，但毕竟过于局限，针对某一新的图应用算法，算法的移植甚至重开发操作需要重写大量的代码，不仅代码的复用率极低，框架的通用型、可拓展性以及可维护性也很差，对编程人员与使用人员的挑战性极大；MapReduce[16]作为一个并行的计算框架，及其开源实现Hadoop[17]为海量的数据处理提供了便利，为此，用户只需要定义Map和Reduce函数就可完成相应的逻辑计算，不需要关心并发、容错、一致性等复杂问题。但是由于MapReduce框架的自身特点，任务的执行划分为多次Map和Reduce阶段，每个阶段都需要多次的读写分布式文件系统，并且，任务的执行期间无法共享更新数据，再加上图计算的过程通常涉及多轮的迭代，各迭代涉及顶点间的复杂联系，且迭代完成时还要依赖对点的消息传递以满足图计算正确性，因此若将每轮迭代抽象成MapReduce作业不能高效的进行并行图计算，虽然Haloop[18]等处理框架对MapReduce过程进行许多改进，但是并没有从根本上解决基于分布式文件系统所带来的I/O开销问题。针对上述图计算框架的问题，为解决对大规模图数据计算的迫切需要和传统的计算平台无法满足当今的图处理需求的问题，面向大规模并行图计算框架的研究吸引了越来越多的研究者投身其中。

## 国内外研究现状

目前面向大规模图数据计算的分布式并行图计算模型及其相关优化研究取得了不错的结果，依据调度方式的不同，大致分为同步计算模型、异步计算模型、混合计算模型、自适应模型四类。

### 同步计算模型

为改进MapReduce计算模型在大规模图数据上的计算效率低等问题，1990年，图领奖获得者Valiant提出了基于消息传递的BSP(bulk synchronous parallel)计算模型[19]，Google也在2010年推出了基于该计算模型的大规模图数据并行计算框架Pregel[20]，这种计算模型区分于现有的MapReduce计算框架在于其适合做数据的迭代计算，其采用一种“块”同步的思想，即通过消息传递机制实现块间数据同步、块内并行计算。其核心思想是将每一次迭代称为一个超步(Superstep)，整个图计算划分为多个超步，而一个BSP作业由一系列顺序执行的并通过全局同步路障分开的超步组成，并行任务时每个节点按照各自的超步组织，接收来自上一个超步的全局消息，执行本地计算并将本次计算生成的消息发送给下一次超步，计算过程如图1所示，并行迭代直到全部顶点收敛，即每个顶点相邻两次迭代的变化量小于给定的常量或达到预设的最大迭代次数。



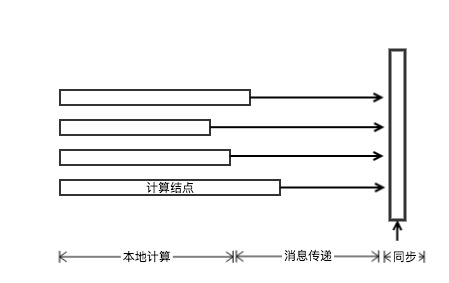
**图1**图 1 **同步模型计算垂直结构**

在每一次的超步计算中，又会分为本地计算、消息传递、全局同步三个阶段，图2和图3分别展示了每个超步过程中的计算过程及同步模型的整体计算流程。

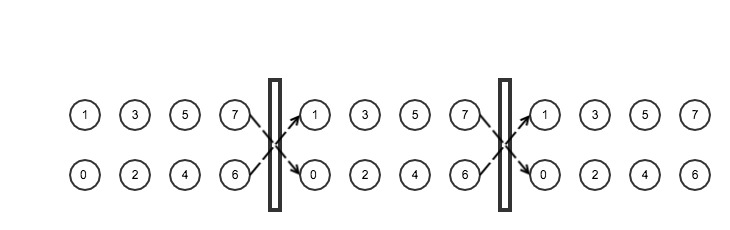
1. 本地计算: 计算节点间相互独立，每个节点负责读取本地内存数据执行并行计算。

2. 消息传递: 本地计算阶段完成后，每个计算节点将生成消息通过通信传递给与之相关联的其他节点，以便交换彼此计算所需要的数据。

3. 全局同步: 用于数据间的整体同步，即块见同步。各个计算节点在完成本地任务后进入全局同步阶段，先完成本地任务的节点需要等待没完成任务的节点，直到最后一个节点完成本地任务，所有节点统一进入下一轮的本地计算。



**图2 同步模型超步内计算过程**



**图3 同步计算模型**

从上图中我们可以清楚的看出，每轮计算中所有顶点只能使用上一轮迭代计算的更新消息，而通过计算得到的最新消息也只能通过同步在下一轮的计算中使用。因为全局同步的存在，使得整个图计算的调度十分简单，同时，每一轮的消息会在全局的同步下批量的传送，极大限度的利用了网络的流量。

为高效的处理大规模图数据的计算问题，在上述BSP计算模型的启发下，Google首次提出了以点为中心的计算框架Pregel[20]。在点中心的计算模型中，用户只需要从点的角度出发，重写一个与具体应用紧密相关的计算函数Compute，而不需要考虑数据的处理过程。在Compute函数内，涉及1) 处理上轮迭代的更新消息; 2) 本地计算逻辑，并更新本节点的值或节点状态信息; 3) 消息传递逻辑。 Pregel的具体执行过程如下: 首先，选定一个图分区策略[21-23]进行图分区操作，划分后的不同分区分配到不同的计算节点上，然后在每一个图节点上执行Compute函数，即用户自定义的超步处理逻辑，每一次的迭代后，通过全局同步操作保证所有计算节点统一进入下一轮的迭代计算，整个迭代过程直到没有任何消息需要同步或所有的图节点达到收敛状态时停止，输出结果。Pregel框架从容错性、一致性、同步控制等方面给出了可行的解决方案，整个计算模型简单、易用、可拓展性强。受Pregel框架的启发，相继涌现出很多基于BSP模型的大规模并行图计算系统，如GRAPE[24], Gemini[25]、GPS[16]、Giraph[27]、Hama[28]、Giraph++[29]、GiraphX[30]、Pregel+[31]、X-Pregel[32]、Mazin[33]、Blogel[34]、GRACE[35]等是对Pregel开源实现的改进与扩展，在性能与功能上不断完善了基于BSP同步计算模型。

GRAPE[24]是以子图为中心基于MPI[36]消息传递的分布式并行图计算引擎，采用独创的PIE(PEval、IncEval、Assemble)编程接口，该接口易于使用并保证了并行计算过程对用户完全透明。用户只需要为特定的图应用算法编写这三个函数便可执行期望的图计算任务；Giraph作为点中心模型遵循了Pregel的机制、基于Hadoop平台的开源分布式图计算引擎，其利用Zookeeper[37]解决了单节点故障问题、采用Netty[38]网络通信的消息传递机制将每轮超步串行化，解决了Hadoop平台本身在图计算过程中低效率问题；GPS类似于Pregel系统，拥有容错机制、扩展性强且易于编程的优势，保证其可在超大规模图数据上执行图计算任务。其于Pregel系统的不同在于1) 通过master.compute函数扩展了Pregel的API，该函数可轻松高效地实现图应用算法；2) GPS具有可选的动态重分区功能，该功能在图计算过程中根据通信状态以决定是否将顶点重新分配给不同的机器，以提高性能；3) GPS拥有LALP的优化功能可减少网络通信量；Hama[]也是基于Hadoop平台于BSP模型下的图计算系统，其专注于解决机器学习和图算法领域的大规模计算任务；Gemini[25]作为一个分布式的图计算系统，其关注计算和通信组件的性能，采用高效的划分，积极压缩计算时间，同时隐藏通信成本，从而达到了在保持高效的同时实现可扩展，弥补了高效共享内存系统可扩展性差和可扩展分布式系统性能差的不足。

相对来说，针对需要多轮迭代处理的图计算任务而言，基于BSP的同步计算模型比MapReduce模型的大数据处理平台具有更大的性能优势，也极大的简化了图计算任务，充分利用了网络流量，但是该模型仍有以下几方面限制：

1. BSP模型具有木桶效应，因为全局同步的存在，每一轮迭代的计算时间取决于最慢的计算节点。因此，节点的计算能力不同或图数据的分区不合理都会导致计算节点间的负载不均衡，严重影响图计算效率。

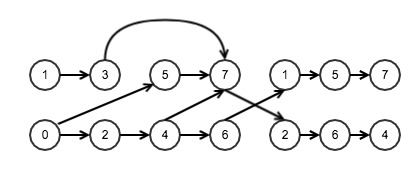
2. BSP模型每轮迭代只能使用上一轮迭代的更新信息，导致迭代次数较多，收敛速度较慢。

3. 消息量大，导致通信代价昂贵。在分布式的图处理框架中，为了满足大规模图数据处理，通常需要将原始的图数据划分为多个子图，并把每个子图分配到不同的计算节点上，因为处于不同计算节点之间的顶点具有一定的关联关系，这就导致不同计算节点间需要进行消息传递以满足图计算的正确性。然而在实际的图数据中，尤其是社交网络等图应用，图中顶点的度数服从幂律分布，从而使得通信的代价成为影响系统性能的瓶颈之一。

4. 某些需要协调图中邻接点的算法无法在BSP模型下实现，例如图着色问题[19],该问题意在使用最少的颜色为图着色，保证相邻的顶点颜色不同。这类问题对于贪心的解决方案在BSP模型下不会收敛，因为图节点试图获得非邻居节点的颜色值，只能通过两者的共同邻居节点来获得，因此同步操作的过程中可能会出现这两个节点颜色不断翻转的情况，本轮迭代具有相同颜色的邻接顶点在下轮的迭代仍会选取相同的颜色，从而导致无法收敛。

### 异步计算模型

为解决上述BSP同步计算模型的诸多问题，科研工作者提出了异步计算模型AP(Asynchronous Parallel)，如图4所示。



**图4 异步计算模型**

异步模型中，相邻两次迭代没有明显的界限，任何计算节点随时接收来自其它节点的消息，并直接接受调度器的调度执行; 同样，本地计算生成的最新消息也随时传递给其它需要的节点。因为异步模型摒弃了全局同步的操作，使得每个计算节点不用关心其它节点的进度、状态，只要消息队列存在消息就可以进行下一轮的本地计算，因此消除了同步计算模型下木桶效应问题。经大量的理论分析和实验表明，异步计算模型在执行效率以及资源的利用率方面相对于同步计算模型具有较大的优势，尤其是在图处理系统负载不均衡时，这种优势体现的更加明显[39,40]。

目前采用异步图计算模型的系统包括GRAPE+[41]、Maiter[42]、PowerGraph[39]、GraphLab[43]、PowerLyra[44]。其中Maiter框架作为完全异步的大规模图数据处理框架，其采用了DAIC计算模型，避免了同步开销，加快收敛速度，极大的提高了大规模图处理效率；GraphLab主要是针对MLDM应用开发的支持同步与异步的图计算系统，其对图计算中数据一致性模型做了详细分析；PowerGraph针对图数据详从工作负载、图分割、存储、通信和计算等多方面带做了详细的分析，提供了完善的图分割数学理论支撑，证明切点法比切边法能提高一个数量级的图计算性能。其采用了以顶点为中心的GAS编程模型，增加了细粒度并发性同时支持同步和异步模型。但它不支持图的动态修改，容错机制未能充分利用顶点副本；PowerLyra从图分割方面对PowerGraph进行了改进。提出了一种混合图分割方法hybrid-cut，即出入度高的顶点采用切点法反之出入度低的顶点采用切边法，经过试验对比性能提高了至少1.24倍。

虽然异步计算模型相比于同步计算模型省去了全局同步所带来的开销，但是也同样存在着以下方面的不足之处：

1. 异步模型可以加速图计算的收敛速度，但由于没有全局同步的限制，导致迭代轮数的增多, 造成了大量的冗余计算。

2. 异步模型在分布式图计算系统的设计方面相对于同步模型来说更加复杂，在设计过程中，不仅要设计正确的调度器，还需要考虑数据的一致性问题，因为异步模型下的任何一个计算节点随时接收来自其它计算节点的消息，因此必须设计一套额外的机制来保证相同数据对象可以被不同的进程和线程互斥访问，例如GraphLab[43]采用分布式锁的形式来保证数据的一致性，而GRAPE+[41]采用了聚合函数Aggr来保证数据的一致性。

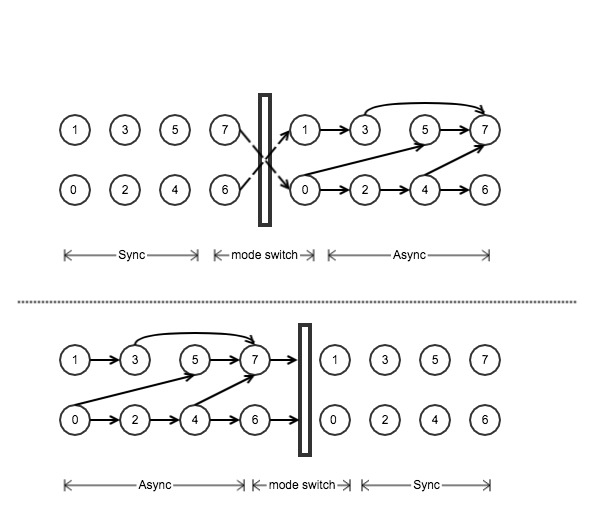
3. 异步模型的计算过程与计算结果具有不确定性，如果算法设计的不合理，迭代计算可能无法达到预期的效果，甚至不能收敛，即使能够收敛，也同样会在迭代计算的最终收敛前，造成大量的冗余计算。不仅如此，对于采用异步模型的分布式图计算系统来说，编程难度有所增加，调试十分困难[45]。

### 混合计算模型

最近研究表明不同计算模型下，图算法的执行效率受到图应用算法、图分区方式、图数据特性与规模、分布式集群规模等多方面因素的影响，不仅如此，在整个迭代周期的不同阶段，可能也需要不同的计算模型以达到全局最优[46]，因此为了解决上述问题，科研工作这提出了混合计算模型。

延迟计算模型[47]作为一种轻量级的混合计算模型，简称SSP模型。其通过引入一个固定的限定值bound来保证最快与最慢的计算节点间相对的迭代轮数不会超过给定的限定值。因此，其允许计算能力强计算节点采用异步的方式执行下去，来减缓同步模型下的木桶效应; 同时通过bound值的限定强制要求计算能力强的节点需等待计算能力差的节点以减少异步模型下的冗余计算。

除此之外，混合计算模型[47]其通过一组启发式的算法建立代价收益模型来动态预测同步模型和异步模型两种调度方式的性能，并实现了在计算过程中对两种计算模型自由的切换，如图5所示。



**图5 混合计算模型**

[46]的实验结果表明，采用了上述混合计算模型的PowerSwitch分布式图计算引擎可以准确预测两种模型的性能，并完成调度模式的切换，相对于使用单一的调度模式，在其上执行了大量的图算法，如PageRank算法、单源最短路径算法、图着色算法等，都在执行效率上得到了不同程度的提高。

### 自适应计算模型

自适应计算模型[41]计算模型，简称AAP(Adapt Aysc Parallel)模型。这种模型摒弃了全局同步的操作，因此从本质上来看其属于异步模型。但与异步计算模型不同的是，每个计算节点在自适应计算模型下均保留一些参数来衡量与其它计算节点的相对执行进度，并根据当前接收到的消息决定是否进入下一轮迭代。上述整个过程只在本计算节点下完成，不用关心其它任何节点的执行状态。该模型不同与其它计算模型主要体现在以下几个方面：

1. 同步计算模型：自适应计算模型通过异步的消息传递机制来消除同步计算模型下木桶效应问题，摒弃了全局同步机制从而消除了这一操作所带了的额外开销。

2. 异步计算模型：自适应计算模型通过引入限定值Delay Stretch使得各个计算节点独立决定每一轮迭代前需等待的时间，目的是接收更多的消息传递量从而达到执行更有效的计算，因此很大程度上减少了异步模型下的冗余计算问题。

3. 延迟计算模型：延迟计算模型下通过预设的限定值bound，使得在bound值允许的范围内，各计算节点采用要异步计算模型进行计算；一旦到到达bound限定的范围，计算快的节点就需等待，直到计算慢的节点完成当前轮数的计算，才可进行下一轮的计算。但自适应计算模型不同于延迟计算模型在于其通过限定值Delay Stretch动态调整各计算节点间的相对进程，当出现计算落后的节点时，其允许该节点等待一段时间以接收更多的消息，这样可以最大化的利用其它计算节点快的优势，使得慢节点的每一轮计算更有效率。

4. 混合计算模型：相对于混合计算模型，自适应计算模型不需要全局的预测开关在同步和异步计算模型进行切换，相反，自适应计算模型允许每个计算节点独立决定各自的计算模式，因此在图算法的执行期间，某些计算节点采用同步计算模式，与此同时，其它节点采用异步模式。除此之外，由于自适应计算模型动态调整计算节点间的相对进程，因此其减少了模型间切换所带来的而外开销。

## 研究目标和内容

在已有的图计算模型中，同步模型存在“木桶效应”问题；异步模型存在大量的冗余计算；混合计算模型为实现同步与异步间自由切换产生了额外的内存与预测开销；而自适应模型中，每个计算节点通过引入限定值动态调整节点间相对进度，从而解决了同步、异步、混合模型下的问题，但目前传统的自适应计算模型只是根据接收到来自多少个节点的消息数来调整各节点相对进度，不具备通用型。因此本文在已有自适应计算模型的原型下，提出了新的基于运行时间预测的自适应图计算模型，该模型可在慢机的情况下规避上述模型的弊端，加速并行图计算。本文的主要研究内容如下：

1. 提出基于运行时间预测的自适应并行图计算模型，以下简称 RPAP模型（RunningTime Predict Async Parallel），该模型通过预测每轮迭代算法的运行时间，选择性的控制各计算节点的相对进度。

2. 我们将对算法运行时间的预测问题定义为机器学习中的回归问题，使用均方相对误差（MSRE）做为损失函数，最终选取机器学习中适合图计算领域的回归模型作为候选，进行训练预测，从而分析各模型在分布式图计算时间预测问题上的优缺点。

3. 我们选取包含单源最短路径[48]、深度优先搜索、网页排序PageRank[49]、节点结构分析HITS[50]、图采样[51]、图连通性、标签传播、社区发现算法Louvain[52]等8种学术届与工业界常用的图计算应用算法，并尝试将算法进行分类，针对每类算法，给出不同的特征提取方案，并展示预测效果。

4. 我们在已有的自适应图计算系统原型下，开发完成基于运行时间预测的自适应并行图计算系统，并与同步、异步、及自适应原型系统进行对比，评估效果。

## 论文组织结构

本文主要分为五个部分。主要的组织结构如下：

第一章为绪论。首先介绍了图计算领域的相关背景，分析了在大图数据兴起的时代下，分布式图计算对处理大图数据的影响与意义。随后介绍了目前分布式计算中常见的4种并行模型，包括同步计算模型、异步计算模型、混合计算模型、和自适应计算模型。其中着重介绍了不同计算模型的优势与劣势。最后在现有研究的基础上，为解决上述模型目前存在的弊端，概括了本文的研究内容，即提出一种基于运行时间预测的自适应计算模型，最后说了了本文的组织结构。

第二章为相关理论与研究技术。首先给出了图数据的定义，并通过对现有图数据的顶点度数统计，分析了图数据分割后的分布不均匀情况。之后介绍了本文实验用到的经典图应用算法；其次，本文从图数据计算粒度、消息传递机制、网络通讯方式、顶层信息优化技术、以及图分割技术5各方面介绍了分布式图计算的相关理论与技术；然后，本章概括了现有算法、程序运行时间预测的技术，分析了各种方法的优缺点；最后，本章介绍了GRAPE++系统，我们的研究工作和系统扩展均基于该系统。

第三章为基于运行时间预测的自适应计算模型。该章首先给出了该计算模型的定义细节与总体结构，其中详细介绍了该计算模型包含的各个模块；接着，该章从并行模式与终止条件、消息传递机制、消息交互方式、以及模型的整体执行流程等方面进一步对该模型进行详细的介绍；最后给出了单源最短路径算法在该模型下的编程细节。

第四章为算法的运行时间和消息到达速率预测。该章首先选用了工业界与学术界经典的8种图应用算法，通过分析算法在PIE编程模型下的运行时行为，将算法进行分类，并针对每类算法，给出了影响图计算运行时间的特征向量提取方案；其次，该章通过机器学习回归技术和“均值预测”思想对上述算法的运行时间和消息到达速率进行预测，并展示了预测结果。

# 

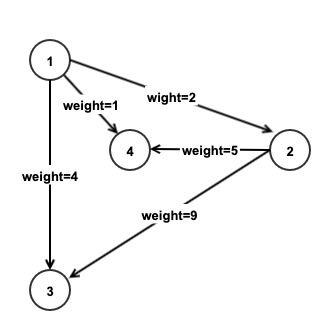
# 相关理论与技术研究

## 图模型与图应用算法

### 图模型

图作为计算机领域重要的数据结构之一，其语义与结构的表达都要比线性表和数结构更加复杂。最基础的图模型被称为简单图，这类图中，仅有顶点与连接顶点的边组成，根据边是否具有指向性又分为有向图与无向图。在图计算领域中，除了需要关注由顶点与边组成的图数据结构外，还关注顶点与边上附加的属性信息，在多数的图应用背景下，这种附加的属性信息更为重要，我们将在下一小节详细介绍本文用到的经典图应用算法，这里给出本文所讨论的简单图模型的基本定义。

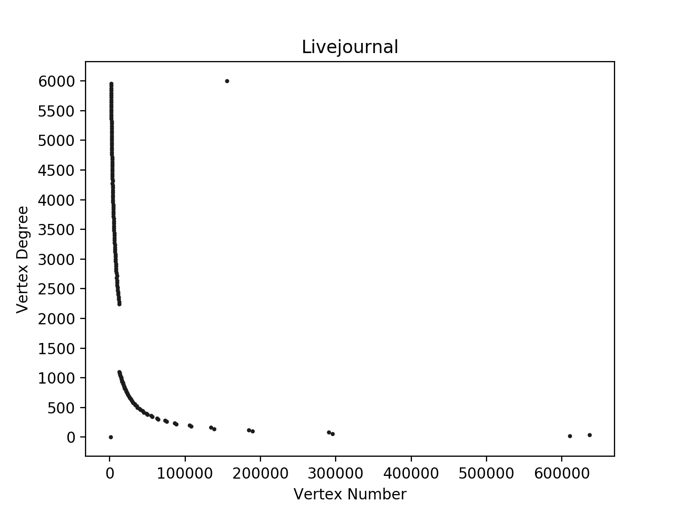
给图，其中代表图中点的有限集合，代表图中边的集合，代表图中顶点或边上的附加属性信息，即，其中表示图中的顶点；，其中表示图中的边，且，表示是连接顶点与的边*；*表示顶点与边上属性的集合。根据边是否具有指向性，图又可以分为有向图与无向图，其中无向图中若存在顶点指向的边，那么一定也存在顶点指向的边，而在有向图中则不成立，图6所示为简单的有向图模型示例：



**图6 简单有向图**

图6中包含4个顶点与5条边，顶点的编号为从1开始的整数序列，图中顶点没有附加属性信息，边上附有单一的属性信息且为整数。

现实世界中，多数图数据的分布极其不均匀，即少数顶点会与周围大量的邻居相连，而多数顶点则与周围邻居交互较少。这类图数据反映到图结构即为少数顶点度数较高，而多数顶点度数较低，造成这一现象的的原因也十分直观，比如社交网络中少数明星会与大量的粉丝产生交互行为，如评论留言；淘宝订单数据中，少数用户的年均订单量明显大于其余用户，以此来鉴别是否用户具有刷单行为。这类分布不均匀的现象对图计算造成极大的挑战，容易出现负载不均衡现象，造成计算效率低甚至由于内存限制导致无法完成图计算。下图7为真实的图数据中顶点度数的分布情况，我们可以从图看出，这个分布不均匀现象十分明显。



**图7 现实图数据顶点度数分布图**

### 图应用算法

图数据处理与图数据的挖掘得到了学者们的广泛研究，不同的实际问题会抽象出不同的图应用算法，除了最短路径、图着色、最小生成树、连通性、可达性等基础图论算法外，还包括社区发现、标签传播、页面排序、子图匹配、图中元素分类与聚类等多种复杂的算法，本小节给出工业界与学术界常用的图应用算法。

**1. 单源最短路径**

单源最短路径问题中，给定一个带权有向图 ，其中代表每条边上的权重且，给定某点做为源点，现要计算从源点到图中其它各顶点的最短路径长度，该长度即为到目标顶点的路径中所走边的权重之和。Dijkstra[53]算法作为解决单源最短路径问题经典的算法之一，其按各顶点与源点间的路径长度的递增次序，生成源点到各顶点的[最短路径](https://baike.baidu.com/item/%E6%9C%80%E7%9F%AD%E8%B7%AF%E5%BE%84/6334920)的算法，既先求出长度最短的一条最短路径，再参照它求出长度次短的一条最短路径，依次类推，直到从源点到其它各顶点的最短路径全部求出为止。

**2. 页面排序**

页面排序对于从超大量的页面文档中筛选出有价值的信息十分重要。PageRank的提出开启了链接分析研究的热潮，基于这种链接分析方法，可以客观的衡量网页质量，并且能够做到独立于内容，不需要人工干预就可以自发的挖掘互联网上有价值的资源。PageRank算法作为最早提出的链接分析算法之一，并在Google公司获得了巨大的商业成功，包含了两个基本假设：一是超文本链接包含用户对一个页面或网站的判断信息；二是某个页面被其他页面链接的数量越多，该网页的重要性越高，越能提供有价值的信息。其基本思想是如果网页存在指向网页的链接，表明网页的所有者任务网页重要，从而把网页的一部分重要性传递给网页，该部分重要的计算由网页的PageRank值与该网页的出链接数共同决定，即，而网页的PageRank值计算公式如下：

其中表示阻尼系数，其意义为用户到达某一页面后，继续沿该页面向后浏览的概率；代表所有链接到网页的页面。上述公式经过多轮迭代，系统的PageRank值逐渐收敛。

另一种对网络图结构做分析的HITS算法，是由康奈尔大学该的Jon Kleinberg博士于1997年首先提出的，该算法与PageRank算法类似，最早用于搜索，现在也常用来做社交网络结构分析。但PageRank算法不同的是，其核心思想是建立在页面链接关系的基础上，对链接结构的改进算法。HITS算法通过两个评价[权值](https://baike.baidu.com/item/%E6%9D%83%E5%80%BC)——内容权威度（Authority）和链接权威度（Hub）来对网页质量进行评估。其中Authority认为如果一个页面提供了关于某个主题的信息，那么这个页面就是有价值的，这样的页面就是Authority页面，因此如果有许多其它的页面都指向该页面，说明此页面的Authority值就高；Hub认为还有一些页面，比如Google页面，虽然它本身不提供任何主题信息，但是从这个页面出发，可以跳转到很多有价值的页面上去，这样的页面就会成为Hub页面，因此如果某个页面指向了很多Authority值高的页面，那么它的Hub值就会高。这种利用页面之间的引用链来挖掘隐含在其中的有用信息（如权威性），具有计算简单且效率高的特点。页面的网页，其链接权威度越高。

**3. 社区发现算法**

互联网络的复杂度对计算机的计算能力提出了严峻的挑战。而随着对复杂互联网络的深入研究，发现这类复杂网络具有较强的社区结构，社区发现就是通过网络中节点间的连接关系挖掘社区结构的过程，因此对复杂互联网络社区发现算法的研究对了解网络的结构具有重要意义。

Louvain算法作为一种基于图数据与模块度优化的社区发现算法，相比于普通的模块度和模块度增益算法，具有明显的性能优势，不仅如此，该算法还能探测到层次的社区结构,不会遗漏一些小型的社区。Louvain算法分为两个阶段，初始阶段网络中每个节点独立为一类，第一阶段的目的是分类，根据划分后模块度的分值将相似的点分为一类，并做好标识；第二阶段将分类后的图重新初始化，即属于同一类别的节点合并为一个节点，继续执行第一阶段，这个迭代过程直到整张图的模块度不在发生变化。

标签传播算法作为另一种高效的处理大规模网络的社区发现算法，由于其近似线性的时间复杂度而收到广泛的关注，其基本思路是根据已标记的节点标签信息来预测该节点周围邻居中未标记的节点标签信息。每个节点标签按相似度传播给周围邻居，在传播的每一步，每个节点根据邻居节点的标签来更新自己的标签，且相似度越大，其邻居节点对其标注的影响权值越大，他们的标签越趋于一致，越容易传播。在标签传播过程中，保持已标记的标签不变，使其将标签传给未标注的数据。最终当迭代结束时，相似节点的概率分布趋于相似，可以划分到一类中。然而，该算法由于每个节点的标签依赖于周围邻居，其聚类的有效性和迭代速度对标签的更新顺序十分敏感，因此一定程度上影响了社区发现结果的稳定性和准确性。

**4. 图采样算法**

一方面，针对社会网络中节点对的采用工作可用于大规模网络的用户兴趣识别；另一方面，随着Graph Embedding等将复杂网络映射到低维空间的机器学习技术的兴起，图采用技术吸引了越来越多的研究者投身其中。

图采样算法大体可分为两类，一类是随机游走，另一类是随机采样。其中随机采样算法的思想是随机的选择图中的点或边，其要求图中的点和边可以被自由的乱序地随机访问，因此该算法并不适用于某些注重结构的图数据；面向随机游走的算法相比于随机采样算法，只需要记录游走过程中所处的当前状态以及当前状态的邻居状态，换句话说，这种算法只需要直到局部的信息，而不以前知道全图信息，因此随机游走算法从这一角度来说更加可行。

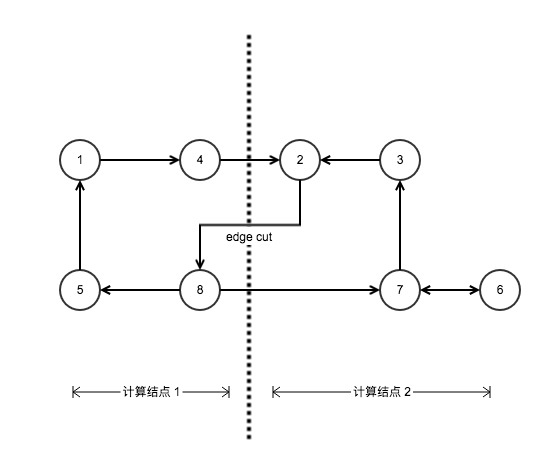
## 分布式图计算相关技术

近些年来，云计算及其在大规模数据处理方面的诸多优势，得到了学术界和产业界的广泛关注，依靠云计算环境对大规模图数据进行高效的处理，被证明非常具有发展潜力。尤其在分布式并行处理方面，利用云计算分布并行处理的特点，可将一个大图分割成若干个子图，从而将对一个大图的处理分割为若干个针对子图的处理，所以能够显著的提高对大规模图数据的处理能力。但是，面向复杂迭代计算任务的大图数据处理是一项系统性的工作，涉及诸多关键性的技术，本节将从图计算粒度、消息传递机制、网络通讯方式、顶层消息优化机制、和图数据划分5个方面进行介绍。

### 图计算粒度

在迭代计算的过程中，根据基本的计算粒度，图数据读取和遍历的模式可分以点为中心、以边或路径为中心、以子图为中心三类。

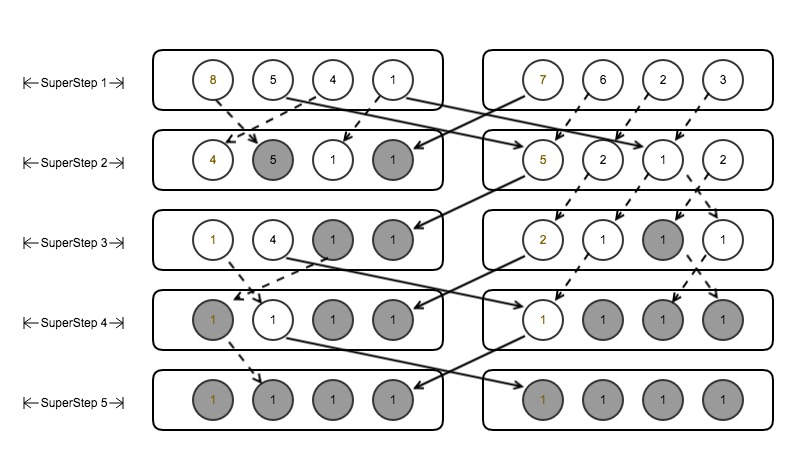
**顶点为中心的计算粒度：**对于以顶点为中心的系统，其迭代计算过程是通过遍历图中顶点完成的。对于每个图顶点，通过处理其收到的消息，完成顶点值的更新，并按照出边向其目的顶点发出新的消息。很多图应用算法可以自然的采用这种方式实现，例如PageRank算法和单源最短路径算法等，由于简单易用，目前绝大多书主流的分布式图计算系统如Pregel[20]、Giraph[27]、Hama[28]等均采用以顶点为中心的计算方式。此外，通常图算法在迭代过程中是逐渐收敛的，即已经收敛的顶点不必重复访问出边以发送消息，如单源最短路径计算问题。Pregel等系统采用voteToHalt机制实现该功能，即为图中每个顶点设置一个标志位，处于激活状态的顶点需要进行更新计算并发送消息，否则直接在遍历过程中跳过该顶点。



**图8 求解联通分量图示例**

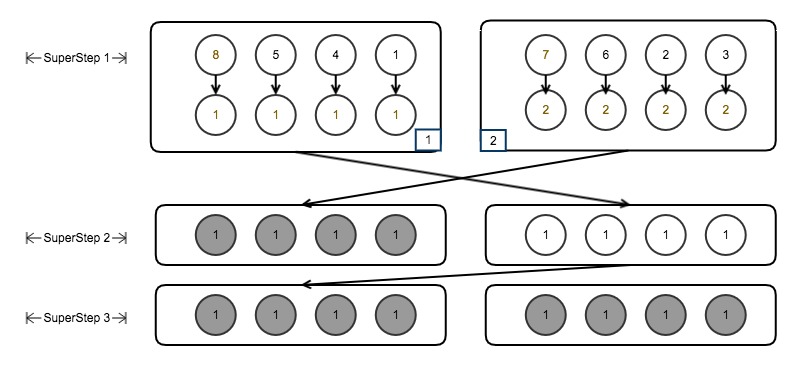
我们以图的连通性问题为例，给定一个无向图，若中的子图满足1）中的任何两个顶点间均存在连通路径2）向中增加任何一个属于的顶点都会使不具有连通性，我们就认为子图是一个连通分量，图8所表示的图只有一个连通分量，但包含两个子图，分配到不同的计算节点，图中每个顶点含有全局唯一的整形值，我们以最小的顶点号表示该顶点所在连通分量，图9中展示了在以定点为中心的计算粒度模式下，求解图连通分量问题的最大迭代过程。

从图中可以清楚的看到，在第一次迭代中，即SuperStep-1，每个顶点向其邻居节点发送自己的value值，如图中的箭头所示，其中虚线表示属于同一子图内顶点间的内部通信，实线则表示跨越计算节点的子图间通信；接下来的每次迭代，每个顶点选取其所收消息的最小值与自己的value值进行比较，如果最小值小于自己的value值，则对自身的value值进行更新，并向邻居借点广播更新后的新值；否则，将自己的状态设置为非活跃状态，即图中的灰色节点，迭代过程一直持续到图中所有的顶点都是非活跃状态且没有任何需要传递的消息，在本例中，共进行5次迭代，8次子图间的消息传递。



**图9 以顶点为中心求解连通性问题**

**以子图为中心计算粒度：**Giraph++[29]、GoFFish[54]、Blogel[34]、GRAPE[24]等部分较新的系统提出采用以子图为中心的计算方式，以加快迭代收敛速度，减少消息传递数量。这种处理方式将图数据划分为若干个子图。在迭代过程中，同一个子图内的顶点首先进行计算，然后子图之间进行消息互换，从而减少子图内部计算产生的冗余消息，加速收敛，减少同步开销。图10展示了以子图为中心的计算粒度在求解联通分量问题的步骤。



**图10 以子图为中心求解连通性问题**

在第1次迭代中，每个子图根据遍历或者其它方式求解出子图内的各连通分量，子图的每个内部点将所在连通分支的最小值设置成自身的value值，处于边界的节点向邻居发送自己的value值，接下来的每次迭代，每个子图选取消息中的最小值与消息对应的连通分量进行比较：如果该最小值小于对应的value值，则更新连通分量内所有的内部点的value值，并通过边界结点传递消息；否则，将自己设置成非活跃状态。迭代持续进行，直到子图内所有节点为非活跃状态且没有任何需要传递的消息，在该例中，共进行了3次迭代域3次跨计算节点间的消息传递。

通过对比图9与图10可以发现，以子图为计算粒度的分布式图计算系统不仅可以有效的减少运行过程中的迭代次数，还可以显著的减轻网络负载，但是相对于以顶点为中心的计算粒度，用户需要针对不同的图算法确定出子图内的边界节点，以便正确的进行消息传递。特别的，Blogel还支持以子图为中心的通讯方式，可以进一步降低消息规模。子图为中心的处理可以有效地减少整个迭代和同步的次数，但大大增加了本地节点内部冗余消息发送的概率，从而增加本地处理的开销。其虽然作为以顶点为中心处理在跨节点消息和同步方式方面的优化，但并不能完全取代以顶点为中心的处理方式，具体的性能与应用迭代情况和图数据的特点相关。

**以边为中心的计算粒度：**以顶点为中心的计算方式虽然简单易用且对某些算法可以避免访问非激活顶点的出边，但会引入对出边的数据，或按照出边生成的消息数据的随机访问，这个访问在磁盘环境下，引入昂贵的I/O开销，影响性能。单机磁盘处理系统GraphChip[55]通过出边按照目的顶点的存储顺序进行排序，避免了随机磁盘访问，但同时引入较大的预处理时间。为次，X-Stream[56]系统采用以边为中心的计算方式。具体地，以此迭代分为两个阶段：1) Scatter阶段，顺序扫描出边，获取对应的源顶点值及其激活标志位，如果源顶点处于激活状态，则产生并顺序存储消息数据，否则跳过该边；2) Gatter阶段，顺序扫描消息数据，更新对应的源顶点值。显然，以边为中心的计算方式保证了边数据和消息数据可以被顺序存取，且不需要额外的预处理开销。但同时引入了对原顶点数据的随机访问，也无法避免非激活顶点的出边数据的顺序扫描操作。对于前者，X-Stream对顶点数据进行分块操作并逐一加载到内存，以满足Scatter和Gatter阶段的数据访问需求，降低随机存取的开销。此外，以经典为计算中心和以边为计算中心的计算方式，均破坏了图本身的联通性，使得数据访问过程的局部性较差。单机磁盘系统PathGraph[书37]设计以路径为中心的计算方式，将图数据划分基于树的若干个分区，提高了图遍历过程中的数据加载的局部性。但以边和以路径为中心的遍历方式都存在着较大的应用局限，尤其缺乏在分布式环境下的成功实践。

### 消息传递机制

在图处理应用中，每一个图顶点都需要向周围邻居节点发送消息或从周围邻居节点接受消息，而图中边可以理解为收发消息的通道。对于一般的图而言，边的数目要远大于图中顶点的数目，如此大规模的消息通信，如果处理不当，很容易成为整个图计算过程的瓶颈，本小节从消息时序控制、消息交互模式、网络通信平台和顶层消息优化技术四个方面阐述消息传递的核心技术。

**消息发送时序控制**：根据消息发送的时序控制，可以将现有的图处理分为同步集中式与异步两种方式。其中同步集中式中计算任务与消息通信串行之行，在计算完毕后，统一进行消息传递，控制和实现方式简单，可以最大效率的利用网络I/O，但也容易造成瞬间的网络通信阻塞，增大了发送端的消息存储开销；异步方式中，计算任务与消息通信并发执行，在计算过程中就可以发送消息，此种方式将全部的消息量分散在不同的时间段，可以充分的利用空闲的网络资源，避免瞬间的网络通信阻塞，主要图处理系统GRAPE+[]，Maiter[42]就是采用异步通信的方式。

**消息交互模式**：图处理过程中的消息交互模式可以分为Push和Pull两类。Push方式是一种以源顶点为中心的计算方式，即图计算系统遍历图中所有顶点，完成顶点的更新计算，然后按照出边广播消息给目的顶点。这种方式处理机制简单，且在内存资源充足的情况下，由于各个任务可以分别处理源顶点，因此具有很好的并发性。另一方面，考虑到出边中的目的顶点的分布具有较差的局部性，从目的顶点的角度分析，其收到的消息在时间和空间方面具有较差的局部性。而作为同步处理系统的内在约束，在第次迭代，一个顶点只有接收到了所有步的消息之后，才可以启动本地计算更新顶点的值。Push方式下较差的时空局部性，使得图处理系统必须存储所有已收到的消息，知道所有源顶点完成消息的发送，这显然增大了接收端昂贵的I/O开销。目前，Pregel、GRAPE、Giraph等系统均采用Push方式实现消息传递。其中Giraph系统同时支持内存和磁盘两种版本图计算应用。根据Zhou[57]等人的测试结果，当内存资源紧张而采用磁盘处理时由于频繁的存取磁盘信息，Giraph的处理性能极具下降。分布式内存系统Trinity[58]尝试在源顶点与目的顶点之间建立一个二分图，协调消息的生成顺序，使得接收端收到的消息能够及时被处理，从而减少内存开销。然而二分图的建立是一个NP-难问题，为简化该问题，Trinity忽略了出度顶点，即高出度顶点所发送的消息仍然需要一直驻留在内存。由于现实图数据具有幂律偏斜特点，因此大量消息无法被及时处理。

另一种常用的消息传递模式是以目的顶点为中的Pull模式，即各个任务遍历目的顶点，根据入边向对应的源顶点主动请求消息数据。显然，当一个目的顶点收到所有请求的消息后，即可立即执行更新计算，避免存储大量的消息数据。Chronos[59]、Seraph[60]等系统均支持Pull方式消息传递机制。然而，Pull方式下，在请求消息数据时，需要传送对应源顶点的ID，在分布式环境下引入额外的通信开销，尤其对于单源最短路径问题，在迭代的收敛阶段大部分目的顶点可能已经处于收敛状态，Pull方式则会引入额外的请求开销。Kylin[61]系统根据目的顶点的状态决定是否向源顶点发送消息请求，这虽然降低了Pull的请求开销，但是可能导致部分需要发送消息的源顶点被忽略，最终影响计算结果。GraphLab[43]则采用共享顶点状态的方式将Pull请求的网络通信开销转为本地操作，具体的地，对于某个任务，如果位于的顶点是中顶点的入度邻居，即中的请求消息，则在中维护一个顶点的共享状态。在迭代计算过程中，如果顶点的值发生变化，通过网络消息同步对应的共享状态值。而目的顶点在启动更新计算后，仅需要根据入边从本地的共享状态值中获取需要的消息数据即可，避免了传输源顶点ID所引入的额外网络通信开销。然而，根据Zhou等人的测试结果，这个方式本身也会引入较大的内存开销，尤其对顶点值规模较大的算法[57]。

### 网络通信方式

这对现有分布式图计算系统的研究，网络通信的实现主要包括以下四类：1) 基于MPI(Message Passing Interface)的通信方式，典型代表有GRAPE[24]和Maiter[42]；2) 基于RPC(Remote Proceduce Call)的通信方式，典型代表有Hama[28]和Giraph[27]等；3) 基于ActiveMQ的通信方式，如BC-BSP[62]；4) 基于Netty的通信，如Giraph。其中部分系统同时支持多种通信方式。

### 顶层消息优化技术

目前，大规模图数据的计算过程中，网络通信依然是计算瓶颈之一，因此需要对通信规模和消息数据存储加以优化控制以降低消息维护的时空开销。首先利用分割技术，可以降低子图之间的联通性，使大部计算任务可在子图内完成，将网络通信变为本地通信，从根本上减少节点间的消息传递数量。良好的图划分算法可以减少网络开销，但无法降低消息存储的空间开销。其次，针对具体应用，采用聚合机制，也可以减少网络通信规模，如Pregel等系统的Combine技术，将发送给相同目的顶点的消息合并为一条消息，以达到同时降低网络通信开销与内存开销。特别的，异步通信下，由于发送端的消息被异步发送，而消息在目的顶点之间的分布具有较差的局部性，因此能够参与聚合的消息比例较少，导致通信收益降低，甚至难以抵消聚合所引入的消息遍历等额外开销，故GRAPE、Giraph等系统屏蔽了发送端的Combine功能，仅允许接收端的消息聚合，以降低存储开销。最后针对Combine技术的缺点，Zhou等人设计了一种消息在线处理技术，即MOC(Message Online Computing)技术[57]。与Combine技术仅对消息数据可见不同，MOC技术在接收消息时，目的顶点的value也是可见的，因此接收的消息可立即参与顶点的更新计算，而不必缓存，从而节省了大量存储空间，作者在Giraph上实现了MOC技术，实验结果发现，在相同内存配置的情况下，MOC可以显著提高系统的数据处理能能力。同样使用了该技术的图计算系统还包括GRAPE。

### 图数据划分

如何针对复杂的图计算任务合理的组织数据，从而提高整个处理过程的执行效率，已经成为图计算领域的基础性研究问题。主要包括图数据的划分、图数据的存储和索引。首先实现低耦合的划分是实现大图分布式处理的基础操作，是保证负载均衡、减轻“木桶效应”的基础，特别是在处理过程中根据实时的负载变化进行动态的重划分，面临着诸多技术挑战，本小节主要介绍大图数据的划分技术。

图划分是分布式图计算系统进行分布式图数据计算的前提。由于图计算通常按照拓扑结构访问数据，所以每次迭代处理均会引入巨大的通信开销，成为制约分布式处理行性能的关键因素。因此，一个良好的划分算法保证划分后的子图在负载均衡的前提下，减少子图之间的边的数目，从而减少网络通信。另一方面，云计算资源会随着并发处理作业数目的变化和集群中计算节点的增删而动态变化，因此同一个作业在不同时刻提交时，其分布式任务数目不尽相同，这一点被称为分布式处理粒度的弹性变化，这导致图数据需要按照当前的任务数目重新划分，划分结果的不可重用性使的图划分的执行效率也成为影响总计算的重要因素。目前的图划分算法主要包括离线划分算法、和在线划分算法算法。近年来，相关研究主要可以分为两类，以METIS[63]为代表的离线划分算法和以LDG[64]为代表的在线划分算法。前者可以显著优化切分边的规模，降低迭代计算过程中的消息通信开销，因此受到学术界和工业界的广泛关注。然而，离线划分过程需要频繁访问图顶点，引入了昂贵的时间开销。另一方面，在线划分算法可以在图处理系统的数据加载阶段完成图划分，该过程仅扫描一次图数据。与离线划分算法相比，在线划分算法在一定程度上牺牲了分图效果来获得较高的执行效率。但是此类算法通常为集中式的，以便于维护复杂的启发式规则，保证相对较好的划分效果，其扩展性显然受到单机处理能力的限制。虽然已经存在分布式在线划分算法的相关研究工作，但是启发式规则的维护开销仍然显著影响了算法运行效率。

## 运行时间预测

针对算法运行时间预测问题，现有的研究分析方法主要分为基于数学模型的预测、基于机器学习技术的预测、和基于相似性的预测。

### 基于数学模型的预测

基于数学模型的预测，主要是针对算法所使用的运行模型进行时间复杂度分析，除此之外，还可以结合运行环境对算法的运行过程进行更细致的分析。如果对所运行算法的专业领域知识认知比较深刻，并熟知算法运行过程中对运行环境对算法程序产生的影响，那么可以对算法的运行过程进行精确的建模分析。为了量化这一过程，可以首先对系统的特性进行量化，之后建立一个基于系统特性的分析模型，最后对目标机器进行特性量化，从而使用模型的出算法在目标机器上的运行概况，分析资源的消耗情况。

这类方法需要研究人员对算法所应用的专业领域具有较深刻的了解，能够理清不同输入下算法的运行逻辑，并且同时对算法运行所在的环境也具有一定的了解，能够分析算法运行时所需要的开销。因此基于数学模型的预测方法通常适用于大型专有机器上的软件开发，对跨平台性的分析较为困难。

### 基于机器学习技术的预测

Hutter[65]等人提出了基于机器学习技术的算法预测方法，他们根据具体问题实例的特征，建立了算法运行时间与这些特征之间的函数关系，通过对已有数据的训练，得到具体模型参数，最终根据得到的模型用于接下来的时间预测，他们的研究结果显示，他们采用随机森林等模型相比于之前的方法，有更好的预测效果。

此外，对于预测模型的输入，Chun et al.(2010)中提出了一种对输入数据进行特征分析记录的方法，具体做法为：使用程序的插桩技术，在具有循环和分支语句结构中插入计数操作，并将技术的结果作为输入数据的特征，后续对于新的输入，需要事先运行程序的片段，以获取这些计数结果，这种方法虽然需要获取程序的源代码，但是对代码的运行逻辑以及其他程序相关知识不做任何要求。这种基于机器学习的预测方式不需要了解领域的知识背景，适合计算机专业人员进行应用，但是由于缺乏对应领域的知识，因为对模型的解释存在一定的难度，因此需要在训练模型后进行专家分析，以增强模型的可解释性。

### 基于相似性的预测

目前，很多的计算系统均基于同步计算模型设计与实现的。对于这种计算模型，程序的整个运行过程通常可以分为计算、消息同步两个阶段，迭代执行。根据这一特点，可以将一个程序可以近似的认为是多个相同阶段的多次叠加，因此可以根据某次计算中已知阶段的计算时间来预测接下来的未知阶段。这种基于不同阶段的相似性预测对程序运行的逻辑要求较高。目前同步计算模型盛行的环境下，这个假设能够较好的符合大多数程序的运行逻辑。但是并行分布式计算的发展趋势是计算与通信的相互隐藏，异步进行，因此难以使用该技术进行预测工作。

## GRAPE+系统

无论何种图数据处理技术都需要以系统的形式作为载体呈现给用户，合理的设计系统的定位，选取适合的软硬件环境，灵活的运行相关数据结构以提高程序质量，才能将核心的关键技术高效、完整的呈现出来。近些年来，针对大图数据的分布式计算系统层出不穷，采用了不同的计算模型、数据组织等优化技术，在功能与适用性方面也不尽相同，对他们进行充分的研究可以更好地理解相关技术的实现方法。特别是大量高质量的开源图数据计算系统的出现，推动了大图分布式数据处理学术研究和产业业务的快速发展，以它们为平台实现具体算法的优化和二次开发成为了可能。其中GRAPE+系统是Fan等人开发的基于自适应异步计算模型的大规模分布式图计算系统[41]，该系统中，首次结合了同步计算模型与异步计算模型的优点，利用每个计算节点自己维护的参数机制，实现了计算节点可独立的在同步与异步计算模式下自由切换。因为本文的研究内容基于GRAPE+系统，因此本小节该系统的体系架构、编程模型、自适应计算模型机制、和图计算执行流程方面详细地介绍该系统。

### 系统架构概况

GRAPE+体系架构如图11所示：

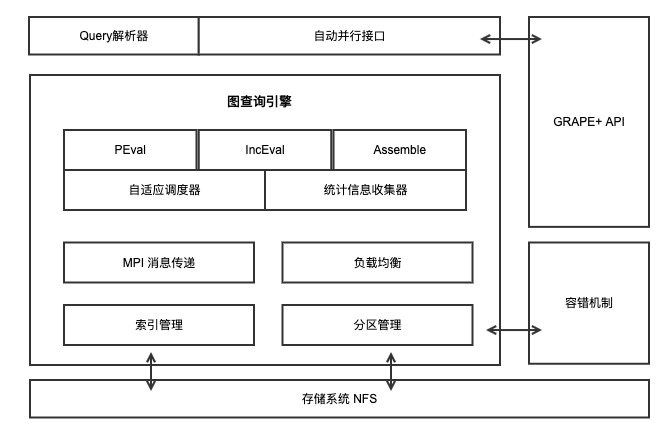


图11 GRAPE+系统架构图

从图中可以看出，整个系统主要分为数据存储、GRAPE+核心组建、系统接口、容错管理四大部分。

**数据存储层**：表示GRAPE+系统处理的源数据的存储形式与存储位置，目前仅支持源数据存储在文件系统的某些文件中。且GRAPE+支持文件格式的图数据载入功能，即文件包含图中所有的顶点，每行表示为的二元组形式分别代表顶点的ID与顶点的附加属性；文件属于图分区文件，GRAPE+支持图数据的预分区功能，该文件包含图中每个顶点所属于分区，每行表示为的二元组形式分别代表顶点ID与该顶点所属的分区ID，GRAPE+指出文件可用现有的图分区系统计算生成，如Xtrapulp图分区工具[66]；文件包含图中所有的边，表示为三元组形式，分别代表边中源点ID，目的点ID以及边上的附加属性信息。

**核心组件层**：位于数据存储层上面的就是GRAPE+核心组件层，该层由图数据存储管理、分布式MPI消息传递机制、自适应异步控制、统计信息收集器等组件组成。其中图数据在GRAPE+中以临接表的形式存储，该存储方式可轻松的获取某顶点的一度出边，因此适合Push模式的消息交互方式；MPI消息传递负责整个系统的通信过程，包括数据的迁移、消息的收发等；自适应异步控制主要负责控制各计算节点的执行进程，即每个计算节点都会独立维护自己与其余各计算节点的执行进度参数，新一轮的迭代计算开始前，各个节点根据参数的状态以决定是否立即执行本轮迭代计算或等待一段时间后进行计算，等待的目的即为了接收更多的消息，以执行更有效的计算，从而加快收敛；统计信息收集器在GRAPE+系统中主要负责收集各节点的迭代进度，并通过MPI消息传递机制广播给所有计算节点。

**系统接口**：GRAPE+系统的接口层提供两种功能，其一是提供用户以为计算框架，基于MPI消息传递的编程接口。用户实现上述三个接口函数完成对图应用算法的处理，即可在GRAPE+系统上运行期望的图应用任务，整个计算过程并行化操作对用户透明。

同时GRAPE+提供了负责集群的环境搭建以及部署工作，并提供了一键式日志管理、系统启停等服务。

### PIE编程模型

**图符号术语**：给定某有向图或无向图，其中代表图中点的有限集合，代表图中边的集合，代表图中顶点或边上的属性；

给定某一自然数，我们采用某种分图策略将图分割为个片段，其中每个均为的子图，且有，，，，针对每个顶点 ，有，针对每条边，有。

编程模型采用边分割或点分割策略的来分割图，当使用边分割时:

1) **内部点集合**：代表内部点集合，其中针对每个点存在一条边 ，使得，且。

2) **外部点集合**： 代表外部点集合，其中针对每个点存在一条边 ，使得，且。

我们将上述成为子图的边界点集合。

**PIE编程模型**：众所周知，图计算问题作为图查询问题的一类，针对某一问题实例给定问题，为回答该问题，该模型需要用户提供以下三个函数：

1. PEval (局部估值函数): 该函数将子图与问题作为输入，并在子图上计算回答该问题，得到结果，其中针对该问题的所有单机图算法均可直接使用。

2. IncEval (迭代增量函数): 该函数将已有结果 、子图增量 、与问题作为输入，并在更新后的子图上回答问题，保证有 ，其中 是在原有的基础上做的更新计算。

3. Assemble (聚合函数): 该函数用于聚合其他子图在PEval或IncEval阶段计算所产生并同步本子图的更新量，最后将聚合结果更新到已有结果。

### 图计算执行流程

1) 首先，用户在系统环境下提交GRAPE+作业，整个过程中用户通常以脚本命令的方式执行并添加相应的控制参数，作业提交成功后，用户可以在控制台监控作业状态，查看作业日志信息。

2) 系统接收作业提交会，根据提交参数中的进程数执行分布式图计算任务，该任务分配计算节点，同步提交参数，正式进入图计算流程。

3) 图计算流程中，每个节点首先进行图数据的加载任务，GRAPE+系统本身不支持图数据的划分功能，因此载图工作需要读取预先处理好的图分区文件，因此加载工作时只需要根据图中顶点所属的分区信息选择性的载入本地的数据管理器即可。

4）图数据载入完成后，才真正进入迭代计算的计算框架。根据用户编写的接口函数，即函数，计算框架自动在每个计算节点上迭代完成上述函数，整个计算过程与消息传递过程异步进行，直到满足迭代终止条件或到达预设的最大迭代轮数，整次图计算任务结束，输出计算结果到本地文件系统。

## 本章小结

本章介绍了本文研究所涉及的相关技术背景，首先给出了图的定义，并介绍了工业界与学术界常用的图应用算法，包括单源最短路径算法、PageRank算法、HITS算法、联通性算法、随机游走图采样算法、标签传播算法、以及社区发现算法。然后，本章介绍了分布式图计算的相关技术，包括计算模型、图划分技术、消息传递机制。接着，本章介绍了算法运行时间预测的常用技术，包括基于数学模型的预测、基于机器学习与历史数据的预测、以及基于相似度的预测。最后本章介绍了GRAPE+自适应分布式图计算系统，本文接下来的研究工作均在该系统上开展。

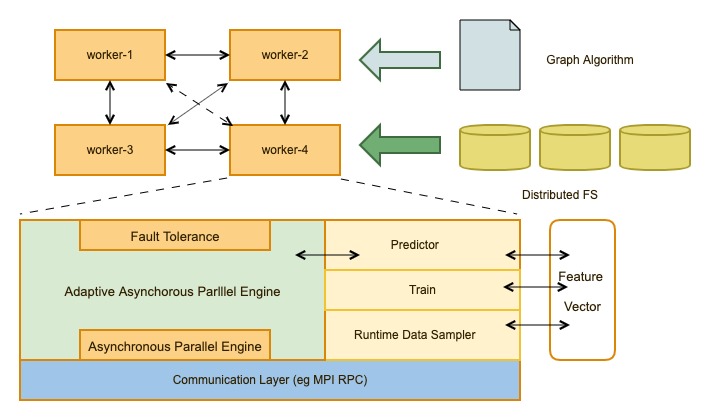
# 基于运行时间预测自适应计算模型

在前面章节中，我们针对现有的图计算模型进行了研究，并且分析了不同计算模型的优缺点，发现了潜在的性能提升空间。在这一分析的基础上，我们基于传统的自适应计算模型，设计了基于运行时间预测的自适应计算模型，即RPAP计算模型，并在GRAPE+系统的基础上扩展了该系统的架构，实现了GRAPE++系统。

RPAP计算模型的设计与实现的关键部分在于重构分布式图计算系统的调度层，使得每个计算节点独自控制自己的调度进程。在RPAP模型调度模式的设计中，我们需要定义严谨且正确的调度方式，不仅如此，还要时刻维护正确的图并行计算语义。在具体的实现中，则需要确定不同时刻下的每个进程的当前状态以及状态转换过程。同时，在系统的设计层面行需要增加对运行时信息的收集模块、模型的训练与预测模块，以利于自适应计算模型的顺利进行。整个系统的实现还需要针对使用的数据结构进行简化，最大可能降低训练与预测引入的额外开销，以提升图计算性能。

本章首先介绍RPAP模型的结构，并详细介绍该计算模型的设计与实现上的细节，接着本章给出RPAP计算模型下的工作流程，并给出单源最短路径算法在该模型下的伪代码。

## RPAP计算模型总体结构



**图12 RPAP计算模型整体结构图**

RPAP计算模型的总体结构如图12所示，整个模型分为RPAP调度引擎模块、网络通信模块、数据采样模块、模型训练与预测模块4个部分。

**自适应调度引擎模块**：在RPAP计算模型中，我们将分布式图计算的异步调度实现为一个新的独立调度引擎，整个引擎的执行过程中，会与预测模块进行交互，根据交互得到的预测结果，通过参数控制，以决定是否立即进入下轮迭代或等待一段时间后进入下一轮迭代计算，该过程看起来就像是每个计算节点独立控制自己的图计算进度，不受任何其他节点影响。同时RPAP计算模型通过参数控制，兼容同步与异步两种不同的计算模型，这一部分在接下来的小节中会详细说明。

**运行时数据采集模块**：主要负责采集图计算过程中的运行时信息，采集后的信息会收集到模型训练模块做为机器学习预测模型的训练数据，采集过程各个计算节点相互独立。目前主要分为离线数据采集和实时数据采集两种。

我们发现，某些算法针对相同的图数据，有不同的图查询问题。例如单源最短路径算法，其目的是求解源点到图中其它顶点的最短距离，因为图中的任意顶均可做为源点，因此每个顶点都可作为一次图计算的查询任务。而离线数据采集就是通过独立运行多遍图计算任务，但是每次任务的图查询问题不同，并通过日志输出的方式记录每次任务的运行时数据，从而得到相同图数据下不同求解问题的运行时信息，后续的模型训练工作完全可以基于这些已收集的日志数据，提供预测服务。我们可以看到，通过离线数据采集方式以获取大量的训练数据，使我们的模型更加有效。

但我们也注意到例如图连通性分析、PageRank这一类算法，往往针对固定图数据，查询任务与结果单一，因此无法像单源最短路径算法一样获取大量离线数据。所以，这类算法可以通过实时数据收集的方式，即在图计算开始阶段，通过前几轮迭代以收集小部分数据，同于后续模型的训练预测，因此这需要我们的模型选取在小样本集上具有较好的预测效果。

**预测模块**：通过数据采集模块收集到的运行时信息，会在预测模块中进行异步训练，如果采取的是离线数据采集，则在图计算引擎启动阶段，便根据离线的数据完成对模型的训练工作。所以，在整个图计算系统的执行过程中，预测模块会针对现有的运行时信息，通过已训练好的模型，给出对运行时间的预测。因此各个计算节点会轮询的与预测模块进行交互，以控制自己的执行进度。

## RPAP计算模型的并行模式

**RPAP计算模型**：RPAP计算模型中，为解决同步模型下的“木桶效应”及异步模型下的冗余计算问题，每个计算节点引入了限定值，之后每轮迭代计算开始前，各计算节点需要选择性的是否等待长的时间以累积更多的消息，我们给出基于下述函数进行动态调整：

上述函数中的各参数描述如下，表示计算节点新一轮计算等待的时间；表示计算节点当前接收的消息量。直观上来看，如果 越大，则当前计算节点接收消息越多，应尽快开始下轮迭代计算；与分别表示预测的时间与消息到达速率；表示该计算节点前一轮计算后的空闲时间，引入的目的是防止计算节点无限期的等待；为RPMP模型中唯一的超参数，取值处于(0,1)之间，用于控制计算节点等待的时间，且越大，计算节点等待的时间越长。

上述公式的执行过程在各个计算节点间相互独立，每轮迭代计算开始前，各计算节点首先预测下轮迭代计算的运行时间以及消息到达速率，即对应公式中的与，并通过简单的乘法运算得出本轮计算期望的消息量，并将期望的消息量与当前已接收到的消息量进行比较，若期望的消息量大于已接收的消息量，则通过计算需要等待的时间；否则通过计算需要等待的时间。

图

**并行模型**：RPAP模型本质上仍然采用上述介绍的PIE编程模型，给定某一图计算问题实例及在该实例上的问题，RPAP模型需要用户指定上述PIE函数。如图13所示，首先，将输入的图数据 通过边分割或点分割策略分为多个片段 ，每个片段存储在某计算节点上，除此之外，RPAP模型需要做为Coordinator节点，用于完成后续将要提到的集群终止状态检测与运行时特征的收集等任务。因此，若 ，则多个片段会分到同一个虚拟机上并分享内存。

传统的自适应计算模型中，每个计算节点等待的时间是通过判断是否接收到来自足够多的其它节点的消息决定的，例如，如果最开始的阈值设为20，即代表如果本轮迭代计算开始前，已经接收到了来自20个或以上的节点发来的消息，那么则立即进入本轮计算，否则无限期的等待，直到接收到20个节点发来的消息为止，因此无论某一节点是否产生同步消息，都需要向其它计算节点发送空消息以防止图计算运行中卡死问。我们可以看出，在传统自适应模型中，每个计算节点只需要记录当前已经接收到消息的节点数目，便可选择性的决定是否等待一段时间，整个过程虽然参数单一，逻辑简单，但该模型仅仅适用图数据划分极不均匀或节点间计算能力差距极大的情况，并且需要不断的手动调试以找到最佳的阈值参数。

与传统的自适应计算模型不同的是，RPAP计算模型通过分析并提取影响迭代计算时间的关键特征，通过模型的训练，合理的预测每轮迭代计算的运行时间，并通过计算选择性的决定是否等待一段时间来控制各计算点的相对进度。因此，RPAP模型还需要PIE模型的基础上，增加以下声明。

1. **节点状态变量**：PEval中每个片段都会声明并维护一个集合，该集合用于存储子图中每个点的状态信息，其中包括结果信息，在接下来的IncEval阶段中，该集合也用于根据接收到的消息，更新有用信息。

2. **聚合函数**：PEval中需要指定聚合函数来解决多台机器传递的消息同时作用于相同节点的问题，常用的聚合函数如函数、函数、函数等。

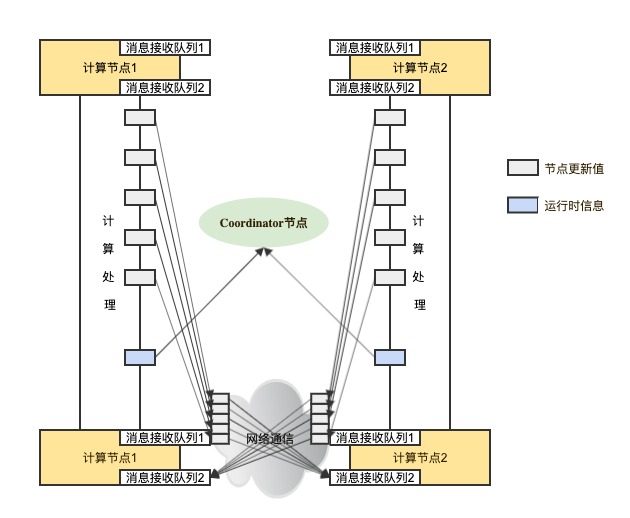
3. **运行时数据信息收集**：IncEval中每个片段在完成本轮计算后，会向Coordinator节点发送本轮的运行时数据信息，该信息以数组的形式存储，且数组中的每个值代表一个特征，主要包含当前该片段的状态信息与特征信息，具体的特征细节在下一章详细讲述。

4. **延迟函数**：RPAP模型需要延迟函数判断节点的等待事件，该函数的返回值为节点需要等待时间，输入则为当前的特征参数。

**终止条件**：RPAP模型的终止条件与传统自适应计算模型模型相同，即新一轮迭代计算开始前，如果没有接收到任何来自其它计算节点的消息，则回向Coordinator节点发送结束命令, 当Coordinator接收到所有计算节点的命令后，会广播各节点终止命令，后续各节点会对此进行确认自己是否真的结束，如果此时某个节点仍有任务需要计算，则回应Wait命令，继续进行下一次迭代计算，Coordicator节点则广播其余节点继续执行任务，并重新进入监听等待状态。

## RPAP计算模型的消息传递机制

RPAP计算模型的消息传递机制本质上与传统自适应计算模型相同，属于点对点异步通信，通过过程发生在图计算的任意时刻，即任意节点在任意时刻都可将消息发送给其余各节点而不需要关系其余节点的状态，同样，该节点也会在时刻接收来自其它各节点的消息，保存在缓冲buffer中，该过程不会阻塞任何计算过程。但与传统自适应模型不同的是，每个节点除发送本轮更新的消息外，还需像Coordicator节点发送本轮的运行时信息，该信息用于后续的模型训练，消息传递的过程如图14所示。在每轮迭代计算结束后，各节点更新本轮计算的中间结果，并将更新结果发送给其它计算节点，为实现该消息传递机制，各计算节点需声明维护以下数据结构：1) 全图顶点与该顶点所属计算节点的映射，以便顺利将该顶点的更新消息发送给对应计算节点；2) 一个消息缓冲区接收队列，用于接收来自其他计算节点发来的消息，即图14中的消息接收队列

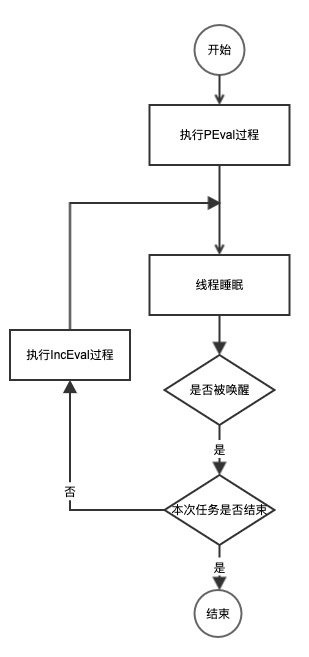


**图14 RPAP计算模型异步消息传递机制图**

除此之外，RPAP计算模型行采用Push类型的消息交互方式。其以源顶点为中心，即在每轮迭代计算后，遍历图中所有顶点，完成顶点值的更新，然后按照出边广播消息给目的顶点。这种方式处理机制简单，且在内存资源充足的情况下，由于各个任务可以并行处理源顶点，因此具有很好的并发性。

## RPAP计算模型执行流

RPAP计算模型的整个执行过程包括两个线程，其中计算线程用于迭代执行RPAP模型下的PIE函数，控制线程根据RPAP模型的预测结果来控制线程的执行。

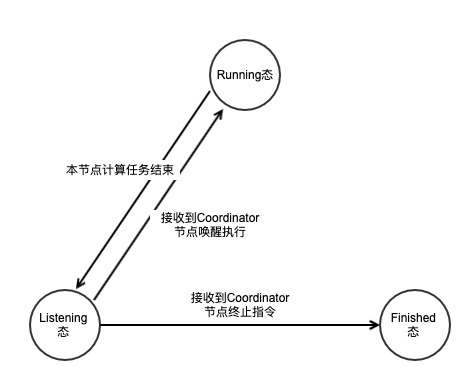


**图15 计算线程执行流程图**

计算线程的执行流程如图15所示，因为PEval过程不参与预测，因此初始阶段线程首先执行PEval函数，根据图分割后的子图数据执行本地计算，得到运行结果，并根据子图临接情况生成消息数据，存放在数据结构中。此时线程进入睡眠监听状态；当接收到来自控制线程的唤醒通知后，判断本次任务是否结束，如果任务结束，则清理现场，线程退出；否则迭代执行IncEval函数，并在执行完成后重新进入睡眠监听状态。

如图17所示，控制线程首先初始化节点状态为Running态，接下来轮训执行下述流程：

1. 根据计算节点当前所处的状态，走不同的分支流程，节点状态包含运行态Running、监听态Listening、和终止态Finished，状态转换图如图16所示。

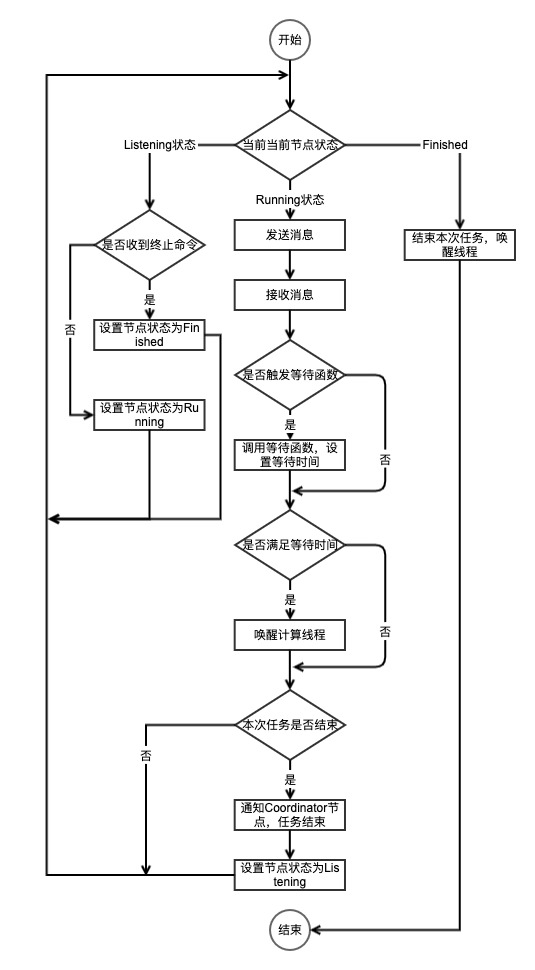


**图16 计算节点状态转换图**

2. 如果节点状态为Running态，1) 首先检测本地是否有未发送的消息，若有则通过MPI消息传递机制，发送给目标节点其次；2) 其次，检测是否接收到其他计算节点数据的发送请求，若有则接收消息数据；3) 之后根据是否成功接收或发送消息，判断是否需要触发延迟等待函数，如果需要，则调用延迟函数，设置该几点等待时间，否则跳过调用延迟函数；4) 接下来判断是否满足等待时间，如果满足，唤醒计算线程，否则跳过唤醒计算线程；5) 之后判断本计算节点计算任务是否全部结束，如果结束，通知Coordinator节点，并设置节点状态为监听态Listening，重新执行步骤1。

3. 如果节点状态为监听态Listening，1) 首先判断是否接收到来自Coordinator节点的终止指令，如果接收到了该指令，则设置节点状态为终止态Finished，否则设置节点状态为运行态Running，重新执行步骤1。

4. 如果节点状态为终止态Finished，则结束本次任务，唤醒计算线程。



**图17 控制线程执行流程图**

从上述流程可以看出，RPAP模型虽然每个计算节点根据预测结果独立的控制相对进度，但其本质上仍属于异步计算模型，因此所有异步模型下的语义正确性保证，都适用于RPAP计算模型。

## RPAP计算模型调度实例

之前我们介绍了图计算领域中的经典算法，本小节我们以单源最短路径算法为例子，给出RPAP模型的下的调度实例。算法的输入输出形式化表述如下：

Input: 带权有向图 ，与图中某顶点

Output:

下面我们以单源最短路径问题为例，给出对应的RPAP并行模型

**表1 RPAP计算模型下单源最短路径算法PEval函数**

|  |
| --- |
|  |

如表1所示，在PEval阶段中采用了众所周知的求单源最短路径的算法—Dijstra算法。但与该算法单机版本不同的是1) 我们针对该子图中的每个点定义元组，其中代表该点与源点的距离，初始化为无穷大，意味着与源点不可达；代表在本轮计算中该点是否被访问过，初始化为。该元组声明在PEval中，但也在IncEval中共享，后续也同样用于更新IncEval接收来自其它计算节点的消息。2) 我们采用做为聚合函数，即当该片段中某个点接收到了来自多个节点的更新消息时，我们选取其中的最小值做为最终的唯一更新消息。

上述PEval过程执行结束后，每个子图会将外部点的距离存储到预先声明的数据结构中，接下来会被计算节点异步的发送给目的节点。

IncEaval的过程如表3所示，首先对接收的消息执行聚合函数，并更新对应点的值，之后同样采用Dijstra算法计算本轮最短路径，并将计算后的最新值以消息的方式传递给其它计算节点。与传统自适应模型不同的是，IncEval过程还需将本轮计算的运行时信息发送给Coordinator所在的节点，用于接下来的模型训练与预测。

如表2所示，除PEval与IncEval函数外，RPAP计算模型还需要Attl函数，即延迟等待函数，用于动态调整各节点间相对进程，该函数对用户透明。此例中不需要Assemble函数，因此不做介绍。

**表2 RPAP计算模型下延迟等待Attl函数**

|  |
| --- |
| *min step* |

**表3 RPAP计算模型下单源最短路径算法IncEval函数**

|  |
| --- |
|  |

## 本章小节

本章首先介绍了基于运行时间预测的自适应计算模型，即RPAP计算模型的总体结构，其中详细介绍了RPAP计算模型的自适应调度模块、运行时信息收集模块、以及预测模块；接着，本章给出了RPAP计算模型定义与并行模式，并从消息传递机制、消息交互方式、以及模型的执行流程方面分别介绍了该模型的执行细节；最后，本章通过单源最短路径的例子介绍了RPAP计算模型的编程细节。

# 运行时间与消息达到速率预测

通过上一章对RPAP计算模型的详细介绍，我们知道该计算模型希望通过将预测的消息量与已经接收到的消息量进行比较，从而计算出下一轮迭代计算前可能需要等待的时长，目的是接收更多的消息，从而增加每轮迭代计算的质量，加快收敛。但是我们发现，分布式图计算中消息的预测是极其困难的，因为某个计算节点接收的消息量不仅取决于该计算节点与其余各节点的的边连接情况，更取决于与之有边相连关系的各节点在上一轮迭代的计算情况，甚至网络波动等外界因素都会对消息量的预测造成致命的影响。因此，我们通过分析，完成对每轮迭代运行时间和消息到达速率的预测，并通过公式计算，已达到预测消息的目的。

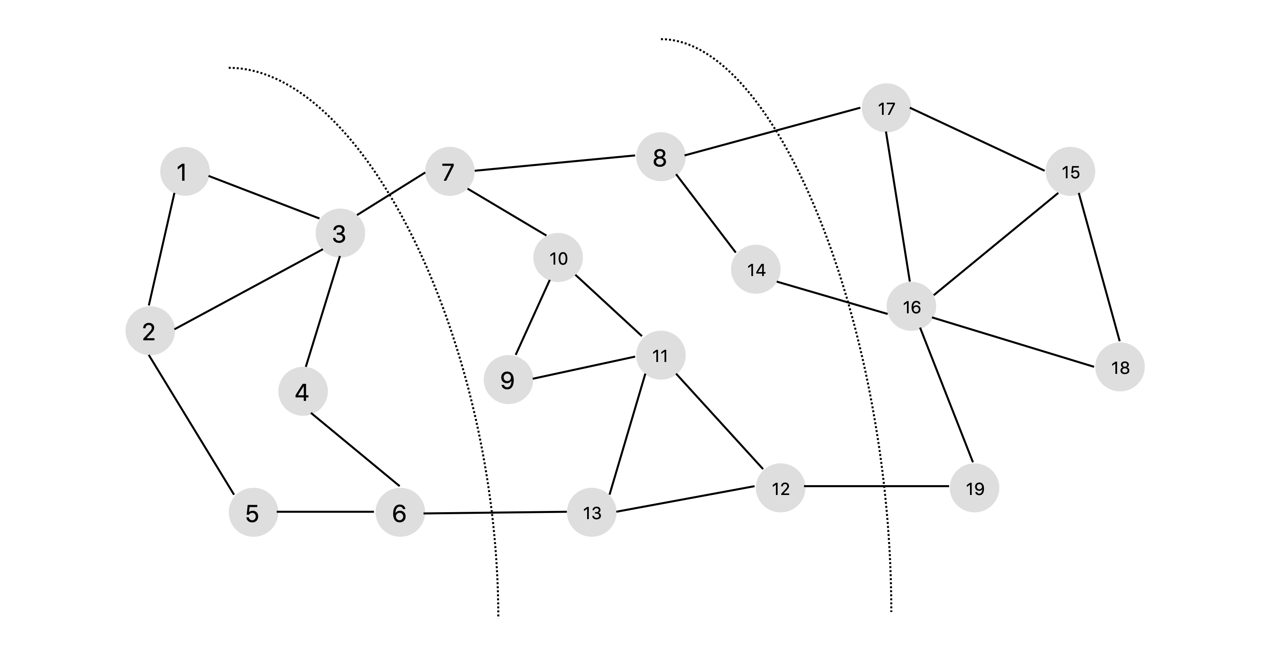
本章首先分析了目前工业界与学术界常用的8种图应用算法，包括单源最短路径算法、广度优先搜索算法、PageRank算法、HITS算法、社区发现Louvain算法、标签传播算法、随机游走图采样算法、和弱联通性算法的两种实现，在PIE编程模型下的通过对上述算法运行时状态的打桩监测，我们将算法分成3类。针对每类我们给出了不同的图特征提取方案，并采用不同的机器学习模型进行训练预测。实验表明，我们我们的预测十分准确。

## 算法分类

### 算法的运行时行为分析

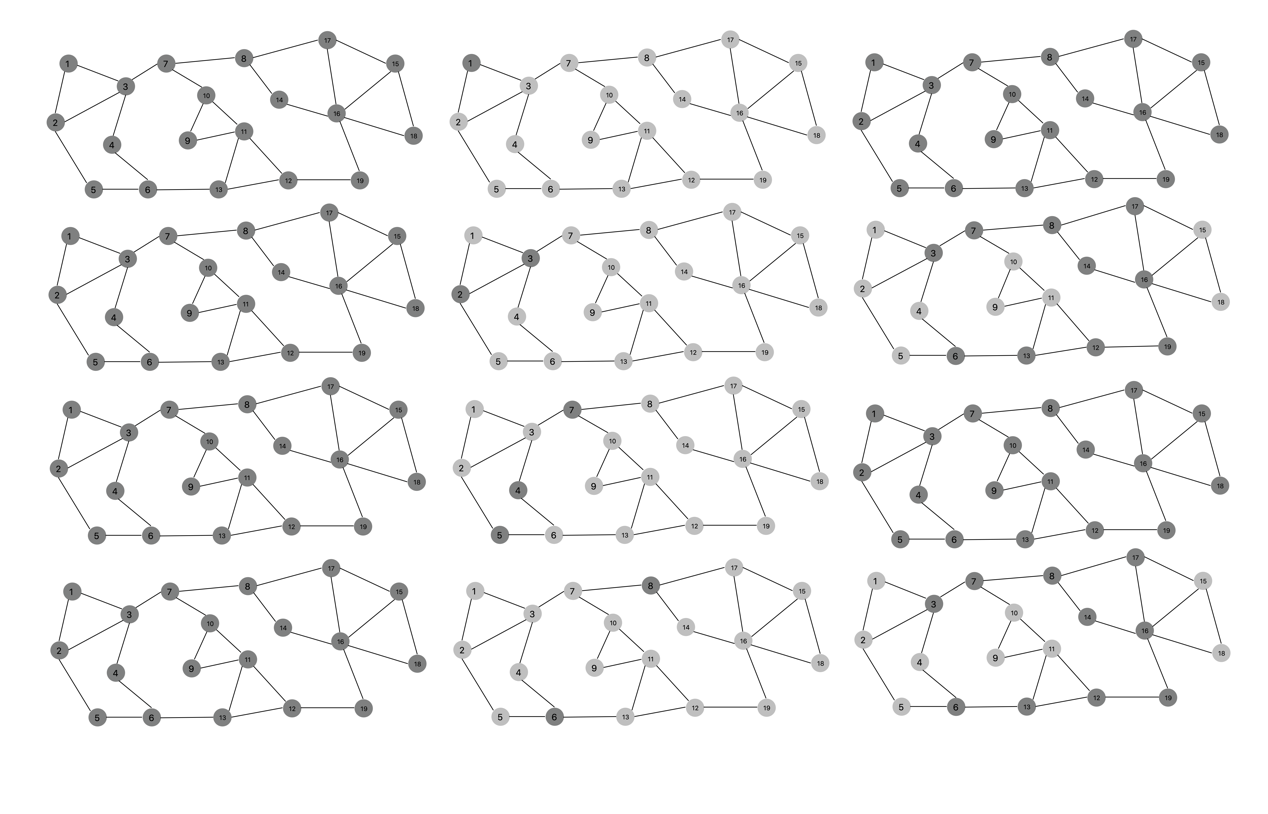
本小节我们调查了上述图应用算法在PIE编程模型的运行时行为。我们发现，算法在迭代计算过程中，子图中的顶点不全都参与计算，大部分算法往往每轮计算真正访问的活跃点数目都不相同，因此我们将图算法在PIE编程模型下的运行时行为定义为本轮迭代计算活跃点的数目，即真正参与图计算的点的集合。

我们以图18为例，详细介绍图应用算法在计算过程中活跃点数目的变化情况。如图所示，图包含19个节点以及27条边，假定该图采用边分割的策略分为3个子图，，和，其中子图包含顶点集合以及与这些点所连接的边，包含顶点集合 以及与这些点所连接的边，包含顶点集合以及与这些点所连接的边。其中与有两条边相连，与有3条边相连。



**图18 算法运行时行为分析样例图**

图19展示了PageRank算法，单源最短路径算法以及HITS算法在同步计算模型中前4个超步运行下每轮迭代计算的活跃点情况。其中，每个超步计算中，浅灰色的点代表没有参与本轮迭代计算，为非活跃点；而深灰色点代表活跃点，意味着参与了本轮迭代计算。



**(a) PageRank算法 (b) 单源最短路径算法 (c) HITS算法**

**图19 算法运行时行为分析样例图**

其中图19-a展示了PageRank算法在前4轮超步下活跃点的变换情况，图中所有顶点在每轮计算中均为活跃点，同样所有子图均参与了全部的迭代计算；图19-b展示了单源最短路径算法在前4轮超步中下的活跃点的变换情况，分别是,,,，每轮迭代计算的活跃点数并不恒定；图19-c代表了HITS算法在前4轮超步下的活跃点的变换情况，可以看出该算法在奇数轮迭代中，与PageRank算法相似，所有顶点均为活跃点，而在偶数轮迭代中只有部分顶点，即接收到更新消息的顶点处于活跃状态。

因此我们从中分析出，不同的图算法在每轮迭代计算中，其活跃点的集合以及活跃点的数量有不同的更新行为，接下来我们给出上述所有算法的现实图数据上的活跃点数量的变化折线图，图中横坐标为图计算迭代轮数，纵坐标为每轮计算活跃数目与图中顶点总数的比值。上述所有算法均需实现上述用户定义的PIE函数。

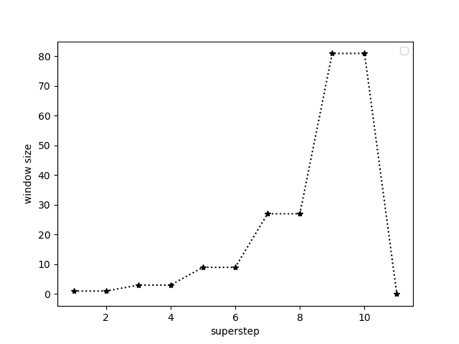
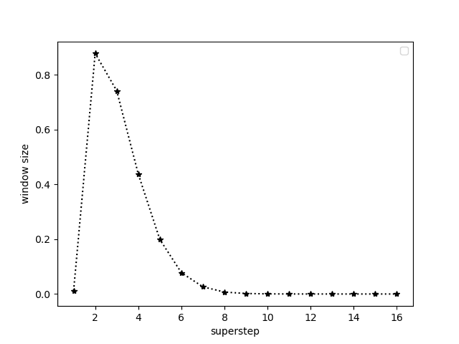


**(a) PageRank (b) 标签传播算法**

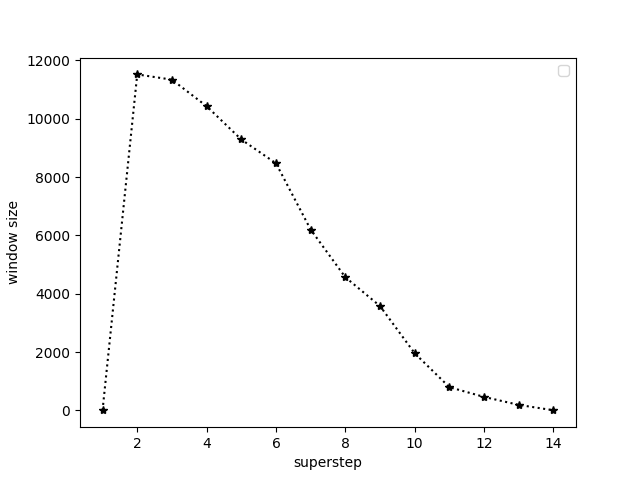
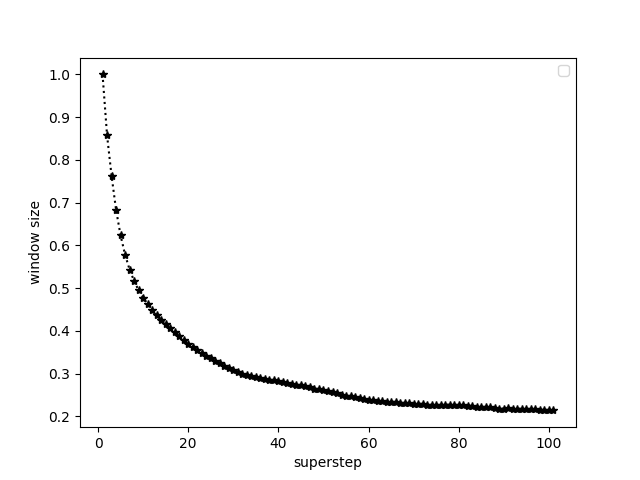
**图20 PageRank算法与标签传播算法在PIE模型下的运行时行为**

图20中可以看出，PageRank算法和标签传播算法在每轮迭代计算中，所有顶点均处于活跃状态，且所有顶点都会向其邻居节点发送消息，因此我们将该类算法称为持续活跃型算法，这类算法最大的特点是每轮计算量与消息传递数量恒定，不会受到其它计算节点的运行行为影响。

图21中可以看出，单源最短路径算法、广度优先搜索算法、以及基于随机游走的图采样算法，其在折线图上活跃顶点数目的变化表现为为先升高后降低。其中图采样算法在采样终止时降为最低；而图22所示的图联通性算法的不同实现版本在折线图的表现形式为开始升高，之后逐渐下降。我们发现这些算法在每轮迭代计算过程中，活跃点的数目主要与接收到的消息情况相关，因此我们将上述5种算法称为消息依赖型算法，这类算法最大的特点是每轮的计算量与传递传递数量不定，且依赖于活跃点数目与活跃点集合。

 **(a) 单源最短路径算法 (b) 广度优先搜索算法 (c) 图采样算法**

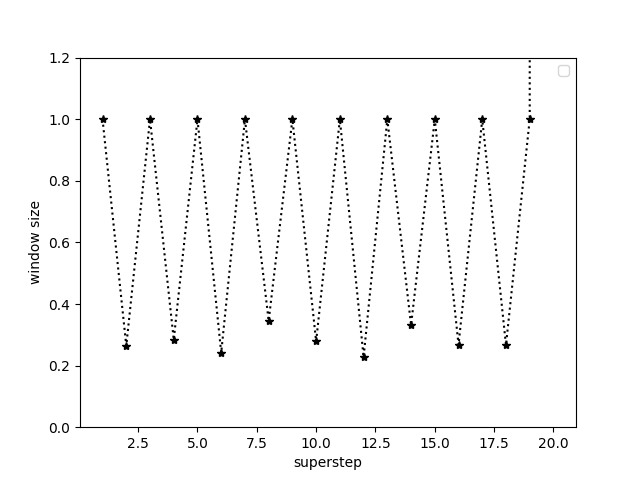
**图21 单源最短路径、广度优先搜索、图采样算法在PIE模型下的运行时行为**



**(a) Hashmin版本 (b) 若连通性**

**图22 若连通性算法及其Hashmin版本在PIE模型下的运行时行为**

图23中可以看出，HITS算法及Louvain算法在折线图上表现出一定的规律性，其中HITS算法在奇数轮迭代计算中，所有顶点均处于活跃状态，而偶数轮的迭代计算中部分点处于活跃状态；而Louvain算法则每隔几轮迭代，活跃点数变化一次。通过分析，这类算法典型的将一次计算流程分为多个阶段，因此我们将这类算法成为多阶段型算法。



**(a) HITS (b) Louvain**

**图23 HITS算法和Louvain算法在PIE模型下的运行时行为**

### 持续活跃型算法

该类算法每个子图在迭代计算过程中，需要遍历子图内所有的顶点，执行规定的图计算任务，并将本轮计算生成的消息传给邻居节点，表4描述了持续活跃型算法的在PIE编程模型下IncEval的执行过程：

**表4 持续活跃型算法IncEval阶段执行过程**

|  |
| --- |
|  |

其中代表该计算节点接收到的消息。在IncEval执行过程中，此类算法算法通过调用聚合函数，根据接收到的消息对子图内的顶点值进行更新（第1-2行）；接着遍历图中内部点，并在每个内部点上执行相应的逻辑计算（第3-4行）；最后遍历图中外部点，将更新后的消息发往其它计算节点（第5-7行）。

表5详细介绍了PageRank算法在IncEval阶段的具体执行过程。其中数组记录每个点的最终结果；数组记录每个点的当前更新量。首先，通过累加方式，记录内部点的当前更新量（第1-2行）；接着遍历内部点，并计算当前该点对周围一度出边邻居贡献的分值（第4-10行），并沿着出边依次更新目的顶点的更新量（第11-12行）；最后遍历图中外部点，将更新后的消息通过信息传递发送其它计算节点（第14-15行）。该过程迭代进行，直到满足预设的最大迭代轮数或收敛条件。我们可以看到，因为PageRank算法在IncEval阶段的末尾会向图中所有外部点所在的计算节点发送消息，因此网络通信量基本恒定；不仅如此，由于计算过程中会遍历内部点并且内部点均贡献了相同的计算量，所以整个计算过程不受消息传递影响。标签传播、图着色算法均属于这一类型。

**表5 PageRank算法IncEval阶段执行过程**

|  |
| --- |
|  |

### 消息依赖型算法

该类算法在计算开始时通常只有一个点或若干顶点处于活跃状态，而接下来的迭代计算过程中，某一顶点的活跃性取决于是否接收到消息，因此消息的传播条件决定了这类算法的活跃点数目以及活跃点集合情况，其在PIE编程模型下IncEval的执行过程如表6所示。从表中分析，在IncEval执行过程中，此类算法在计算开始前通常需要声明活跃点集合（第1行），之后通过调用聚合函数，根据接收到的消息对子图内的顶点值进行更新，并将更新过的顶点插入集合中，意味着该点将参与本轮迭代计算（第2-4行）；接着遍历图中内部点，根据顶点本身是否处于活跃状态，即是否在集合中，选择性的执行相应的计算逻辑（第5-7行）；最后遍历图中外部点，将更新后的消息发往其它计算节点（第8-10行）。

**表6 消息依赖型算法IncEval阶段执行过程**

|  |
| --- |
|  |

表7详细介绍了单源最短路径算法在IncEval阶段的具体执行过程。首先，通过声明优先队列，一方面代替我们上文提到的更新集合，记录本轮迭代计算开始前的活跃顶点（第1行）；另一方面，该队列也是Dijstra算法执行过程中所需要的数据结构。之后，通过轮训的检测队列是否为空，从队列中弹出当前某一活跃顶点（第5-6行），接着根据当前活跃顶点，遍历其一度出边，如果满足最短路径判断条件（第7-9行），则更新目的顶点值，如果目的顶点仍为内部点，则加入队列（第10-11行）。最终，整个队列为空时计算过程截止，并遍历图中外部点，将更新后的消息通过信息传递发送其它计算节点（第12-13行）。我们可以看到，算法在IncEval执行过程中，活跃点集合最开始是由接收的消息决定的，而后续活跃点集合的扩张也是基于最开始集合中已存在的顶点计算得来的，因此我们认为整个计算过程不仅受传递的消息所影响，还与最终更新的顶点集合相关，这些相关性正是我们下一节特征提取的关键。

**表7 单源最短路径算法IncEval阶段执行过程**

|  |
| --- |
|  |

### 多阶段型算法

如表8中所描述，通常该类算法无法在一次IncEval计算中完成完整的逻辑计算，例如HITS算法在计算过程中每个点既要算本身的Authority值，也要算Hub值，并且Hub值的计算要依赖于周围邻居节点的Authority值，所以无法用PIE模型的一轮迭代同时完成对所有顶点Authority值与Hub值的计算，因此需要拆分成两次IncEval去做，其中一次用于计算所有的Authority值，另一次根据上轮的Authority值计算Hub值。通常情况下，这类算法会将一次图计算任务分为PIE模型下的n次IncEval计算，这类算法还包括社区发现Louvain和最小生成树算法。

**表8 多阶段型算法IncEval阶段执行过程**

|  |
| --- |
|  |

表9为HITS算法在IncEval阶段的具体执行过程。其中和数组分别用于存放顶点的Authority值与Hub值。算法首先判断本轮的迭代轮述，如果为偶数轮（第2行），则通过累加方式记录内部点的Authority值（第3-4行），接着遍历内部点，获取该顶点一度出边邻居的Authority值，通过计算更新自身Hub值（第5-7行），计算完成后，对Hub结果做归一化处理（第8行），最后遍历图中外部点，将更新后的Hub值通过信息传递发送给其它计算节点（第9-10行）；如果为奇数轮（第11行），则计算逻辑恰好相反，首先记录内部点的Hub值（第12-13行），接着遍历内部点，计算当前顶点对周围一度出边邻居贡献的Authority值，并沿着出边依次更新目的顶点的更新量（第14-16行），最后对Authority结果做归一化处理后（第17行），遍历图中外部点，将更新后的Authority发送给其它计算节点（第18-19行）。整个过程迭代进行，知道满足预设的最大叠戴轮数或收敛条件为止。在本例中，HITS算法通过两轮的IncEval计算共同完成一次逻辑计算，并且每个阶段均类似于持续活跃型算法，即计算过程不受消息传递影响。

**表9 HITS算法IncEval阶段执行过程**

|  |
| --- |
|  |

## 运行时间预测

针对上述分类后的图应用算法，本节分析并选用了多种机器学习回归技术，对算法的运行时间进行建模，并展示预测效果。

### 问题定义与图计算约束

我们将运行对时间的预测定义成回归问题，对于某一计算节点及给定的运行时信息，我们需要训练出一个预测模型，用于评估下一轮迭代的运行时间，因为该预测模型的输出是以毫秒为单位的实数，因此该问题可以定义成机器学习中的回归问题，具体方程如下：

因此预测过程分为以下两个部分：1) 收集运行时信息，相对于机器学习中的特征提取。一个理想的信息应足以描述下一轮程序的运行状态；2) 针对现有的回归模型进行分析、设计并训练出一个适合的预测器。需要特别指出，由于PEval阶段在图计算过程中仅执行一次，因此我们只针对接下来的迭代增量、即IncEval阶段进行运行时间的预测。

另外，除了机器学习领域中过拟合问题与预测的准确性外，图计算领域还关心额外几个指标：1) 预测时间约束，在迭代预测过程中，我们需保证预测时间足够短并尽可能减少预测开销，如果一次预测占用了迭代计算的大部分时间，那么此次预测没有任何意义；2) 训练数据约束，某些图应用算法中，可能无法采取线下离线训练的方式，必须依靠迭代的线上时间完成训练预测等工作，这需要我们的模型在小训练数据集上有较好的准确性；3) 超参数约束，虽然我们针对众多图应用算法采用不同的训练预测思路，但我们更希望针对多样化的数据输入给出一个通用的模型。

### 特征提取

收集运行时信息的过程即为机器学习领域特征提取的过程，特征提取结果的好坏直接影响接下来模型的训练与预测，图计算中的特征提取涉及多方面因素，包括图结构本身，如图的规模、平均度数、各子图边界点个数；运行时信息，如消息的接收数量与质量等不同特征。本节从分布式图计算角度出发，按照上述图算法的分类结果，给出每一类算法的特征选取方案。

在PIE编程模型中，子图中每轮的计算开销主要依赖于活跃点的数目与接收到的消息数。其中活跃点集合是该子图内部点的子集，因为在某些算法中，一次迭代计算中并不是所有的点都会被接收到的消息所激活，因此我们将某个子图 在第s次IncEval迭代计算的活跃点集合记为，因此在第s次计算中，的计算开销可以表示为：

上述等式也同样表达了，针对一轮计算，某个子图计算的执行时间取决以下三个要素：1) 活跃点数目，通常代表要执行点计算的次数； 2) 外部点数目，影响消息发送的数目与数量；3) 消息接收数目，通常影响活跃点数目。

接下来我们将特征向量表示成m维向量，对每个特征向量我们都有一个运行时间，因此点对代表一个样本，整个数据集合为。

**持续活跃型算法特征提取**：我们从上节分析可知，该类算法活跃点数目恒定，永远等于图中内部点数目，不会受接收到的消息所影响，因此针对此类算法，我们更多的针对子图本身进行特征提取，包括图的拓扑结构，提取的特征如表10所示：

**表10 持续活跃型算法特征提取**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 子图ID | 内部点数 | 边界点数 | 总点数 | 总边数 | 接收消息数目 | 图密度 | 边界点比重 |

表中为图划分后每个子图的编号ID；和为为边分割下子图的内部点与边界点数目；为子图总点数；为子图包含的全部边数；为迭代计算开始前接收到的消息数量；为子图密度；为边界点所占的比重。上述特征均可从IncEval每轮计算中直接得到。另外，通过点与边得统计信息很难表达图的拓扑结构，因此我们采用子图ID作为特征向量中的某一特征值来进行表示，目的是增加模型训练的有效性与效率性，同样也可以使用embedding(找几篇论文)等方法将图的拓扑信息表示成向量，但这种方式的引入计算代价十分昂贵，相比于此，我们通过子图ID的方式及其简便。

**消息依赖型算法特征提取**：与第一类算法不同，该类算法每轮迭代计算的活跃点数目并不恒定，且活跃点集合的具体内容受接收的消息所影响，因此除包含第一类算法所提取的特征外，我们还需对接收的消息做映射。提取的特征如表11所示：

**表11 消息依赖型算法特征提取**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 子图ID | 内部点数 | … | 边界点比重 | 消息映射向量 |

其中是包含消息映射的向量，为超参数，代表向量的大小，代表接收消息的点ID映射后的区间，不同的点映射到相同的区间，则采用累加的方式进行处理，因此如果很大，则会提高模型的预测精度，但同时也会增加模型训练的时间，相反减小则会缩短模型的训练时间，但同时也会降低模型的预测精度。例如，子图包含50个顶点，顶点序号为从1开始的自然数逐渐递增直到50，如果接收到消息的顶点集合为，此时设置，意味着每个映射区间长度为10，最终映射后的消息向量为。采用上述方式对消息量进行映射需要一个前提，即从图结构上相临的顶点应具有相近的顶点ID，这样才能保证落于同一区间的顶点会贡献相同的计算量。而这一前提在PIE编程模型下很容易满足，因为每个计算节点会对划分后子图顶点重新分配ID，该过程可通过广度优先遍历来完成。

**多阶段型算法特征提取**：与前两类算法不同的是，该类算法属于多阶段类型，但上节分析得出，即便处于多阶段类型，若干个阶段之间有相似的重复的规律，因此除上述提取的特征外，我们还需将处理后轮数做为新的特征，最终提取的特征如表12所示：

**表12 多阶段型算法特征提取**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 子图ID | 内部点数 | … | 边界点比重 | 消息映射向量 | 迭代轮数 |

### 模型选取

虽然我们可以将运行时间的预测定义成机器学习中的回归问题，但存在众多的回归模型需要我们进行分析，如局部加权线性回归模型、岭回归模型、树回归模型甚至使用神经网络模型，不同模型在训练的时间及效果上差异很大，因此结合图计算领域约束，我们期望找出每类算法适合的回归模型，并作出训练与预测。这里我们使用均方相对误差做为损失函数，公式如下：

我们之所以采用均方相对误差做为损失函数，是因为相对于均方绝对误差与均方误差而言，均方相对误通过将误差除以实际运行时间来降低慢任务的影响。例如，现有三个真实的运行时事件分别是20ms、50ms、120ms，预测的后的时间对应为15ms、40ms、100ms。那么如果采用均方误差来衡量模型的性能，得到的结果分别25、100、400，整个模型的MSE为171，可以看出，120ms的贡献了误差的绝大部份；相反，如果我们使用均方相对误差，得到的结果分别是0.25、0.2、0.17，数据之间的差别很小，因此我们可以更好的捕捉短时间的训练样本。下面，我们随机森林、岭回归、多元线性回归、和神经网络等预测模型进行讨论：

**多元线性回归模型**：多元线性回归模型作为最简单的回归模型，在许多场景都可以进行应用，该模型基本形式如下。

给定有m个属性描述的实体，其中代表在第个属性上的取值，多元线型回归模型试图学得一个通过各个属性的线性组合来进行预测的函数，该函数表示为：

一般用向量形式表示为：

其中，和学得之后，模型就可以确定下来，用于后续的预测分析。

多元回归线性模型形式简单，易于建模，因此对复杂模型的拟合程度不足，并且容易发生过拟合现象，但是在算法运行时间预测过程中，该回归模型仍然可以作为一个备选的预测模型，主要原因有以下几点：1) 由于直观表达了各属性在预测中的重要性，因此多元回归线性模型具有很好的可解释性；2) 任何函数均可由泰勒展开式逼近；3) 往往设计较好的算法程序运行时间是输入规模的多项式函数，因此完全可使用该模型进行回归分析及预测。

**岭回归：**岭回归是为了解决多元线性回归中过拟合现象而提出的另一种方法，更重要的是，岭回归模型可在数据的特征比样本点多的情况下进行回归分析。因此它特别适合图计算中实时运行时间的预测。不仅如此，岭回归模型还有如下优势：1) 在岭回归模型的预测过程中最多只需要几十次积累操作，因此其满足图计算的时间性能要求；2) 虽然其计算复杂度为，但其只依赖于特征向量的维度，因此对于图计算预测中特征向量维度或样本数据较小的情形下具有一定优势。

岭回归思想是去掉最小平方估计中的无偏性质，通过引入一个偏差量来降低估计的方差，具体而言，岭回归就是在矩阵上加一个从而使得矩阵非奇异，进而能对求逆。其中是单位矩阵，即对角线上元素全为1，其他元素全为0；而是一个用户定义的数值，通过该值来限制所求权重系数之和，通过引入该惩罚项，能够减少不重要的参数，整个技术在统计学中也叫缩减，最终岭回归模型的回归系数计算公式如下：

训练岭回归模型的关键是确定一个合适的值。实际中，通过预测误差最小化的方法得到值，具体做法为在获取数据之后，首先抽一部分数据用于测试，剩余的作为训练集用于训练参数。训练完毕后在测试集上测试预测性能。通过选取不同的来重复上述测试过程，最终得到一个使得预测误差最小的值。

**神经网络**：神经网络是一种运算模型，其基本工作单元被称为神经元，整个结构由大量的神经元相互连接构成，以模仿大脑神经元工作的机制。在神经网络中，每个神经元可以接收一个或多个输入，并产生一个输出。典型的神经网络一般可以分为输入层、隐藏层、和输出层，并且根据隐藏层的的层数进一步分为单隐层神经网络和多隐层神经网络；根据各层之间的连接关系又分为前馈神经网络和反馈神经网络等。由于神经网络对数据隐含关系的发掘具有很好的效果，因此被用于回归分析、分类等多个领域。但是相比于线性回归模型，神经网络虽然可能对数据进行更好的拟合，取得较小的误差，但是其最终模型无法对程序的特征属性与运行时间的关系给出合理的解释，因此本节使用神经网络仅仅比较了预测效果。

**随机森林**（参考文献）：包含了多个回归树，其中每个回归树构建时的样本都是由训练集经过有放回抽样得来的，并进行独立的预测。最终预测结果由各回归树投票决定。

随机森林模型简单、容易实现、计算开销小，并且它在很多现实任务中展现出强大的性能，因为该模型中学习器的多样性不仅来自于样本的波动，还来自属性选取过程中的扰动，这就使得最终集成学习的泛化性可通过个体学习器之间的差异度的增加而进一步提升。随机森岭模型不仅在高维度离散的数据数据下表现出很好的性能，其在以下方面同样特别适合图计算中运行时间与消息速率的预测：1) 对于一个包含个回归树的随机森林，其中所有回归树的平均深度是，最好与最坏情况下，其预测的复杂度分别是是和，因此其满足图计算得时间性能要求；2) [23]实验证明，大多数随机森林模型在训练复杂度上更趋近于最好情况而不是最坏情况，因此这为图计算的线上训练提供了可能性；3) 由于最终的预测取决于所有回归树的投票，因此对于训练数据的轻微变动具有很好的适应性，这为图计算预测的正确定提供了保证；4) 该模型只有一个超参数，即回归树的个数，因此该模型针对不同类别的图算法可能更具有通用型。

### 运行时间预测

**实验配置**：针对每种算法，我们采用离线的方式，按照算法对应分类的特征格式收集运行时信息作为训练测试的数据集，实验中，如果图数据划分为64个分区，共进行20轮迭代计算，则产生的样本数量为64\*20=1280个。不仅如此，如果某些算法可进行多问题查询，如单源最短路径算法，则样本数据量会更多，实验中这类算法随机选取10个查询点。上述回归模型的训练与预测的选取均使用Python Sklearn[]库完成，实验中，通过测试验证，随机森林模型设置回归数的个数等于10；神经网络设置为双隐层网络，且每层神经元个数均为50个，其余参数均采用该库的默认参数。除此之外，过程中采用交叉验证的方法。所有测试任务均在“天河二号”计算集群上进行，每台机器128G内存，2.40GHz。所有实验重复执行5次，并取平均值报告如下。

**表13 PageRank算法在ukweb数据集上预测分析**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **图分区数** | **样本数量** | **迭代计算时间（秒）** | **平均预测时间（秒）** | | | | **MSRE** | | | |
| **LR** | **RR** | **NN** | **RF** | **LR** | **RR** | **NN** | **RF** |
| 64 | 8315 | 0.17 | 4.20e-06 | 3.50e-06 | 8.36e-06 | 3.3e-04 | 0.007 | 0.007 | 2.27 | 0.007 |
| 96 | 9773 | 0.12 | 5.57e-06 | 5.53e-06 | 1.06e-05 | 5.2e-04 | 0.005 | 0.005 | 0.9 | 0.005 |
| 128 | 10368 | 0.13 | 5.67e-06 | 5.51e-06 | 1.27e-05 | 5.1e-04 | 0.005 | 0.005 | 1.98 | 0.005 |
| 160 | 13437 | 0.11 | 5.60e-06 | 5.35e-06 | 1.24e-05 | 5.1e-04 | 0.004 | 0.005 | 1.56 | 0.005 |
| 192 | 26880 | 0.12 | 3.16e-06 | 4.00e-06 | 7.52e-06 | 2.9e-04 | 0.007 | 0.007 | 0.93 | 0.007 |

**表14 Label Propagation算法在com-friendster数据集上预测分析**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **图分区数** | **样本数量** | **迭代计算时间（秒）** | **平均预测时间（秒）** | | | | **MSRE** | | | |
| **LR** | **RR** | **NN** | **RF** | **LR** | **RR** | **NN** | **RF** |
| 64 | 5760 | 19.91 | 5.51e-06 | 6.26e-06 | 1.14e-05 | 4.99e-04 | 0.014 | 0.014 | 0.092 | 0.014 |
| 96 | 8640 | 17.65 | 5.37e-06 | 5.16e-06 | 1.09e-05 | 4.87e-04 | 0.017 | 0.017 | 0.016 | 0.017 |
| 128 | 11520 | 15.28 | 5.44e-06 | 5.59e-06 | 1.19e-05 | 5.17e-04 | 0.018 | 0.018 | 0.019 | 0.018 |
| 160 | 14400 | 12.21 | 6.10e-06 | 6.68e-06 | 1.20e-05 | 4.81e-04 | 0.014 | 0.014 | 0.015 | 0.014 |
| 192 | 17280 | 10.47 | 6.33e-06 | 5.52e-06 | 1.16e-05 | 4.97e-04 | 0.015 | 0.015 | 0.015 | 0.015 |

**表15 SSSP算法在usa-road数据集上预测分析**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **图分区数** | **样本数量** | **迭代计算时间（秒）** | **平均预测时间（秒）** | | | | **MSRE** | | | |
| **LR** | **RR** | **NN** | **RF** | **LR** | **RR** | **NN** | **RF** |
| 64 | 2209 | 0.09 | 4.51e-06 | 3.47e-06 | 6.85e-06 | 2.44e-04 | 3152 | 2907.2 | 5000+ | 0.44 |
| 96 | 4513 | 0.043 | 3.36e-06 | 3.29e-06 | 6.87e-06 | 2.48e-04 | 1380 | 1403.5 | 5000+ | 4.27 |
| 128 | 6635 | 0.041 | 2.65e-06 | 3.07e-06 | 6.38e-06 | 2.32e-04 | 1192 | 1179 | 5000+ | 0.76 |
| 160 | 8651 | 0.032 | 2.92e-06 | 3.03e-06 | 6.44e-06 | 2.24e-04 | 1181 | 1170 | 5000+ | 1.67 |
| 192 | 11481 | 0.029 | 2.88e-06 | 3.65e-06 | 6.13e-06 | 2.23e-04 | 1087 | 1075 | 5000+ | 1.72 |

**表16 BFS算法在com-friendster数据集上预测分析**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **图分区数** | **样本数量** | **迭代计算时间（秒）** | **平均预测时间（秒）** | | | | **MSRE** | | | |
| **LR** | **RR** | **NN** | **RF** | **LR** | **RR** | **NN** | **RF** |
| 64 | 4849 | 2.05 | 1.15e-05 | 1.49e-05 | 2.75e-05 | 1.02e-03 | 0.98 | 1.06 | 2.61 | 0.18 |
| 96 | 8328 | 3.07 | 1.13e-05 | 1.47e-05 | 2.47e-05 | 1.04e-03 | 0.68 | 0.19 | 4.72 | 0.13 |
| 128 | 11787 |  | 1.10e-05 | 1.28e-05 | 2.66e-05 | 1.03e-03 | 6.02 | 7.09 | 1.56 | 0.18 |
| 160 | 20060 |  | 1.05e-05 | 1.27e-05 | 2.66e-05 | 2.24e-03 | 15.77 | 2.09 | 50.7 | 0.26 |
| 192 | 32755 |  | 1.18e-05 | 1.24e-05 | 2.60e-05 | 9.61e-04 | 6.85 | 12.76 | 1.22 | 0.39 |

**表18 WCC算法在ukweb数据集上预测分析**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **图分区数** | **样本数量** | **迭代计算时间（秒）** | **平均预测时间（秒）** | | | | **MSRE** | | | |
| **LR** | **RR** | **NN** | **RF** | **LR** | **RR** | **NN** | **RF** |
| 64 | 6582 | 2.05 | 1.15e-05 | 1.49e-05 | 2.75e-05 | 1.02e-03 | 0.98 | 1.06 | 2.61 | 0.18 |
| 96 | 11328 | 3.07 | 1.13e-05 | 1.47e-05 | 2.47e-05 | 1.04e-03 | 0.68 | 0.19 | 4.72 | 0.13 |
| 128 | 19287 |  | 1.10e-05 | 1.28e-05 | 2.66e-05 | 1.03e-03 | 6.02 | 7.09 | 1.56 | 0.18 |
| 160 | 27260 |  | 1.05e-05 | 1.27e-05 | 2.66e-05 | 2.24e-03 | 15.77 | 2.09 | 50.7 | 0.26 |
| 192 | 40755 |  | 1.18e-05 | 1.24e-05 | 2.60e-05 | 9.61e-04 | 6.85 | 12.76 | 1.22 | 0.39 |

**表19 Sampling算法在dbpeida数据集上预测分析**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **图分区数** | **样本数量** | **迭代计算时间（秒）** | **平均预测时间（秒）** | | | | **MSRE** | | | |
| **LR** | **RR** | **NN** | **RF** | **LR** | **RR** | **NN** | **RF** |
| 64 | 2560 |  | 1.74e-05 | 2.07e-05 | 3.08e-05 | 1.15e-03 | 5000+ | / | 5000+ | 0.35 |
| 96 | 3840 |  | 1.23e-05 | 2.01e-05 | 2.77e-05 | 1.09e-03 | 5000+ | / | 5000+ | 0.32 |
| 128 | 4490 |  | 1.21e-05 | 1.71e-05 | 1.89e-05 | 1.16e-03 | 5000+ | / | 5000+ | 0.64 |
| 160 | 6400 |  | 1.15e-05 | 1.94e-05 | 2.90e-05 | 1.l7e-03 | 5000+ | / | 5000+ | 1.01 |
| 192 | 7690 |  | 1.19e-05 | 1.68e-05 | 3.04e-05 | 1.09e-03 | 5000+ | / | 5000+ | 2.83 |

**表20 HITS算法在ukweb数据集上预测分析**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **图分区数** | **样本数量** | **迭代计算时间（秒）** | **平均预测时间（秒）** | | | | **MSRE** | | | |
| **LR** | **RR** | **NN** | **RF** | **LR** | **RR** | **NN** | **RF** |
| 64 | 9671 |  | 4.45e-06 | 3.27e-06 | 8.76e-06 | 3.48e-04 | 0.13 | 0.20 | 39.56 | 0.13 |
| 96 | 12959 |  | 4.00e-06 | 4.29e-06 | 9.06e-06 | 3.66e-04 | 0.11 | 0.35 | 17.59 | 0.10 |
| 128 | 19097 |  | 3.48e-06 | 3.52e-06 | 8.83e-06 | 3.65e-04 | 0.14 | 0.39 | 28.55 | 0.13 |
| 160 | 39118 |  | 2.45e-06 | 2.22e-06 | 5.48e-06 | 2.09e-04 | 0.11 | 0.22 | 35.45 | 0.13 |
| 192 | 57580 |  | 1.86e-06 | 2.05e-06 | 4.62e-06 | 1.75e-04 | 0.11 | 0.16 | 48.35 | 0.09 |

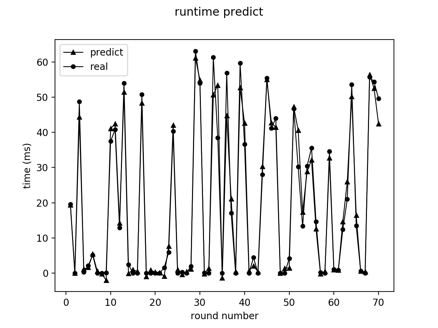
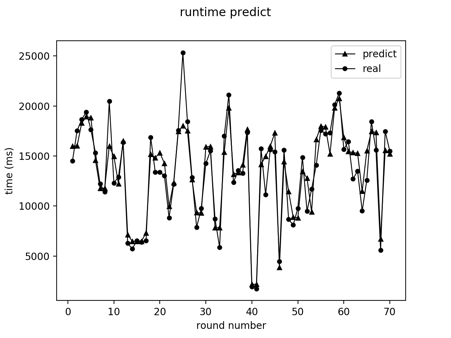
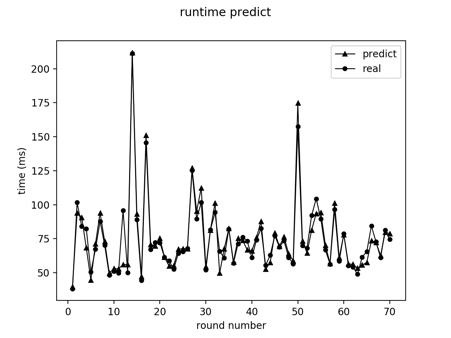
**表21 Louvain算法在上预测分析**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **图分区数** | **样本数量** | **迭代计算时间（秒）** | **平均预测时间（秒）** | | | | **MSRE** | | | |
| **LR** | **RR** | **NN** | **RF** | **LR** | **RR** | **NN** | **RF** |
| 64 | 4920 |  | 3.61e-05 | 3.82e-05 | 3.21e-05 | 1.03e-03 | 0.56 | 0.53 | 32.83 | 0.22 |
| 96 | 8820 |  | 3.83e-05 | 3.27e-05 | 2.99e-05 | 1.06e-03 | 0.49 | 0.48 | 29.22 | 0.21 |
| 128 | 13272 |  | 3.28e-05 | 2.92e-05 | 3.97e-05 | 1.07e-03 | 0.46 | 0.46 | 38.99 | 0.19 |
| 160 | 18700 |  | 3.93e-05 | 3.29e-05 | 3.46e-05 | 1.l7e-03 | 0.51 | 0.39 | 35.17 | 0.18 |
| 192 | 30380 |  | 3.02e-05 | 3.01e-05 | 3.25e-05 | 1.02e-03 | 0.43 | 0.33 | 33.46 | 0.18 |

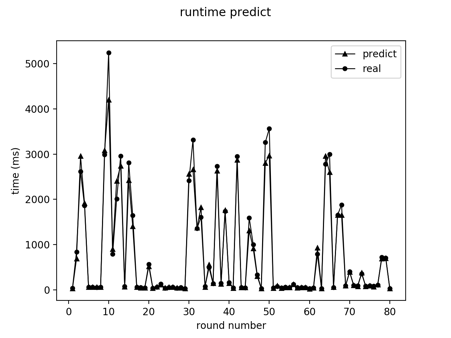
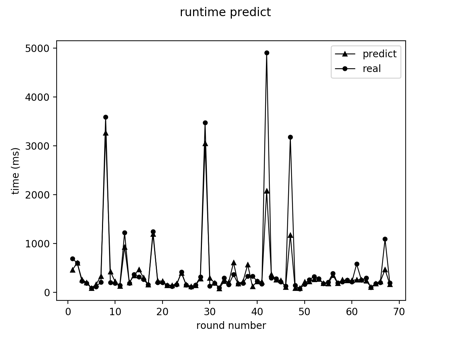
**持续活跃型算法**：表13与表14分别为PageRank算法和Label Propagation算法的预测结果，我们可以看出1) 所有回归模型的预测时间均远小于每轮迭代的计算时间，因此很好地满足了图计算的时间约束；2) 除神经网络外模型外，岭回归、随机森林、多元线性回归都具有较低的均方相对误差，因此这些模型的预测十分准确。

**消息依赖型算法**：表16-19分别为单源最短路径、广度优先搜索、连通性、和图采样算法在对应数据集上的预测分析结果，我们可以看出：随机森林模型（RF）在所有算法上均保持较低的误差范围，而神经网络（NN）模型在样本数量较少的单源最短路径算法以及图采样算法上具有较高的均方相对误差；同样，岭回归（RR）和多元线性回归模型（LR）在单源最短路径算法上具有较高的误差，甚至岭回归在图采样算法中无法完成预测过程，因此随机森林模型在小样本集合上仍具有很好的预测效果。

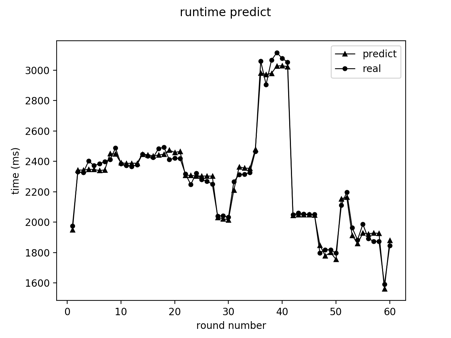
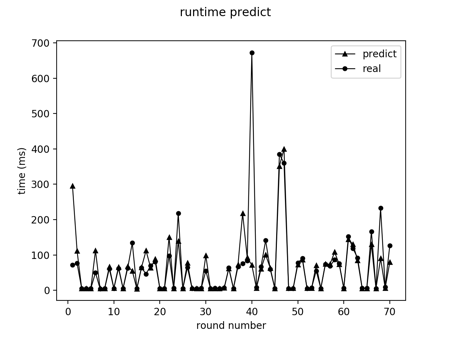
**多阶段型算法**：表20和表21分别为HITS算法与社区发现Louvain算法在对应数据集上的预测分析结果，我们可以看到，除神经网络模型（NN）外，其余回归模型具有较低的均方相对误差。

****

**(a) PageRank (b) Label Propagation (c) SSSP**

****

**(d) BFS (e) WCC (f) Sampling**

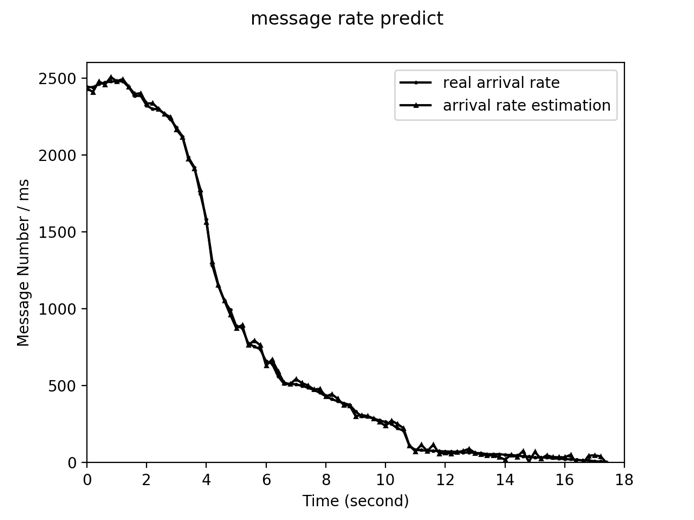
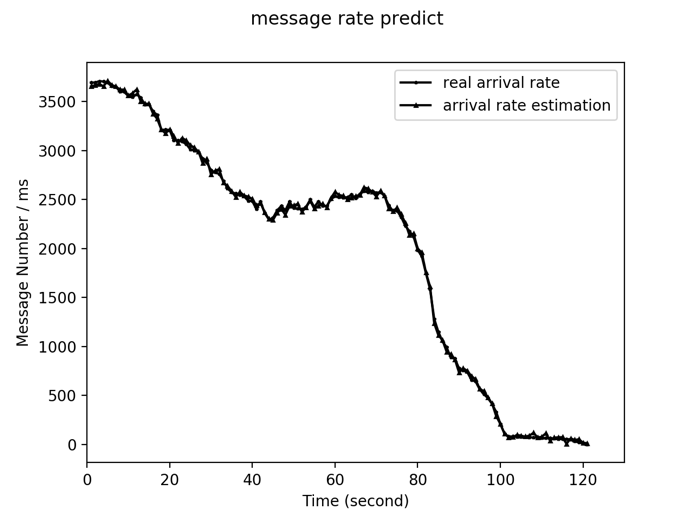
****

**(g) HITS (h) Louvain**

**图25算法运行中70轮迭代预测结果**

我们最终得出，持续活跃型算法由于其迭代过程的计算量主要取决于图结构特征，如内部点个数，而不受消息传递所影响，因此所有模型在这类算法上均具有较低的误差范围；与续活跃型算法不同，消息依赖型算法迭代过程中的计算量主要取决对活跃点集合情况，因此在样本数据较少的情况下，神经网络、岭回归、多元线性回归均具有较大的误差范围，但随机森林模型仍具有较好的预测效果；由于我们选用的HITS和Louvain多阶段算法在每个阶段均倾向于持续活跃型，因此上述回归模型表现出较低的误差范围。为进一步展示随机森林模型的预测效果，图24展示了上述算法在随机抽取70轮迭代的预测结果，我们可以看出大部分的预测结果都接近这是值，但在运行时间相对较大的数据点上，如PageRank算法大于125ms和Label Propagation大于17500ms的预测不是非常准确，主要原因是受限于在这一时间范围内的样本量。

## 消息到达速率预测

****

**(a) Label Propagation (com-friendster) (b) SSSP (usa-road)**

**图25消息到达速率预测结果**

我们采用“最近的平均预测不远的将来”思想来预测消息到达速率，即我们计算时间内所收集的消息数量，并用 来表示下一轮的消息达到速率。之所以使用这种方式，主要原因如下：1）我们发现，某个子图接收消息的速率，不仅跟与之相连的其它子图的个数有关，还与其它子图中前一轮计算的活跃点集合相关，甚至还与当时网络延迟等多方面不可控因素相关，因此无法采用机器学习的方式提取特征，从而达到想训练时间那样训练消息到达速率，2）RPAP模型下消息的到达速率随时间的减小为趋于连续平稳变化，几乎没有凸起值，因此这种最近的平均预测不久的将来思想确实有效，同样我们也用实验的方式验证了其有效性3）即便有很多已有的模型如神经网络模型可以很精确的预测，但这些模型有很昂贵的部署代价，相反，我们的做法代价极低。图25分别为Label Propagation算法与单源最短路径算法在对应数据集上消息到达速率的预测结果，实验中取值为100ms，我们可以看出预测值非常接近真实值，且真实的到达速率曲线连续，没有凸起值，因此我们预测消息到达速率的思想可行。

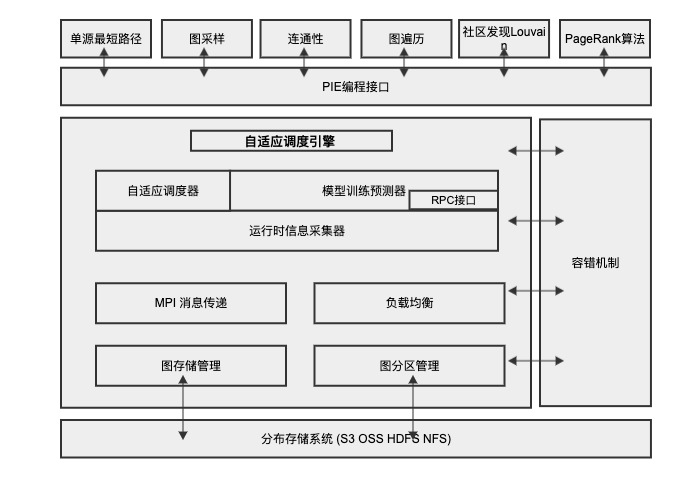
## 本章小节

本章首先选用了工业界与学术界经典的8种图应用算法，通过分析算法在PIE编程模型下的运行时行为，将算法分为持续活跃型、消息依赖型、和多阶段型三类，并针对每类算法，给出了影响图计算运行时间的特征向量提取方案；其次，本章通过机器学习回归技术和“最近的平均预测不远的将来”思想分别对上述算法的运行时间和消息到达速率进行预测，实验结果表明，随机森林模型在运行时间预测方面普遍具有较低的均方相对误差和较好的预测准确性；我们预测消息达到速率的思想也十分合理，且预测效果十分准确。

# 基于运行时间预测的图计算系统的设计与实现

我们在GRAPE+系统的基础上，扩展开发了基于运行时间预测的自适应图计算系统GRAPE++，本节将从系统的架构、图存储结构、消息传递机制等方面给出系统的详细设计与实现。

## 系统架构概况

****

**图25 GRAPE++系统架构图**

GRAPE++系统的结构如图25所示。从图中可以看出GRAPE++系统整体分为数据存储、核心组件、系统编程接口、内置应用算法实现、和容错机制五大部分。

其中数据存储层表示GRAPE++系统处理的源数据存储形式，这些数据可以是存储在某些系统文件中，也可以存储于如HDFS、Aws S3、Aliyun OSS等分布式存储系统或云存储环境中。无论采用何种数据源存储方式，都必须保证数实现据源I/O的相应接口。

数据存储层上面是核心组件层，包括图分区管理、图存储管理、负载均衡管理、MPI消息传递机制、自适应调度器、算法运行时信息采集器、和模型预测等7大组件，这些组件构成了整个系统最核心的部分。其中图分区管理组件用于完成系统的载图工作，因为GRAPE++作为分布式图计算系统，因此需要对源数据进行图分区操作，这一工作也同样在图分区管理组件内完成。目前该组件支持哈希图分区与块分区两种图分区方式，前者根据构建哈希表来决定图中顶点所属分区，后者则根据图中顶点ID均匀的将顶点划分若干块，每块中的顶点属于一个分区；图存储管理组件主要负责每个分区在内存中组织对应的图数据，因为分布式图应用算法会频繁的遍历图中顶点及顶点的一度出(入)边，因此GRAPE++系统与临接表的形式存储图数据；负载均衡管理组件主要解决图中超大度数顶点的问题，往往有些图数据，某些顶点的度数非常高，如果仅根据哈希或块分区方式完成图数据的划分，则该顶点所属的分区往往负载较其余分区大，因此该组件在完成图数据划分后，根据每个分区的具体情况进行顶点重划分，以达到图计算负载均衡的目的；MPI消息传递机制主要负责各节点间消息通讯；自适应调度器与模型预测组件实时交互，共同负责各计算节点独立的控制本节点图计算进程；算法运行时信息采集器则负责图计算过程中，运行时特征的收集工作，收集到的数据既可用于模型的训练，也可用于接下来运行时间的预测。

GRAPE++系统仍然采用GRAPE+系统的编程接口，即PIE接口，任何图应用算法若想在GRAPE++系统上运行，均需完成对PEval、IncEval、Assemble函数的开发。

最上层是GRAPE++系统的应用算法层，目前该系统支持单源最短路径、随机游走图采用等8个内置算法。

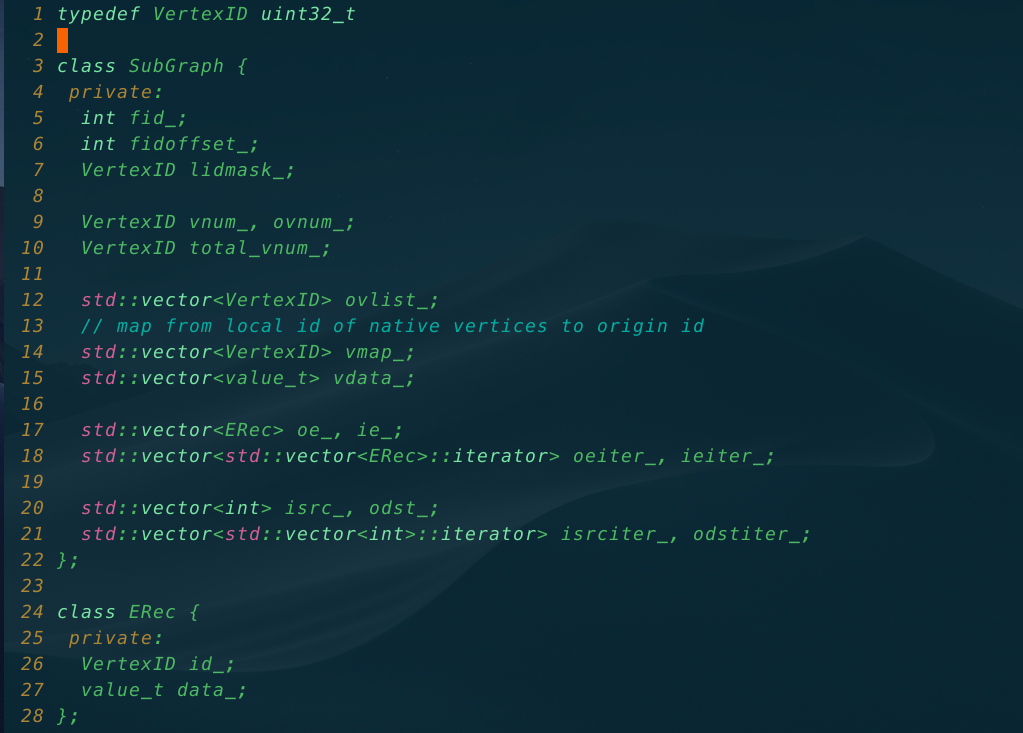
## 模块设计与实现

本节详细介绍GRAPE++系统核心组件的设计与实现，包括图存储结构、消息传递机制、自适应调度控制、一致性保证、和容错方案。

### 图存储结构

通常，图数据的存储需要考虑图的有向性，由于无向图的存储可以看作是对称的有向图存储，因此GRAPE++系统采用临接表的形式存储图数据，这种方式结构简单，存储开销相对较小，并能很好的按顺序访问图数据，图5-3为GRAPE++系统图边分区后的子图存储结构。

图5-4 图存储结构实现

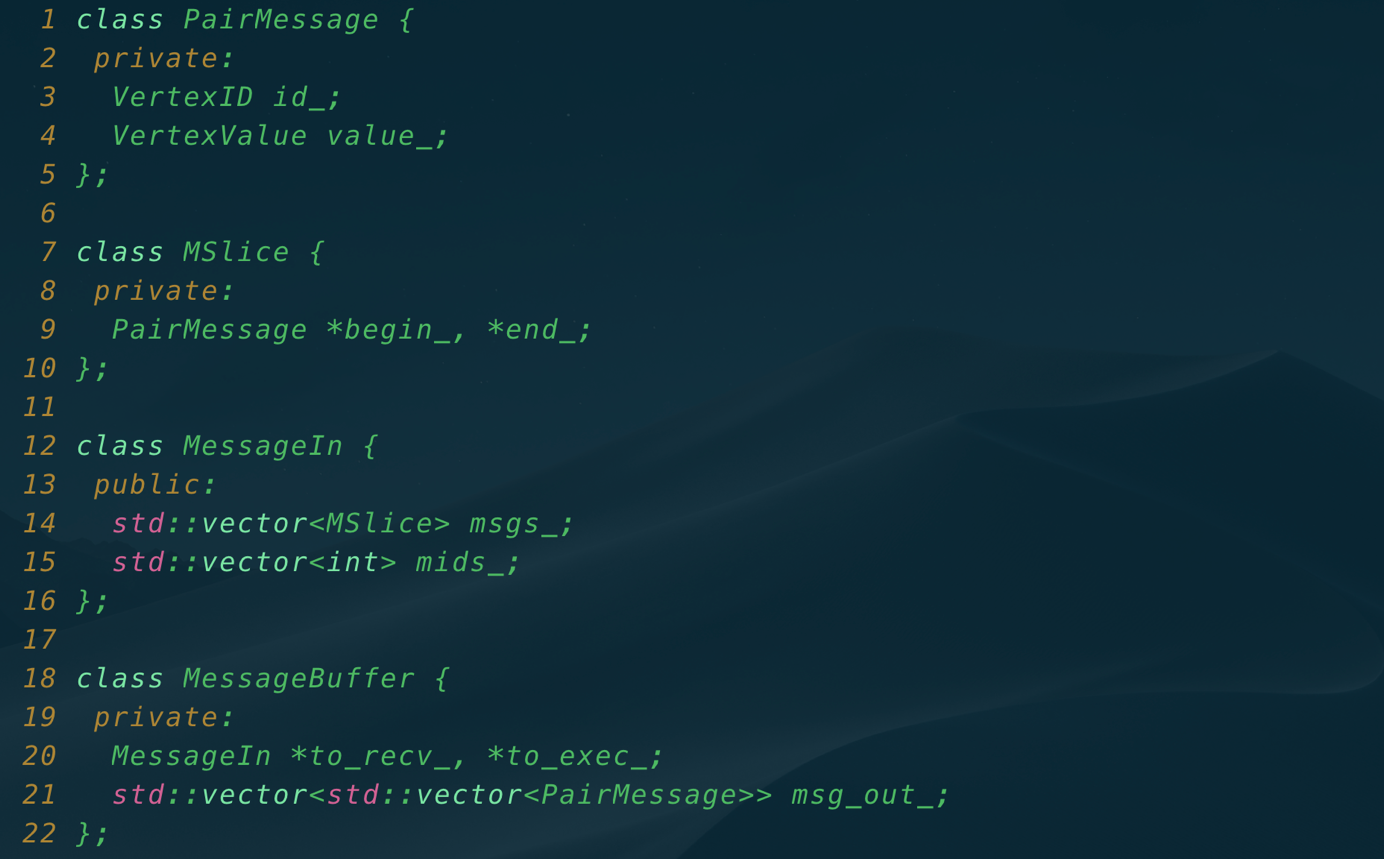


因为GRAPE++采用边分割当时进行图分区操作，因此每个子图中的点包含了该子图的内部点和与该子图直接相连的其余子图的外部点，后者对应图中的ovlist\_数据结构，所有顶点的数据存储在vdata\_中。另外，图数据中每个顶点都有一个原始ID，我们称之为oid，通常为从1开始的自然数逐渐递增。但在GRAPE++系统中，分区后的子图需重新定义图节点ID，我们称之为lid，以简化用户编程接口，因此需要存储lid与oid之间的映射，对应图中vmap\_数据结构；图中oe\_和ie\_分别存储该子图的所有出边与入边，并通过oeither\_(ieiter\_)迭代器完成对特定点的出边(入边)的遍历操作。

### 消息传递机制

GRAPE++系统采用异步的方式进行消息传递，即计算节点可在任意时刻将消息通过MPI函数发送给目标节点，同时。每个计算节点时刻接收来自其余各节点的消息，该过程贯穿于分布式图计算的始终。因此我们定义消息结构如下：

图5-5消息结构实现

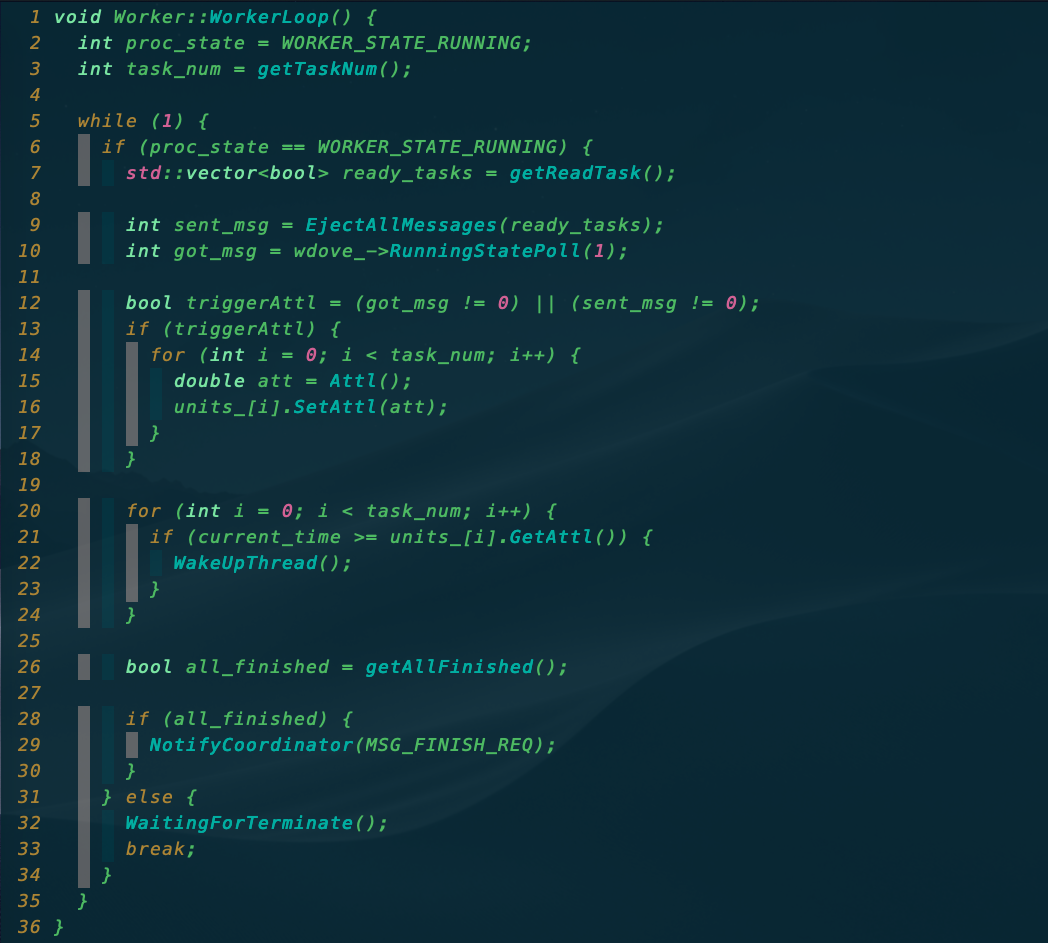


其中MessageBuffer为我们最终实现的消息存储结构，msg\_out\_数据结构用于存储该节点发送的消息；to\_recv\_结构用于存储该节点接收的消息；由于接收消息发生在图计算的任意时刻，因此需要to\_exec\_结构负责原子性的交换to\_recv\_中的数据，用于下一轮迭代计算。PairMessage是消息结构，包含消息作用的顶点ID和对应的消息值。

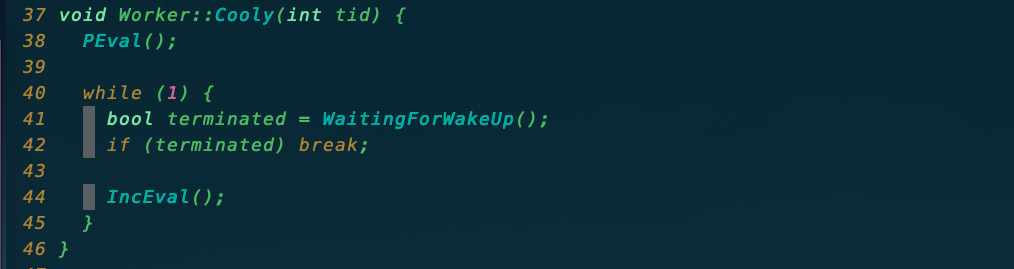
### 自适应调度控制

图5-2为GRAPE++系统的工作流图。其中Coordinator节点不存储图数据，也不参与图计算任务，其主要负责收集各工作节点的运行时信息，并监听各节点的指令信息；Worker节点主要参与分布式图计算工作，并通过与Coordicator节点的通信，获取其余各节点的执行状态。

自适应调度控制工作于Worker节点，每个计算节点需要维护两个进程，其中进程执行过程如图所示，用于执行用户开发的PIE图算法函数，每完成一次迭代计算，该进程便进入睡眠状态，等待被唤醒，继续执行下一轮迭代；进程负责该计算节点的消息发送与接受工作，并根据预测的结果，选择性地唤醒进程，主要程序框架如下图所示：



图中WorkerLoop函数即对应线程所做的工作，首先初始化当前工作状态，并获取当前Worker的任务数目，即该计算节点上分配的子图数量（第2行）；接着进入循环过程，循环过程中，如果该节点的工作状态处于RUNNING状态（第6行），则进行消息的接受与发送工作（第9-10行），根据接受与发送消息的结果，执行自适应预测函数，设置每个任务的等待时间（第12-18行），如果满足当前等待时间，唤醒PIE执行函数进程，执行分布式图计算（第20-24行），最后根据任务的完成情况，通知Coordinator节点（第26-30行），后续或重复上述过程，或接收到Coordinator的终止确认，结束本次任务（第32-33行）。



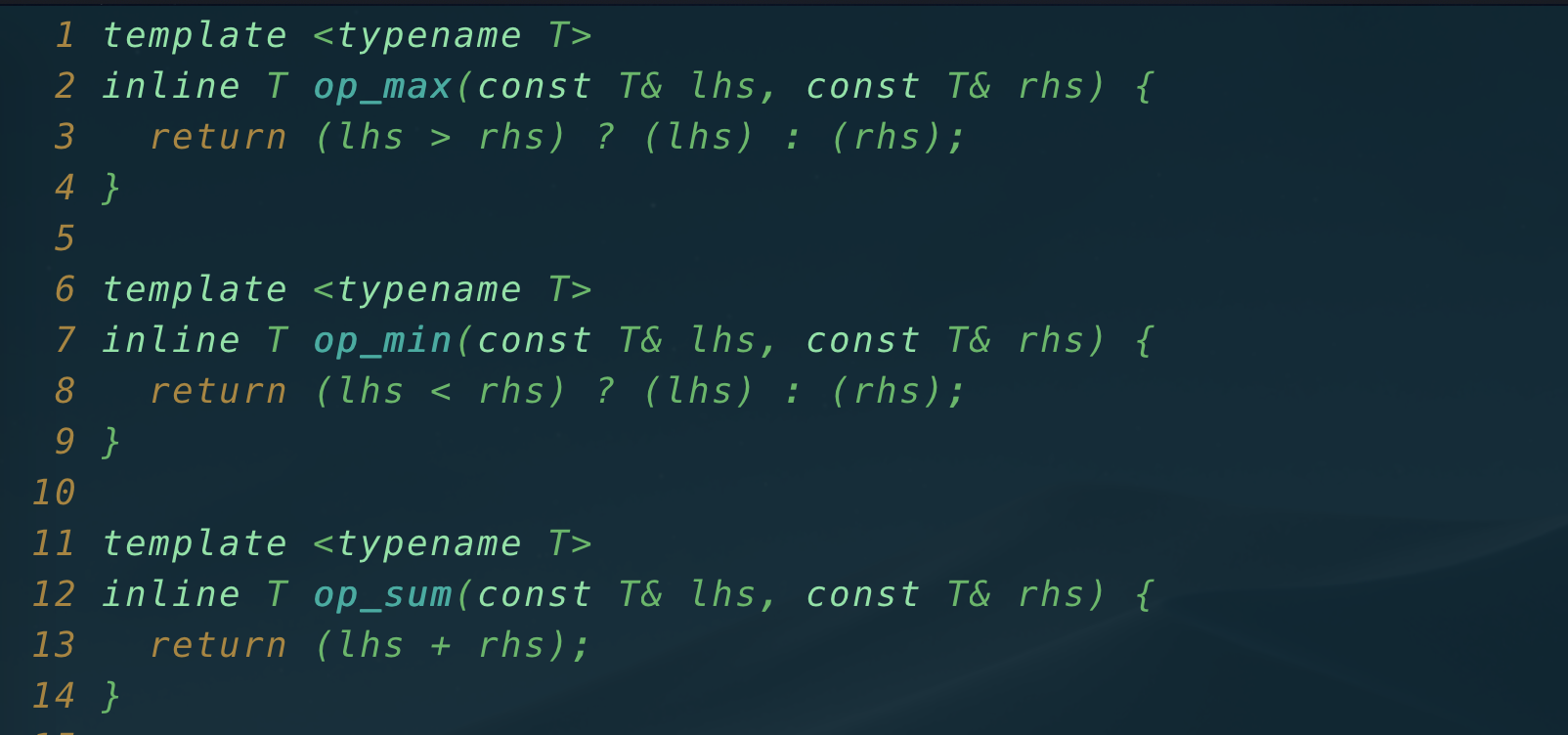
### 容错方案与一致性保证

**容错方案**：GRAPE++系统采用快照记录和故障恢复两个过程完成容错操作。具体操作为，当图数据载入过程完成后，每个计算节点会通过序列化操作将所负责的子图写到分布式文件系统中形成元快照，这些信息包括子图中的所有点（内部点和外部点）以及和这些点相关的边信息。为了避免写同一个文件时出现资源竞争，每个计算节点会写在单独的文件中，文件名是根据本次计算生成唯一标识符；在图计算执行过程中，每个节点需要将节点的当前状态记录到分布式文件系统中，因为GRAPE++系统本质属于异步计算模型，因此采用Chandy-Lamport算法[grape论文20]记录一个一致性的全局快照。同时，为进一步降低快照引入的额外开销，GRAPE++系统选择每若干轮迭代计算记录一次快照，具体数目由用户通过参数传递决定。此外，由于部分图应用算法各计算点收敛时间不同，因此GRAPE++系统支持增量式快照记录，即每次记录时，只记录若干轮迭代中参与计算的节点状态，状态没有变化的顶点则不在本次快照记录当中。

GRAPE++系统的Coordinator节点会定期以心跳的方式监测各Worker节点的当前状态，如果在系统运行过程中某个Worker节点出现故障，Coordinator节点会从集群中找出一个空闲机器来代替故障机器，并广播所有Worker节点发生故障，需要进行恢复工作。GRAPE++系统的恢复工作包含以下步骤：1) 故障节点从元快照中读取图结构信息，反序列化完成载图工作；2) 根据增量化的记录，依次完成图计算的中间数据恢复 3) 由于其余顶点包含了故障节点的副本信息，因此恢复后的节点向其余各节点发送当前数据状态，保证系统中所有节点恢复自己负责的副本节点。

一致性保证：如果每个节点通过维护一个消息缓冲区来接收其它计算节点的消息，会给系统带来额外的存储开销，因此GRAPE++系统采用聚合函数解决同一顶点接收来自不同计算节点更新消息的竞争问题，该过程为原子操作。目前实现的聚合函数包括Max、Min、Sum函数等，代码如图4-5所示：

图4-5 聚合函数



## 系统性能评测

本小节我们首先介绍测试使用的输入数据集与环境配置，然后我们针对上述图应用算法，将基于运行时间预测的自适应计算模型同传统自适应计算模型、同步计算模型、异步计算模型进行对比，从而对GRAPE++系统的整体性能表现做一个测试与总结。

### 环境配置

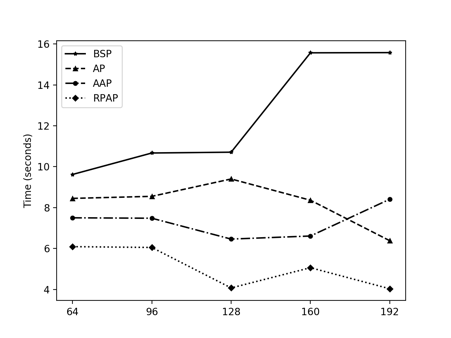
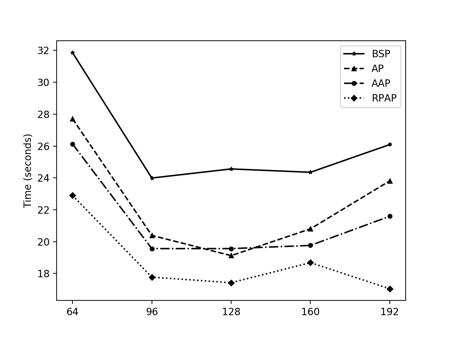
表5-5列出了我们在评测中使用的输入数据集。其中1) Friendster数据集为社交网络图，包含6500万个用户实体和18亿条边，我们在边上随机的赋予权重以测试单元最短路径算法；Usa-road数据集是美国公路网络图，包含2300万个顶点及5800万条边；Uk-web数据集是一个Web图，包含1.33亿个顶点及50亿条边；Livejournal数据集同样是一个社交网络图包含480万个顶点和6800万条边。所有的图数据均可通过互联网进行下载[<http://snap.stanford.edu/data/soc-LiveJournal1.html>]。

我们使用XtraPulp[grape65]默认配置的分区结果作为GRAPE++系统的图分割策略；为了对比短板效果下的计算性能，我们随机挑选了某一个计算节点，并人为的制造慢机环境，即将对应节点睡眠若干倍计算时间，实验中为3倍。

为验证结果的正确性，我们将GRAPE++系统在RPAP模型下的执行结果分别与同步模型与异步模型的执行结果进行对比，其中由于部分数据挖掘算法在本质上为非确定性算法，每次执行生成的结果会有少量偏差，因此我们输入参数和收敛精度相同的情况下，比较这些算法的计算结果，并确定结果的偏差值在容差范围内。

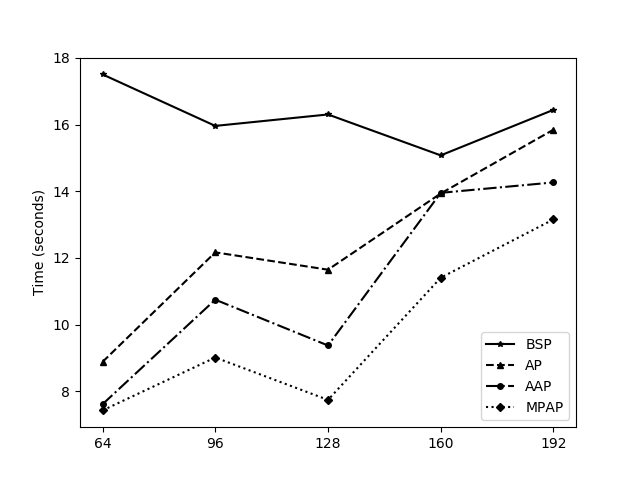
所有测试任务均在“天河二号”计算集群上进行，每台机器128G内存，2.40GHz。图计算的每个计算节点独占一台机器，所有实验重复执行5次，并取平均值。

### 实验结果

****

**(a) sssp-com-friendste (b) sssp-usa-road**

**图33 SSSP算法性能评估**

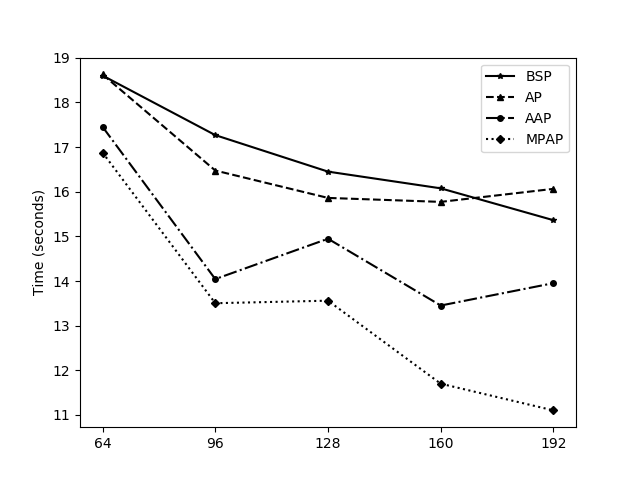
****

**(a) com-friendster (b)**

**图33 BFS算法性能评估**

图33展示了在64分区到192分区配置下，基于运行时间预测的自适应模型（RPAP）与同步（BSP）、异步（AP）、传统自适应计算模型（AAP）在不同图算法、图数据集上的性能对比。实验结果表明，RPAP模型整体表现优于其它计算模型，其中相对同步计算模型性能提高2.3倍（最低1.9倍），对于异步计算模型性能提高2.4倍（最低1.1倍），对于传统自适应计算模型性能提高2.8倍（最低1.1倍）。这些性能的提升主要来自于模型准确的时间与消息到达速率预测以及各计算节点及时的根据当前状态动态调整本节点的进度。下面进一步分析不同图应用算法的性能表现。

**BFS算法**：如图33所示，在BFS算法以及com-friendster图数据上，RPAP模型分别在64-fragment、96-fragmnet、128-fragment、160-fragment、192-fragment比BSP模型快57%、43%、52%、24%、19%；比AP模型快16%、24%、36%、18%、16%；比AAP模型快2%、16%、17%、18%、7%；可以看到RPAP模型比BSP于AP模型具有较大的性能优势，与AAP原型相比，也具有一定的性能优势，但在64分区下优势不明显。

****

**图34 HITS算法性能评估**

**HITS算法**: 如图34所示，在hits算法以及ukweb图数据上，RPAP模型分别在64-fragment、96-fragmnet、128-fragment、160-fragment、192-fragment比BSP模型快9.28%、21.8%、17.57%、27.22%、27.73%；比AP模型快9.4%、18%、14.5%、25.8%、30.9%；比AAP模型快3.3%、3.84%、9.28%、13%、20.4%；可以看到RPAP模型比BSP于AP模型具有较大的性能优势，与AAP原型相比，也具有一定的性能优势。

## 本章小节

本章在GRAPE+系统的基础上，扩展开发了基于运行时间预测的自适应图计算系统GRAPE++，并从从系统的架构、图存储结构、消息传递机制等方面给出GRAPE++系统的详细设计与核心代码实现。并针对前文分析的8种图应用算法在不同的数据集上，对GRAPE++系统的性能进行测试，实验结果表明，我们的计算模型的整体性能优于同步、异步、以及传统自适应计算模型。

总结与展望

## 论文总结

大数据与云计算的背景下，分布式图计算系统的蓬勃发展为大规模图数据处理提供了可能。而本文通过分析了分布式图计算系统中，主流计算模型的优缺点，创新性的提出了RPAP基于运行时间预测的自适应计算模型，并在GRAPE+系统的基础上，扩展实现了支持运行时间预测自适应模型的GRAPE++系统。此外，本文在8个工业界与学术界常用的经典图应用算法，对GRAPE++系统的计算性能进行了全面的评测，GRAPE++系统相对与同步、异步、和传统自适应的调度模式，获得了xx、xx、xx的性能提升。本文主要创新与贡献如下：

1. 提出了RPAP基于运行时间预测的自适应图计算模型，该模型在图划分不均匀、慢机等情况下，充分避免了同步、异步模型下的存在问题，具有明显的性能优势。

2. 本文选用了8个工业界与学术界常用的经典图应用算法，通过分析算法在PIE编程模型下的运行时行为，将算法分为持续活跃型、消息依赖性、多阶段型三类。针对每类算法，给出不同的运行时特征信息提取方案，并选用多种机器学习回归模型对算法在图迭代计算过程中的运行时间进行训练预测，实验结果表明，随机森林模型相比于岭回归、神经网络、多元线性回归模型具有较低的误差范围和较好预测效果。

3. 本文通过扩展开发GRAPE++系统，给出了RPAP自适应计算模型与同步、异步、传统自适应模型在不同图应用算法上的性能评测，进一步验证了RPAP计算模型在系统实验和性能优势方面的可行性。

## 未来工作展望

为使得并行图计算系统更好的满足日益增长的数据需求，本文从图计算的计算模型角度，通过基于时间预测的自适应调度，对图计算的性能进行了优化。但相比于调度模式，影响图计算系统性能还包括网络通信、图分区效果、底层图存储结构与索引构建等多个方面，这些都是今后值得去深入研究的方向。

参考文献

1. San Murugesan, Irena Bojanova, Mark Smiley. An Introduction to Big Data[M]. 2016.
2. Dolinski, K, Troyanskaya, O. G. Implications of Big Data for cell biology[J]. Molecular Biology of the Cell, 26(14):2575-2578.
3. Kum, Hye-Chung, Krishnamurthy, Ashok, Machanavajjhala, Ashwin. Social Genome: Putting Big Data to Work for Population Informatics[J]. Computer, 47(1):56-63.
4. Iacus, Stefano M. Big Data and Social Science - A Practical Guide to Methods and Tools[J]. Journal of Statistical Software, 2017, 078.
5. Zhu, Ting, Zhang, Qingquan, Xiao, Sheng. Big data in future sensing[J]. International Journal of Distributed Sensor Networks, 2015, 2015.
6. Marcus Paradies, Hannes Voigt. Big Graph Data Analytics on Single Machines – An Overview[J]. 2017, 17(1):1-12.
7. Yu Lu, Fei Hu, Xin Li. Towards the Security of Big Data: Building a Scalable Hash Scheme for Big Graph[M]// Information Technology - New Generations. 2018.
8. Sun, Yongjiao, Li, Boyang, Yuan, Ye. Big graph classification frameworks based on Extreme Learning Machine[J]. NEUROCOMPUTING, 2019.
9. Poulovassilis A. Database research challenges and opportunities of big graph data[M]// Big Data. 2013.
10. Balaji J , Sunderraman R . Scalable storage structure for pattern matching on big graph data[C]// 2015 IEEE International Conference on Big Data (Big Data). IEEE, 2015.
11. Quoc-Dinh Truong, Taoufiq Dkaki, Quoc-Bao Truong. Graph Methods for Social Network Analysis[C]// 2nd EAI International Conference on Nature of Computation and Communication - ICTCC 2016. 2016.
12. Pržulj N. Protein-protein interactions: making sense of networks via graph-theoretic modeling.[J]. Bioessays News & Reviews in Molecular Cellular & Developmental Biology, 2011, 33(2):115-123.
13. <http://tinkerpop.apache.org/>
14. <https://neo4j.com/developer/cypher-query-language>
15. Gregor D, Lumsdaine A. Lifting sequential graph algorithms for distributed-memory parallel computation[C]// Acm Sigplan Conference on Object-oriented Programming. ACM, 2005:423-437.
16. Dean J , Ghemawat S . MapReduce: simplified data processing on large clusters[M]. ACM, 2008.
17. Apache Software Foundation. Apache Hadoop [EB/OL]. <https://hadoop.apache.org/> , 2013
18. Bu Y, Howe B, Balazinska M, et al. HaLoop: efficient iterative data processing on large clusters[J]. Proceedings of the Vldb Endowment, 2010, 3(1-2):285-296.
19. Valiant L G . A bridging model for parallel computation[J]. Comm Acm, 1990, 33(8):103-111.
20. Malewicz G, Austern M H, Bik A J C, et al. Pregel:a system for large-scale graph processing[C]// ACM SIGMOD International Conference on Management of Data. ACM, 2010:135-146.
21. Slota G M , Rajamanickam S , Devine K , et al. Partitioning Trillion-edge Graphs in Minutes[C]// Parallel & Distributed Processing Symposium. IEEE, 2017.
22. Hendrickson B, Leland R. A multilevel algorithm for partitioning graphs[C]// Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference. IEEE, 1995:28.
23. Spielman D A, Teng S H. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems[C]// 2004:81-90.
24. Wenfei Fan, Jingbo Xu, Yinghui Wu,等. GRAPE: parallelizing sequential graph computations[J]. Proceedings of the VLDB Endowment, 2017, 10(12):1889-1892.
25. Gemini
26. Salihoglu S , Widom J . GPS: a graph processing system[J]. 2013.
27. Apache Giraph. <http://giraph.apache.org/>
28. Apache Hama. <http://hama.apache.org/>
29. Tian Y , Balmin A , Corsten S A , et al. From "Think Like a Vertex" to "Think Like a Graph"[J]. Proceedings of the VLDB Endowment, 2013, 7(3):193-204.
30. Xin R S, Gonzalez J E, Franklin M J, et al. GraphX:a resilient distributed graph system on Spark[C]// International Workshop on Graph Data Management Experiences and Systems. ACM, 2013:1-6.
31. Yan D , Cheng J , Xing K , et al. Pregel Algorithms for Graph Connectivity Problems with Performance Guarantees[C]// Proceedings of the VLDB Endowment. VLDB Endowment, 2014.
32. Bao N T , Suzumura T . Towards highly scalable pregel-based graph processing platform with x10[C]// International Conference on World Wide Web Companion. 2013.
33. Khayyat Z , Awara K , Alonazi A A , et al. Mizan: A system for dynamic load balancing in large-scale graph processing[C]// Acm European Conference on Computer Systems. ACM, 2013.
34. Da YAN, James Cheng, Yi Lu,等. Blogel: A Block-Centric Framework for Distributed Computation on Real-World Graphs[C]// Proceedings of the VLDB Endowment. 2014.
35. Zhao S . Grace: Graph-Constrained Estimation and Hypothesis Tests[J]. 2017.
36. <https://www.open-mpi.org/>
37. <https://zookeeper.apache.org/>
38. <https://netty.io/>
39. Gonzalez J E, Low Y, Gu H, et al. PowerGraph: distributed graph-parallel computation on natural graphs[C]// Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2012:17-30.
40. SU Jing, SUO Bo, CHEN Qun, et al. GraphHP:A hybrid platform for iterative graph processing[J]. Journal of East China Normal University, 2016.
41. Fan, Wenfei, Xu,LuPing, XiaoJian, Luo, et al. Adaptive Asynchronous Parallelization of Graph Algorithms[J]. Proceedings of the SigMod, 2018, 6(10): 1141-1156
42. Zhang, Yanfeng, Gao, Qixin, Gao, Lixin,等. Maiter: An Asynchronous Graph Processing Framework for Delta-Based Accumulative Iterative Computation[J]. IEEE Transactions on Parallel and Distributed Systems, 25(8):2091-2100.
43. Low, Yucheng, Gonzalez, Joseph E, Kyrola, Aapo,等. GraphLab: A New Framework For Parallel Machine Learning[J].
44. Chen, Rong, Shi, Jiaxin, Chen, Yanzhe,等. PowerLyra[J]. ACM Transactions on Parallel Computing, 2019.
45. GuoZhang Wang, Wenlei Dai, Alan J. Demers, and Johannes Gehrke, 2013. Asynchronous Large-Scale Graph Processing Made Easy. In CIDR.
46. Xie C, Chen R, Guan H, et al. SYNC or ASYNC: time to fuse for distributed graph-parallel computation[C]// Acm Sigplan Symposium on Principles & Practice of Parallel Programming. ACM, 2015:194-204.
47. Ho Q, Cipar J, Cui H, et al. More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server[J]. Advances in Neural Information Processing Systems, 2013, 2013(2013):1223.
48. Sara Karamati, Jeffrey Young, Richard Vuduc. An Energy-Efficient Single-Source Shortest Path Algorithm[C]// 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2018.
49. Ishii, Hideaki, Tempo, Roberto. The PageRank Problem, Multiagent Consensus, and Web Aggregation: A Systems and Control Viewpoint[J]. IEEE Control Systems, 34(3):34-53.
50. Farahat, Ayman, LoFaro, Thomas, Miller, Joel C,等. Authority Rankings from HITS, PageRank, and SALSA: Existence, Uniqueness, and Effect of Initialization[J]. SIAM Journal on Scientific Computing, 27(4):1181-1201.
51. Dengwang Tang, Vijay G. Subramanian. Balanced Allocation on Graphs with Random Walk Based Sampling[C]// 2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton). 2018.
52. Ning Liu, Xueli Yu, Li Yao,等. Mapping the Cortical Network Arising From Up-Regulated Amygdaloidal Activation Using -Louvain Algorithm[J]. IEEE Transactions on Neural Systems and Rehabilitation Engineering, 2018, PP(99):1-1.
53. DU Li-ping, SUN Jia-si. Research on Optimizing Ramps for Group-division Interchange Based on Dijstra Algorithm[J]. Technology & Economy in Areas of Communications, 2009.
54. Yogesh Simmhan, Alok Kumbhare, Charith Wickramaarachchi, et al. GoFFish: A Sub-Graph Centric Framework for Large-Scale Graph Analytics[J]. 2013.
55. Aapo Kyrola. Large-scale Graph Computation on Just a PC[C]// Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation. 2012.
56. Amitabha Roy, Ivo Mihailovic, Willy Zwaenepoel. X-Stream: Edge-centric graph processing using streaming partitions[M]. 2013.
57. C. Zhou, J. Gao, B. Sun,等. MOCgraph: Scalable distributed graph processing using message online computing[J]. Proceedings of the VLDB Endowment, 2014, 8(4):377-388.
58. Bin Shao, Haixun Wang, Yatao Li. Trinity: A distributed graph engine on a memory cloud[C]// Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, 2013.
59. Wentao Han, Youshan Miao, Kaiwei Li,等. Chronos: A graph engine for temporal graph analysis[C]// EuroSys 2014. 2014.
60. Z Yang, J Xue, Z Qu, et al. Seraph: An Efficient System for Parallel Processing on a Shared Graph[J]. 2013.
61. Li-Yung Ho, Tsung-Han Li, Jan-Jan Wu,等. Kylin: An efficient and scalable graph data processing system[C]// Big Data, 2013 IEEE International Conference on. IEEE, 2013.
62. Yubin Bao, Zhigang Wang, Yu Gu,等. BC-BSP: A BSP-Based Parallel Iterative Processing System for Big Data on Cloud Architecture[C]// International Conference on Database Systems for Advanced Applications. Springer Berlin Heidelberg, 2013.
63. Yang Liu, Bin Li, Kai-kun Dong, et al. Reaching of topology partition for parallel network simulation[C]// 2010.
64. Tianshu Bi, Yixin Ni, C.M. Shen,等. An efficient graph partition method for fault section estimation in large-scale power network[C]// Power Engineering Society Winter Meeting, 2001. IEEE. IEEE, 2001.
65. Hutter, Frank, Xu, Lin, Hoos, Holger H,等. Algorithm runtime prediction: Methods & evaluation[J]. Artificial Intelligence, 206:79-111.
66. Slota G M , Madduri K , Rajamanickam S . PuLP: Scalable multi-objective multi-constraint partitioning for small-world networks[C]// IEEE International Conference on Big Data. IEEE, 2015.

攻读硕士学位期间取得的学术成果

**硕士期间发表的论文**

1. **Jiang Kun**, Hu chunming, Sun Jie, Shen Qi. Email importance evaluation in mailing list discussions [C]. International Conference on Web Information Systems Engineering. 2018. (CCF推荐会议列表C类，workshop，12页)
2. **Jiang Kun**, Hu chunming.：Characterizing and predicting important emails in mailing list discussions [C]. 北京航空航天大学第十五届研究生学术论坛. 2018.

**硕士期间申请的专利**

[1] 胡春明, **蒋坤**, 何潇安. 一种基于会话网络的邮件重要性评价方法[P]. CN：201811441956.X，2018-11-29.(专利学生第一发明人)

致 谢

在北京