**北京航空航天大学计算机学院**

**学位论文开题报告**

**论文题目**：基于机器学习预测的并行图计算系统

**专 业**：计算机技术

**研究方向**：分布式图计算系统

**研 究 生**：李东泽

**学 号**：ZY1706212

**指导教师**：樊文飞

**北京航空航天大学计算机学院**

2018年11月18日

目 录

1 论文选题的背景与意义 1

2 国内外研究现状及发展动态 2

2.1 同步计算模型 2

2.2 异步计算模型 3

2.3 混合计算模型 5

2.4 自适应计算模型 5

3 论文的研究内容及拟采取的技术方案 6

3.1 本课题的目标与主要研究内容 7

3.2 拟采用的技术方案 8

4 关键技术与难点 9

5 论文研究计划 9

6 主要参考文献 10

**基于机器学习预测的并行图计算系统**

# 论文选题的背景与意义

图计算是以“图论”为基础的对现实世界的“图”结构的抽象表达，以及在这种数据结构上的计算模式。在实际的应用中，存在着许多图计算问题，如图的连通性[1]、单源最短路径[2]、网页排序[3]等，同样在社会或科学领域中的许多问题都可以通过转换成图计算模型从而得到解决。例如: Google公司每天都需要对数亿级别的网页进行排序[4]; 生物医学上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[5]；社会安全方面更需要对整个社交网络进行精准的模式匹配，从而增加寻找贩毒、犯罪集群的可能性。因此，对图数据的分析和计算具有很重要的现实意义。

早期的图计算数据规模较小，因此大部分使用BGL等单机图处理算法[6]，极大的限制了图计算问题的规模，后来随着图规模的增长，出现了Parallel BGL[7]等并行图计算系统，他们实现了很多并行图计算算法，但是系统本身不具备良好的稳定性和容错性，同样对于部分算法也没有提供很好的正确性保证。后来，随着大数据时代的来临，大图数据的分析与计算已经远远超出了单台计算机的存储和计算能力，给图分析计算带来了极大的挑战。于此同时，MapReduce[8]作为一个并行的计算框架，及其开源实现Hadoop[9]为海量的数据处理提供了便利，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迭代，各轮迭代涉及顶点间的复杂联系，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。

针对上述图计算的特点，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，面向大规模图计算处理的研究吸引了越来越多的研究者投身其中。最近，在计算模型和运行时优化方面涌现出不少具有前瞻性的技术和图计算系统，代表性的大规模系统包括Giraph[10]、GPS[11]、GRACE[12]、GraphLab[13]、PowerSwitch[14]、GRAPE+[15]。这些图计算系统在计算模型上分别属于同步模型、异步模型、混合模型以及自适应模型，它们围绕着减少迭代次数加快算法收敛、减少消息传递数目减轻网络负载、实现负载均衡消除木桶效应这3个优化目标，分别采用一种或多种优化技术来对自身进行优化，但是它们往往都是在优化某个目标的同时牺牲了其它性能，因此本课题以自适应并行图计算模型为基础，采用基于机器学习模型的预测技术，综合上述三个优化目标，实践开发出更有效的并行图计算系统。

# 国内外研究现状及发展动态

## 同步计算模型

1990年，图领奖获得者Valiant提出了BSP(bulk synchronous parallel)计算模型[16]，这种模型区分于现有的MapReduce计算框架在于其适合做数据的迭代计算。其核心思想是将每一次迭代称为一个超步(super-step)，整个图计算划分为多个超步，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。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2.1展示了同步模型的整体计算流程。



图2.1 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[17]。在该计算模型中，用户只需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，涉及处理上轮迭代的更新消息、本地计算、消息传递等逻辑。其具体执行过程如下: 首先，选定一个图分区策略[18-20]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。同样采用上述BSP模型的分布式图计算框架还有GPS[11]、Giraph[10]以及GRAPE[21]等系统。

虽然BSP模型极大的简化的图计算任务，充分利用了网络流量，但是该模型仍有以下几方面限制：

(1) 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，严重影响图计算效率。

(2) 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) 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]，这类问题对于贪心的解决方案在BSP模型下不会收敛，因为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。

(4) 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，并分配到不同的计算节点上，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。

## 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2.2所示。



图2.2 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，异步模型在执行效率以及资源的利用率方面相对与于同步BSP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[12]，但是异步模型也同样存在着以下方面的不足之处：

(1) 虽然异步模型可以加速图计算的收敛，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，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[22]采用分布式锁的形式来保证数据的一致性，而GRAPE+[15]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[25]。

## 混合计算模型

Hybrid Model计算模型[14]，简称Hsync模型。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2.3所示



图2.3 混合计算模型

[14]的实验结果表明，PowerSwitch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。但这种计算模型在每轮计算后，需要重新预测下一轮的计算模型，且需要解决模型间切换的数据一致性问题，因此产生了额外的内存与时间开销。

## 自适应计算模型

Adaptive Asynchronous Parallel[15]计算模型，简称AAP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。但和之前模型不同的是，AAP模型下每个计算节点通过保留一些参数来测量本节点相对于其它计算节点的迭代进度，并根据当前已接收的消息量决定是否立即进入下轮计算或等待一段时间后再进入下轮计算，上述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。AAP模型不同与其它计算模型主要体现在以下几个方面：

(1) BSP模型：AAP模型通过异步的消息传递机制来消除BSP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) AP模型：AAP模型通过引入限定值Delay Stretch使得各个计算节点自己决定每一轮迭代需等待的时间，聚集更多的消息从而进行更有效的计算，因此很大程度上减少了AP模型下的冗余计算问题。

(3) Hybrid Model模型：相对于Hsync模型，AAP模型不需要全局的预测开关在BSP和AP模型进行切换，相反，AAP模型允许每个计算节点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BSP模式，与此同时，其它节点采用AP模式。除此之外，由于AAP模型动态调整计算节点间的相对进程，因此减少了模型间切换所带来的而外开销。

# 论文的研究内容及拟采取的技术方案

在自适应的并行图计算系统中，为解决同步模型下的木桶效应以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，在接下来迭代计算中，各节点需等待这么长的时间以方便积累更多的消息。其中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：

(1) ：表示计算节点当前接收的消息数量。直观上来看，如果 越大，则当前计算节点接收消息越多，应该尽快进入下一轮迭代计算。

(2) ：表示当前图计算的所有计算节点中，最小与最大的迭代轮数，其中，每个计算节点都会存储本节点当前的迭代轮数，它们共同控制各计算节点的相对进度。

(3) ：用于决定计算节点是否需要立即进入等待状态。例如，如果某计算节点当前迭代轮数且，其中c是预先设置的限定值(表示最快的节点不能快于最慢节点c轮)，则，表示该计算节点需等待直到最慢节点完成本轮计算。

(4) ：表示预测本轮迭代计算应该接受的消息量。该变量定义如下：，其中可由用户自定义初始化，为了尽快减少冗余计算；为预测的消息到达速率；是 的一部分，后者为预测的运行时间。

(5) ：表示计算节点为尽可能的多接收消息而等待的时间，定义为；表示计算节点前一轮计算后的空闲时间，以防止无限期的等待。

经过上述分析，在自适应的并行图计算系统中，对于任何一个计算节点，只需预测下一轮的运行时间 (running time)以及消息到达速率(arrival rate)，便可计算出延迟计算的等待时间，以保证计算前积累更充分的消息，从而减少迭代轮数，加快收敛速度。但是目前的系统中，运行时间与消息到达速率的预测是基于统计连续轮数的数据完成的，其主要缺点有以下2个方面：

(1) **没有充分利用图计算的历史数据**：目前已有的预测模型中，下一轮的运行时间与消息达到速率是通过统计分析前几轮的运行数据而获得的，因此每轮的预测只能基于本次图计算的各轮数据进行分析，而无法充分利用已完成图计算的历史数据，使得预测的准确性很低。

(2) **没有充分利用图算法的特征:** 针对不用的图算法，往往预测分析的角度不同，例如：PageRank、CF(Collaborative filtering)等算法，其在迭代计算的过程中，每轮的计算量与消息传递量相差很小甚至完全相同；而SSSP(Single Source Shortest Path)、WCC(Weakly Connected Component)等算法，在迭代计算的执行期间，活跃顶点的数量会随着计算轮数的增多呈现出先上升后下降的趋势，因此对运行时间及消息到达速率的预测造成了重大的挑战。而目前的预测模型并没有考虑到不同图算法之间的差异，采用上述统一的方法进行分析，使得最终的预测效果不佳。

## 本课题的目标与主要研究内容

**本课题目标**：针对自适应并行图计算系统中现有预测模型的弊端，本课题采用基于机器学习技术对运行时间与消息的到达速率进行更有效预测，加速图数据的并行计算。首先，我们将运行时间与消息到达速率的评估定义成机器学习中的回归问题，我们采用均方相对误差作为评价指标；其次，我们选取机器学习中一些常见的算法时间预测模型作为候选，进行比较，从而分析各个模型在图计算预测问题上的优缺点；最后我们选取出一种或多种模型并针对多种不同类型的图算法进行训练与预测。本课题具体研究内容如下：

(1) 首先我们将运行时间与消息到达速率的预测定义成回归问题，我们以运行时间的预测为例：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) 针对众多的回归模型设计并训练出一个适合的预测器。需要特别指出，由于迭代过程中的估值阶段只执行一次，因此我们只针对接下来的迭代增量阶段进行运行时间与消息到达速率的预测。

(2) 我们需要针对线性回归、决策树回归、支持向量机回归、岭回归、随机森林回归、K近邻回归、Adaboost回归、梯度增强随机森林回归、神经网络、高斯过程回归等众多回归模型进行分析与比较，筛选出适合算法运行时间预测的回归模型，并最终选取出1种回归模型进行预测。

(3) 在预测的过程中，我们还需对图应用算法进行合理的分类，针对不同种类的图应用算法给出模型的训练方法以及参数的调控方向。

(4) 根据已选定的预测模型，开发并实现出基于该模型预测的自适应并行图计算系统，并提供前端界面展示功能。

## 拟采用的技术方案

# 关键技术与难点

通过上文的介绍与分析，本科的主要难点包括：

(1) **图应用算法的分类**：目前图计算领域的重要算法包括图遍历(深度优先、广度优先)、单源最短路径、连通性、子图匹配、最小生成树、页面排序、协同过滤、标签传播等。本课题从分布式图计算角度出发，会对上述算法从query、消息传递、本地计算以及图数据特性等不同方向分析，并将图算法进行分类，给出每一类的预测模型或模型的调参方向。

(2) **收集运行时信息**：收集运行时信息的过程及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。

(3) **图计算约束**：除了机器学习领域中过拟合与预测的准确性外，图计算领域还关心额外几个指标：a) 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，就没有任何意义；b) 训练数据约束，某些图应用算法中，可能无法采取线下训练的方式，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；c) 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计算模型。

(4) **模型的选取**：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，但如上节所述，存在众多的回归模型需要我们进行分析，判断出是否适合运行时间的预测，并针对一种或多种模型进行合理的训练与预测。

# 论文研究计划

2018年11月 – 2019年1月：在并行分布式图计算引擎上完成众多图应用算法的开发，并将图算法进行的分类。

2019年2月 – 2019年4月：测试、分析并选出适合运行时间与消息到达速率预测的回归模型。

2019年5月 – 2019年6月：针对每类图算法，进行模型的训练与预测，撰写小论文。

2019年7月 – 2019年9月：针对不同种类的图算法，完成模型的训练与预测，并与同步计算模型、异步计算模型进行实验对比，迭代调优。

2019年10月 – 2020年1月：整理资料，撰写毕业论文。

# 主要参考文献

[1] <https://en.wikipedia.org/wiki/Connected_component_(graph_theory)>

[2] Frigioni D, Marchetti-Spaccamela A, Nanni U. Fully dynamic output bounded single source

shortest path problem[C]// Acm-Siam Symposium on Discrete Algorithms. Society for

Industrial and Applied Mathematics, 1996:212-221.

[3] Brin S, Page L. The anatomy of a large-scale hypertextual Web search engine[J]. Computer

Networks, 2012, 56(18):3825-3833.

[4] Google. How search works. <http://www.google.com/insidesearch/howsearchworks/thetory>/

[5] 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.

[6] Siek J G, Lee L Q, Lumsdaine A. The boost graph library: user guide and reference manual[M].

Addison-Wesley Longman Publishing Co. Inc. 2002.

[7] 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.

[8] Dean J , Ghemawat S . MapReduce: simplified data processing on large clusters[M]. ACM, 2008.

[9] Apache Software Foundation. Apache Hadoop [EB/OL]. <https://hadoop.apache.org/> , 2013

[10] Apache Giraph. <http://giraph.apache.org/>

[11] Salihoglu S , Widom J . GPS: a graph processing system[J]. 2013.

[12] Spielman D A, Teng S H. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems[C]// 2004:81-90.

[13] Bickson D. GraphLab: Asynchronous Graph Computation in the Clouds and Beyond[J]. Archives of Biochemistry & Biophysics, 2006, 452(2):138-148.

[14] 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.

[15] Fan, Wenfei, Xu,LuPing, XiaoJian, Luo, et al. Adaptive Asynchronous Parallelization of Graph Algorithms[J]. Proceedings of the SigMod, 2018, 6(10): 1141-1156

[16] Valiant L G . A bridging model for parallel computation[J]. Comm Acm, 1990, 33(8):103-111.

[17] 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.

[18] Slota G M , Rajamanickam S , Devine K , et al. Partitioning Trillion-edge Graphs in Minutes[C]// Parallel & Distributed Processing Symposium. IEEE, 2017.

[19] Hendrickson B, Leland R. A multilevel algorithm for partitioning graphs[C]// Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference. IEEE, 1995:28.

[20] Spielman D A, Teng S H. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems[C]// 2004:81-90.

[21] Fan, Wenfei, Xu, Jingbo, Wu, Yinghui, et al. GRAPE: parallelizing sequential graph computations[J]. Proceedings of the VLDB Endowment, 2016, 10(12):1889-1892.

[22] Bickson D. GraphLab: Asynchronous Graph Computation in the Clouds and Beyond[J]. Archives of Biochemistry & Biophysics, 2006, 452(2):138-148.