華中科技大學

本科生毕业设计

面向大规模并行计算系统和网络的动态自适应编码传输方案研究

院　　系	计算机科学与技术
专业班级	本硕博 2001
姓　　名	李茗畦
学　　号	U202015630
指导教师	胡燏翀

2024 年 5 月 29 日

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名： 2024 年 5 月 29 日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保 密□，在 年解密后适用本授权书

2、不保密☑。

（请在以上相应方框内打“√”）

作者签名： 2024 年 5 月 29 日

导师签名： 2024 年 5 月 29 日

摘 要

随着数据量的增加以及各领域对计算能力的需求的不断增长，单个计算机的计算的性能已无法满足需求，因此常常通过多个计算机并行计算来提高计算性能。但是大规模并行计算系统常常发生故障频发的现象，会导致链路拥塞从而降低网络传输性能，导致系统的计算性能的下降，并且这一现象是伴随着网络环境动态变化的。因此需要让大规模并行计算系统具有对动态网络环境的自适应性。

设计并实现了基于 MapReduce 的并行计算原型系统。在此基础上应用编码分布式计算技术，通过编码传输的方式来降低数据传输阶段的通信开销。并实现了由在线学习驱动的动态编码分布式计算方案，通过执行微实验的方式探查对编码分布式计算方案进行动态调整的收益，向性能提升的方向对方案进行调整，在动态网络环境中实现计算开销和通信开销之间的平衡，从而提升整个计算过程的效率。

分别在基于离散型均匀分布，指数分布和 Pareto 分布的动态网络环境中对方案的性能进行了测试。测试结果表明，相比 MapReduce 方案和静态的编码分布式计算方案，由在线学习驱动的动态编码分布式计算方案可以在动态网络环境下具有更好的自适应性，更好地实现计算开销和通信开销之间的平衡。在计算性能方面，相比 Mapreduce 方案提升了约 50%，相比静态的编码分布式计算方案提升了约 30%。

关键词：并行计算；动态网络环境；自适应算法；网络编码

Abstract

With the increasing in data volume and the growing demand for computational power in various fields, the performance of individual computers is no longer sufficient to meet the requirements. Therefore, parallel computing using multiple computers is often employed to enhance computational performance. However, large-scale parallel computing systems often suffer from frequent failures, leading to network congestion and degraded network transmission performance, thereby reducing overall system computational performance. Moreover, this phenomenon dynamically changes with the network environment. Therefore, an adaptive solution is necessary to make parallel computing systems adaptable to dynamic network environments.

A prototype system based on the MapReduce parallel computing model is designed and implemented. Building upon this, coded distributed computing scheme are applied to reduce communication overhead during the data transmission phase. A coded distributed computing scheme driven by online learning is implemented. Through micro-experiments, the benefits of dynamically adjusting the coding-based distributed computing scheme are explored. Adjustments are made towards performance enhancement, achieving a balance between computational overhead and communication overhead in dynamic network environments, thereby improving the efficiency of the computation process.

The performance of the scheme is tested in network environments based on discrete uniform distribution, exponential distribution, and pareto distribution respectively. The result indicates that, compared to MapReduce scheme and static coding-based distributed computing schemes, the proposed scheme achieves a higher balance between computational overhead and communication overhead in dynamic network environments. In terms of computational performance, the proposed scheme improves by approximately 50% compared to the MapReduce scheme and approximately 30% compared to static coding-based distributed computing schemes.

Keywords: Parallel Computing,Dynamic Network Environment,Adaptive Algorithm, Network Coding

目 录

1 绪 论

本章首先介绍了大规模并行计算系统中在动态变化的网络环境中运行时遇到的问题和困难，介绍了本课题的工作意义，以及可能遇到的一些挑战。之后介绍了并行计算领域内的研究现状，主要是关于优化网络开销，实现负载均衡以及应对动态变化的网络环境等方面的相关技术和研究现状。最后介绍了本课题的主要内容和论文的结构。

1.1 课题背景、目的与意义

1.1.1 研究背景与趋势

本课题来自国家重点研发计划重点专项项目《大规模并行计算系统的可靠性编码理论和技术研究》。随着各种领域对计算能力的需求不断增长，单个计算机的串行计算的性能已无法满足需求，因此需要通过并行计算来提高计算效率。大规模并行计算系统由许多计算节点组成，这些节点同时进行计算任务以实现高性能和高吞吐量的计算。系统中的计算节点通过网络互连，彼此之间可以进行通信和协作，并行地执行计算任务，每个节点负责处理一部分数据。常见的并行计算框架有 MapReduce[1][2]等。

1.1.2 面临的问题和挑战

大规模并行计算系统会出现故障频发现象，导致系统的网络联通度降低。会频繁出现链路拥塞、丢包现象，降低系统的网络传输性能。这会导致进行运算时一些必须的数据和信息的传输延迟增加，同时节点之间的调度和同步等行为也变得不稳定，使得系统中执行失败的任务数量增加。因此网络的故障频发最终会导致系统的计算效率下降。这一过程的示意图如图 1-1 所示。并且由于动态变化的网络环境，这一现象也将是动态变化的。在并行计算过程中，数据传输阶段常常成为性能的瓶颈。Facebook 的 Hadoop[3][4]集群中，有 33%的作业执行时间都花费在了数据洗牌阶段[5]。同样，在亚马逊 EC2 集群上运行 Self-Join 应用程序时，数据洗牌也占用了 70%的执行时间[6]。

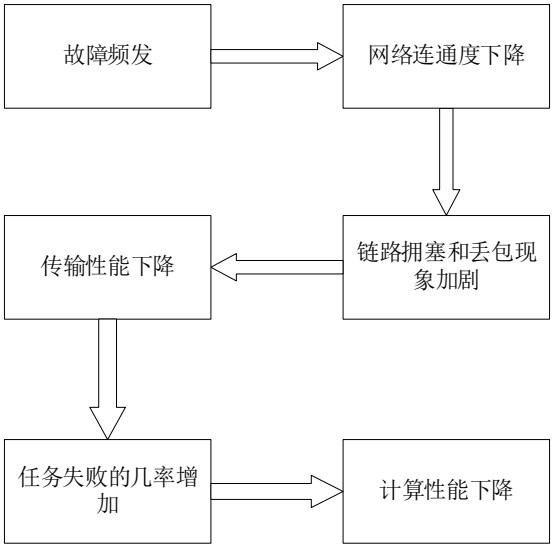

图 1-1 故障频发导致计算性能下降示意图

1.1.3 课题目的与意义

本课题旨在设计一个方案可以最大化地利用系统中现有的网络资源和计算资源，自适应地应对动态变化的网络负载情况，从而加快分布式并行计算的效率。在本课题的研究中，需要在动态变化的网络环境对并行计算系统的计算资源和网络资源进行监视和评估，通过一些策略降低系统的通信开销，并且根据动态变化的网络环境自适应地调整通信开销。方案的关键和挑战在于如何评估并行计算系统的计算开销和通信开销，以及如何根据网络环境对系统进行动态的调整。

1.2 国内外研究现状

1.2.1 并行计算系统中关于编码的研究现状

MapReduce 是为了满足海量数据的计算需求而提出的一种通用计算架构，用户需要提供一个 Map 函数和 Reduce 函数来完成计算任务。为了处理 MapReduce 中通信开销较大的问题，编码分布式计算 CDC（coded distributed computing）[7] 利用分布式计算中额外的算力换取通信能力，它阐述了 MapReduce 架构中计算开销和通信开销的权衡，即二者成反比关系。通过在 Map 阶段将计算开销增加 r 倍（即在 r 个精心选择的节点上执行每个 Map 函数）可以在 Shuffle 阶段创造出编码多播的机会，从而将通信开销也减少 r 倍。

为了避免在分布式学习中运行矩阵乘法时少数滞后节点对整体运算性能的

影响，编码计算（coded computation）[8]以类似纠删码的方式在分布式计算中引入冗余任务，利用一部分节点的运行结果解码得到全部的计算结果，避免少数滞后节点的影响。对于 n 个具有相同计算时间分布的节点，编码计算可以使矩阵乘法的执行速度快了 O（logn）倍。

为了优化分布式学习中执行数据洗牌的性能，编码洗牌（coded shuffling）[8]牺牲了一部分存储空间，在多轮洗牌的间隔中，选择性的保留节点缓存中的冗余数据，使得节点在下一轮洗牌时可以通过解码的方式获得本轮的全部数据，这种洗牌方式使通信开销减少 O(γ(n))倍(γ 为将 n 个消息单独发送与广播给 n 个节点的开销之比)。

[9]提出了扩展 CDC 方案来处理分布式计算中的滞后节点的问题的方案。该方案将 CDC 方案与 MDS 编码（maximum distance separable code）应用在计算任务上，从而实现在计算延迟和通信负载之间的平衡。在分布式计算中应用 MDS 编码可以创建冗余的 Map 任务避免滞后节点的影响。[9]把 CDC 和 MDS 编码计算统一为一个应用于矩阵乘法的通用方案。单一的应用 MDS 或 CDC 编码计算可以视作该方案的两个极端。

1.2.2 并行计算系统中关于负载均衡的研究现状

MapReduce 并行计算系统中，在基于任务队列的调度策略下，一个 Map 任务会按照其进入任务队列的顺序被依次分配给空闲的 Worker 节点。这种任务调度策略实现简单，易于编程。然而，在数据分布不均匀的情况下，这种策略会导致很多 Map 任务无法在本地执行，需要从其他节点读取任务所需的数据，引入额外的网络通信开销，降低任务执行效率。[10]提出了一种基于已知数据分布的任务调度策略，该策略充分考虑系统中数据的分布情况，将 Map 任务优先调度给拥有该任务所需数据的节点执行。在 MapReduce 任务开始之前，它首先需要收集每个节点上存储的数据块的信息，这通过从分布式文件系统（例如 HDFS[11]）中获取元数据信息来实现。之后给每个 Map 任务根据其所需数据所在节点的距离来设定优先级，距离越近，优先级越高。在任务调度时，调度器优先考虑高优先级的任务，从而尽可能的将任务分配给拥有该任务所需数据的节点。

任务窃取和任务共享是分布式计算系统中实现动态任务调度的两种基本方

案。为了减少分布式系统中在多个节点间进行任务迁移的通信开销，自适应、分层次的任务调度方案 AHS（An Adaptive And Hierarchical Task Scheduling Scheme）[12]将计算节点进行分区，实现节点间的静态负载均衡。AHS 将任务调度分为节点内的调度和节点间的调度两层，优先进行节点内的任务迁移，在节点间通过任务共享的方式将本地的计算时间和节点间任务迁移的延迟重叠，避免跨节点的任务迁移带来的通信开销。AHS 方案比现有的任务调度方案的性能提升了 11%到 21.4%。

大规模的数据集往往具有倾斜性，在 MapReduce 并行计算架构中运行时会导致一些节点运行 Reduce 函数时的负载较大，从而增加整体的计算处理时间。节点的运行成本估计是实现自适应的负载均衡算法的基础，这需要在工作节点运行 Map 函数时收集运行时信息，并且这一过程需要在全局运行，较多的信息会增加成本估计的复杂度。TopCluster[13]在常数时间内构建全局直方图用于估计分区成本，可以近似的反映出数据集中频率最高的键，并将估计误差限制在一个阈值之内。

MBR（Map-Balance-Reduce）[14]编程模型在 MapReduce 的 Map 和 Reduce 阶段添加了 Balance 阶段，在 Reduce 阶段之前对于负载较重的节点根据 K-hash 将其任务平均分为给多个 Reduce 节点，从而避免 Reduce 阶段的负载不均匀。在测试数据分布不均匀的情况下，在 Hadoop 上运行的 MBR 编程模型可以将效率提升 9.7% 到 17.6%。此外，对于数据为常规的均匀分布时，它只会带来 1.02% 的时间开销。相似的，[15]提出基于抽样的二阶段哈希分区策略，把 MapReduce 中的分区阶段划分为两个阶段。在第一阶段利用哈希算法对输入数据进行分区，并统计每个 Reduce 节点的数据处理量，如果大于某个阈值，则将该节点划分为异常分区。在第二阶段，对第一阶段的异常分区进行进一步的细分处理。

数据中心的很大一部分作业都是周期性的，具有可预测的特征。这些周期性作业会使网络呈现出繁忙时段和空闲时段交替出现的现象。因此利用可预测的网络情况更好地调度 MapReduce 中的 Shuffle 任务，避免出现多个 MapReduce 作业同时处于 Shuffle 阶段，造成系统的通信负载较大的情况。NAS（Network-Aware Shuffle Scheduling Algorithm）[16]利用数据中心可预测的周期性网络状态，在网络

空闲的时间段内安排 Shuffle 任务，加快 Shuffle 阶段执行的时间。

[17]提出了一种基于 Hadoop 日志文件分析 Hadoop 集群中机架的工作负载，从而在机架之间动态平衡工作负载的算法。通过将任务从最繁忙的机架转移到其他机架，可以减少作业的整体运行时间，从而提高集群的性能。[17]的测试表明该方案可以将最繁忙机架上作业的任务执行时间减少 50%以上。

1.2.3 关于网络拥塞的研究现状

TCP[18]的拥塞控制策略是基于对网络环境的假设，例如当发送方接收到 3 个重复的 ACK 或者等待 ACK 时间超过阈值，此时 TCP 便认为发生丢包事件，将触发超时重传或者丢失重传，并缩小发送窗口的长度。TCP 硬性地假设丢包事件或超时事件表示网络拥塞，这使得 TCP 的拥塞控制策略无法处理更复杂的网络环境，在数据中心内部拥塞的情况下表现不佳，并且会产生非常高的延迟。PCC(Performance-oriented Congestion Control)[19]策略由在线学习算法驱动，根据实时的实验结果来理解哪些操作可以优化系统的通信性能，避免了 TCP 对网络环境的硬性假设。它通过观察每个发送方的不同行为与实际性能之间的关系动态地调整每个发送方的行为，提供了相比 TCP 适应性和灵活性更好的拥塞控制策略。在快速变化的网络环境中，PCC 的性能是 CUBIC 的 14 倍，是 Illinois 的 5.6 倍。

参考机器学习中的凸优化，PCC Vivace[20]对在 PCC 的基础上对其效用函数和学习效率控制算法进行了调整，采用了严格凸的效用函数，并根据计算出来的效用值动态调整发送速率的步长，保证多个发送方都可以更快地收敛到最优发送速率，PCC Vivace 在吞吐量、丢包率、收敛速度、对网络状况变化的反应速度上实现了比 PCC 更好的性能。

Custard（CUSTomized And Robust Decision）[21]进一步扩展了 PCC，采用深度强化学习来生成策略，将观测到的指标(如丢包率，平均传输延迟等)映射为发送速率的调整选择。Custard 采用了一个三个隐藏层，每层 32 个神经元的神经网络来完成这个生成策略。可以通过设置不同的奖励函数来让网路达到需要的性能要求。

1.3 论文的主要内容与结构

1.3.1 论文的主要内容

本文的主要内容如下：

（1）实现一个基于 MapReduce 的并行计算原型系统，该系统应当可以在动态变化的网络环境中运行,并且指定网络带宽的具体分布情况；

（2）实现了由在线学习驱动的动态编码分布式计算方案，实现对动态变化的网络环境的自适应，实现系统中通信开销和计算开销的动态平衡；

（3）在实现的原型系统上对本课题的方案的性能进行测试，并且和 MapReduce 原型系统的性能进行比较和分析。

1.3.2 论文结构

本论文的主要内容如下：

第一章首先介绍了大规模并行计算系统在动态变化的网络环境中面对的挑战，阐述了本课题要解决的问题和现实意义。然后介绍了目前国内外在并行计算领域内与该问题相关联的研究工作和一些技术。最后对本文的主要研究内容作了具体说明。

第二章主要介绍了和本课题的方案相关的几项技术，主要是关于 MapReduce 并行计算架构，网络编码技术和编码分布式计算技术，以及由在线学习驱动的拥塞控制策略等。接下来将实现的方案中将应用到这几项技术。

第三章首先对 MapReduce 并行计算系统在动态网络环境中计算开销和通信开销进行了简单的调研。然后介绍了如何通过由在线学习驱动的编码分布式计算实现对网络环境的自适应。它将通过编码分布式计算技术来降低并行计算系统的通信开销，并通过执行微实验的方式对系统中的通信开销和计算开销进行评估来决定对编码分布式计算的参数的调整方向，从而实现根据动态变化的网络环境的自适应调整。

第四章介绍了本课题方案的实现过程。介绍了编程和测试所使用的并行计算模拟平台。之后介绍了 MapReduce 原型系统的设计和实现过程。最后介绍了本课题方案的代码实现过程。

第五章介绍了本课题的测试过程，对上述方案的性能进行了测试，并给出了测试结果。测试对比了 MapReduce 方案，静态的编码分布式计算方案和由在线学习驱动的动态编码分布式计算方案在指定网络带宽分布情况的网络环境中的计算性能，以及对动态网络环境的适应情况。

第六章总结了本课题所做的相关工作和不足，以及接下来要做的工作和未来的展望。

2 相关技术基础

本章节将介绍和本课题的方案相关的一些技术基础，主要包括 MapReduce 并行计算框架，网络编码技术和编码分布式计算技术，以及面向性能的网络拥塞控制策略。本课题的方案将基于编码分布式计算和由在线学习驱动的拥塞控制策略两项技术，在 MapReduce 并行计算框架的基础上通过对编码分布式计算方案的动态调控实现对动态网络环境的自适应。

2.1 MapReduce 并行计算框架

根据 IDC Digital Universe 的研究，从 2009 年到 2023 年，数字数据将增长 44 倍，达到每年 35ZB[22]。为了应对数据生成量级的新变化，提升计算系统的计算效率，有多种并行计算框架被提出，典型的并行计算框架有 MapReduce、Spark[23][24]等。其中，MapReduce 由谷歌提出用来处理其数据中心网络索引构建、日志处理、数据挖掘等问题。MapReduce 受函数式编程的启发，将计算分布在许多节点上，这些节点可以通过分布式文件系统访问所有数据。用户调用 MapReduce 通常需要提供一个 Map 函数和一个 Reduce 函数，每个节点各自运行 Map 函数和 Reduce 函数，共同完成计算任务。

MapReduce 的计算过程主要可分为三个阶段：Map 阶段、Shuffle 阶段和 Reduce 阶段。在一个节点上进行的计算过程的伪代码如算法 2-1 所示。在 Map

算法 2-1：MapReduce 算法

```
输入：input_keys // 键的数组
      input_values // 值的数组
输出：result // 输出的键值对
1.function Map(key,value)
2.   // key:输入数据的键
3.   // value:输入数据的值
4.   for each word w in value:
5.      EmitIntermediate(w, 1)
6.
```

```
7.function Reduce(key, values):
8.    // key: 中间键
9.    // values: 中间键对应的值的集合
10.   count←0
11.   for each v in values:
12.   count←count+v
13.   Emit(key, count)
14.
15.function MapReduce(input_keys,input_values):
16.   result←[]
17.   MapResults[]
18.   for (key, value) in (input_keys,intput_values):
19.       MapResult←Map(key, value)
20.       MapResults.AddMapResult(MapResult)
21.   SplitData←GroupByKeys(MapResults.GetMapResults())
22.
23.   for (key, values) in SplitData:
24.       ReduceResult←Reduce(key, values)
25.       result.AddReduceResult(ReduceResult)
26.       输出 result
```

阶段，每个节点会根据 master 节点的调度，从分布式文件系统中读取本节点需要处理的数据，并根据这些数据执行 Map 函数，将生成一系列键值对。Map 阶段结束后每个节点的本地都保存了一些键值对作为中间数据。在 Shuffle 阶段，每个节点将其本地的中间数据根据一定的规则进行分区，例如根据对中间数据的键进行哈希运算决定该键值对将被传输到的节点。之后将同一分区内的中间数据发送到同一个的节点用于执行接下来的 Reduce 操作。在 Reduce 阶段，每个节点根据传输到自己本地的中间数据执行 Reduce 函数，生成最终的输出结果。MapReduce 系统整体的计算和调度过程如图 2-1 所示，其中 master 阶段主要负责调度 worker 的工作，worker 节点主要负责执行计算操作。在 MapReduce 计算过程中，Map 阶段负责对输入文件进行处理，因此计算开销主要产生在 Map 阶段。在 Shuffle 阶段，需要将中间结果在节点间进行传输，因此通信开销主要集中在 Shuffle 阶段。

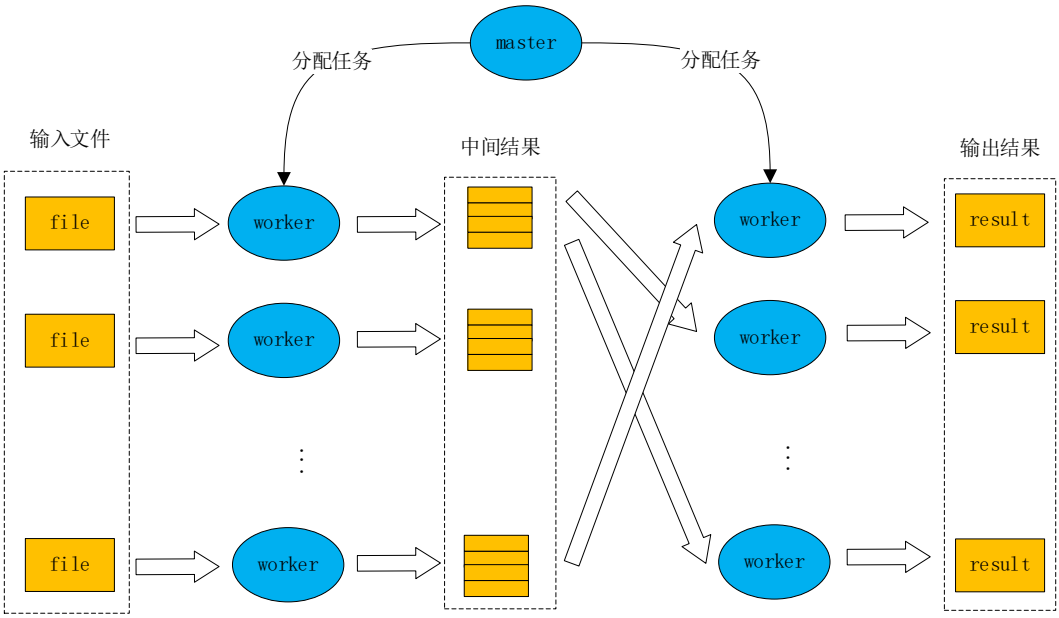

图 2-1 MapReduce 计算过程

相比其他的并行计算框架，MapReduce 主要有以下几个优势：

（1）可扩展性强。MapReduce 框架用于处理大规模数据，可以有效地扩展到数千个节点的集群上；

（2）编程难度低。用户无需关注节点间运行时同步和调度相关的具体细节，只需要提供自定义的 Map 和 Reduce 函数便可以完成计算任务；

（3）具有较好的容错性。MapReduce 提供了容错机制，可以在节点故障时重新执行未完成的任务，以确保整个作业的成功执行；

（4）成本低廉。MapReduce 系统将多个廉价的机器组成集群，在保持较低的成本的同时可以提供较高的计算性能。

2.2 网络编码

网络编码[25]是一种允许网络传输过程中的中间节点将多个数据包进行编码为单个数据包之后再转发的网络传输方式。传统的网络传输中，中间节点只负责对数据包进行存储和转发，而基于网络编码技术的网络传输将数据包进行处理后进行多播，从而减少数据包的传输量。一个简单的利用网络编码的传输过程如图 2-2 所示。在该示例中，节点 s 需要把数据包 x 和数据包 y 发送到节点 d_1 和节点 d_2，需要经过中间节点 a、b、c、d。在网络带宽受限的情况下，譬如每个节点一

次只能转发一个数据包，那么节点 c 向节点 d 转发数据包 x 和 y 则需要两次路由转发。由于节点 d_1 和 d_2 分别可以通过节点 a 和节点 b 接收到数据包 x 和数据包 y，那么节点 c 在转发数据包 x 和 y 时便可以对其进行异或编码处理之后再转发给节点 d，节点 d 将编码后的数据包转发给节点 d_1 和 d_2 后，两个节点再根据已经接收到的一个数据包进行异或解码得到另一个数据包，从而减少整个过程中的数据包传输数量。[26]证明了网络编码可以实现从源节点集合到目的节点集合进行组播通信的数据传输速率的上界。

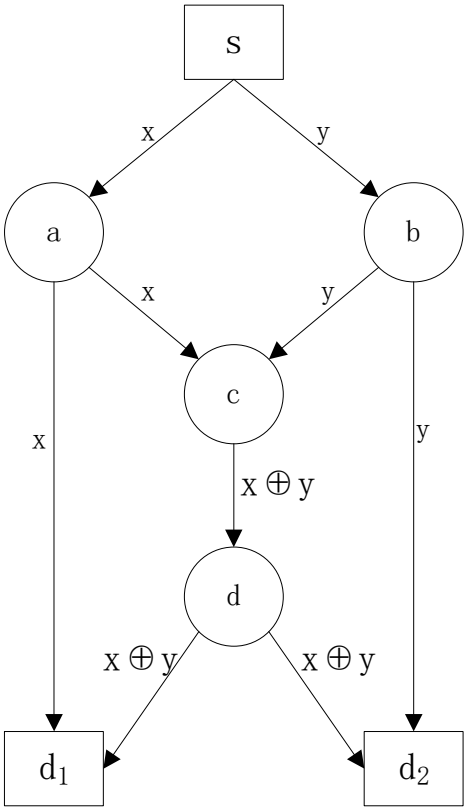

图 2-2 基于网络编码的网络传输示意图

线性网络编码是网络编码的一种实现形式。其编码方式通过矩阵乘法实现。假设源节点s需要向目的节点集合发送N个数据包。将第i个数据包定义为$x_i = (x_{i,1}\ x_{i,2}\ \dots x_{i,L})(i = 1,2,\dots,N)$，其中$L$表示单个数据包的长度，数据包中的每个符号$x_{i,j}$都属于有限域$GF(q)$($q$为有限域的大小)。对于网络中的中间节点$v$，将$p_i^{[v]} = \left(p_{i,1}^{[v]} p_{i,2}^{[v]} \dots p_{i,L}^{[v]}\right)\left(i = 1,2,\dots,M^{[v]}\right)$定义为该节点接收到的第$i$个数据包，$M^{[v]}$表示该节点接收到的数据包的数量。节点 v 在向节点 u 进行转发前会对自己接收到的

数据包进行编码处理，编码方式为：

$$P_{out}^{v,u} = \sum_{i=1}^{M^{[v]}} c_i^{(v,u)} p_i^{[v]}$$

其中$c_i^{(v,u)}$属于有限域$GF(q)$，$c^{(v,u)} = \left(c_1^{(v,u)} c_2^{(v,u)} \dots c_N^{(v,u)}\right)$是数据包$p_{out}^{(v,u)}$的编码向量。当目的节点集合中的节点 r 接收到经过编码的数据包之后，经过求解线性方程便可解码获得所有的源数据包。将源数据包简写为$x_i(i = 1, \dots, N)$，编码数据包简写为$y_i\left(i = 1, \dots, M^{[r]}\right)$，编码数据包和源数据包的转换关系为：

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{M^{[r]}} \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,N} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ c_{M^{[r]},1} & c_{M^{[r]},2} & \cdots & c_{M^{[r]},N} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

根据此转换关系，经过矩阵的逆运算便可以根据编码数据包解码获得源数据包。

2.3 编码分布式计算

根据 2.1 节中的描述，在 MapReduce 计算过程中，计算开销主要产生于 Map 阶段，Shuffle 阶段会产生较大的通信开销。对于一些计算负载相对较小，通信负载相对较大的计算任务，Shuffle 阶段会持续较长时间从而降低整体的计算性能。

编码分布式计算[7]通过在 MapReduce 中的 Map 阶段引入冗余的计算，利用网络编码来减少 Shuffle 阶段的的通信负载，从而加快整体计算。通过把 Map 的计算负载增加 r 倍，即根据编码方案将同一个输入文件同时映射到 r 个节点上执行 Map 函数，将会生成冗余的中间结果，从而在 Shuffle 阶段创造出编码多播的机会将 Shuffle 阶段的通信开销降低 r 倍。经过编码的 MapReduce 算法的伪代码如算法 2-2 所示。一个没有经过编码的和经过编码 MapReduce 并行计算过程的

算法 2-2：经过编码的 MapReduce 算法

```
输入：input_files // 输入文件的集合
输出：result      // 输出结果
1.function CodedMapReduce(input_files):
2.   // 在每个计算节点上运行该函数
3.   GenerateMultiCastGroup() // 生成编码多播集合
```

```
for file in input_files:
    ImmediateData←ExecuteMap(file) // 执行 Map 函数
    ImmediateDatas.Add(ImmediateData)
ExecuteEncode(ImmediateDatas) // 根据编码多播集合将属于同一个多播集合内的中间结果进行异或编码
ReceiveDatas = []
将编码后的中间结果多播到相应节点
ReceiveDatas←接收来自其他节点的中间结果
ReduceDatas←ExecuteDecode(ImmediateDatas,ReceiveDatas) // 根据本地的中间结果对接收到的中间结果进行异或解码
result←ExecuteReduce(ReduceDatas) // 执行 Reduce 操作
输出 result
```

示例如图 2-3 和图 2-4 所示。在 K=3 个节点上，输入 N=6 个文件，执行 Q=3 个函数。对于未经过编码的 MapReduce 计算过程，Shuffle 阶段需要传输的中间结果为 12 个。对于经过编码的 MapReduce 计算过程来说，在 Map 阶段把每个输入文件映射到 r=2 个节点上生成冗余的中间结果，从而在 Shuffle 阶段通过将中

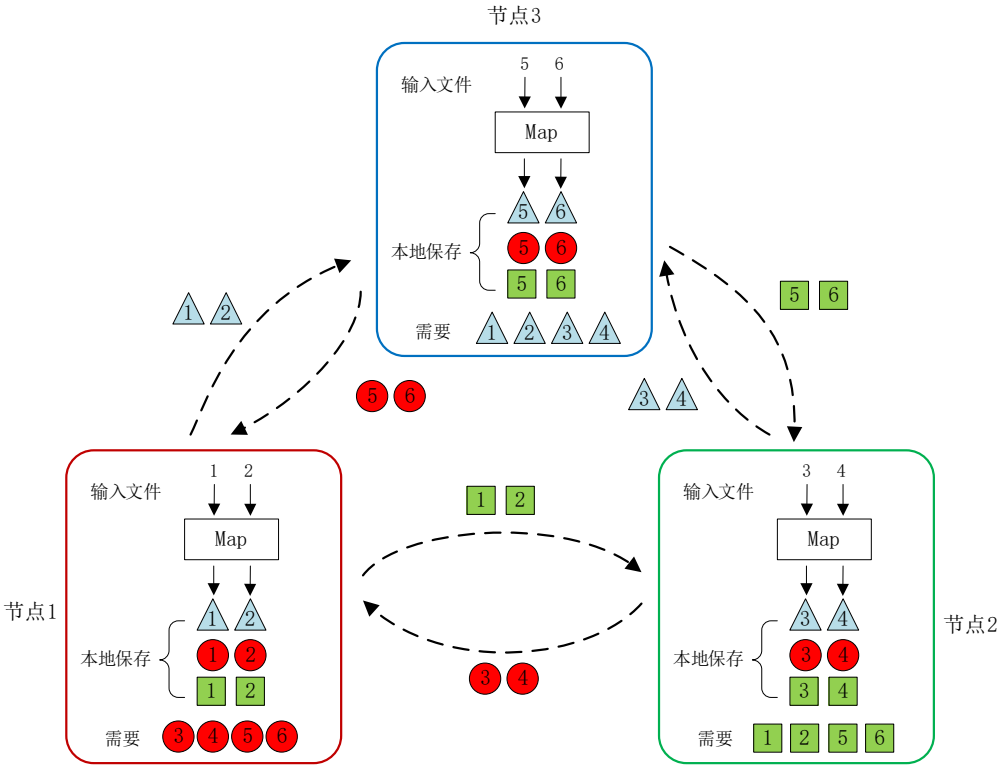

图 2-3 普通的 MapReduce 计算过程

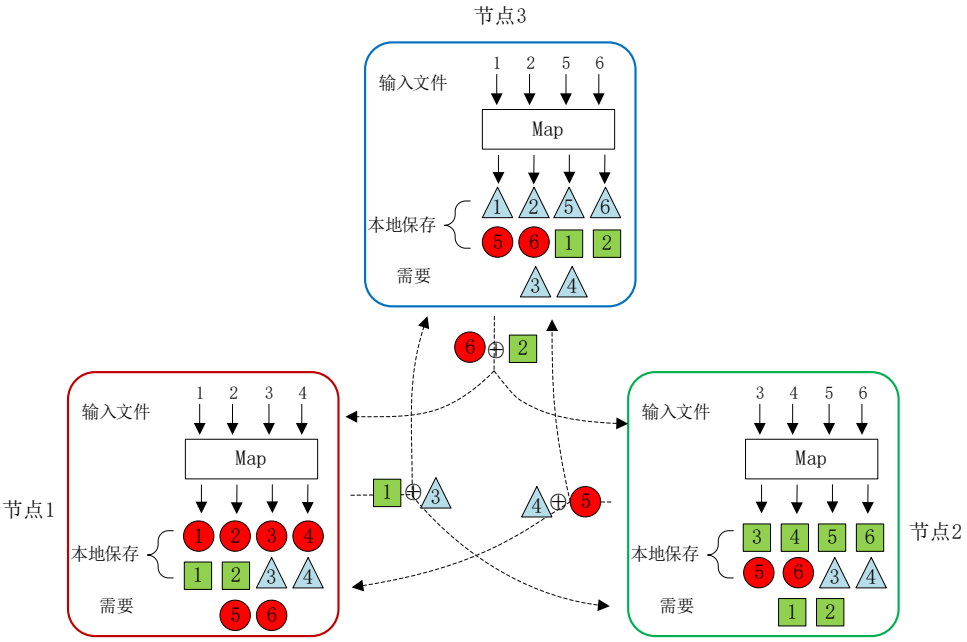

图 2-4 经过网络编码的 MapReduce 计算过程

间结果进行异或编码同时多播给多个节点。如果不经过异或编码，Shuffle 阶段需要传输的中间结果为 6 个，经过编码后需要传输的中间结果为 3 个。在 Map 阶段将计算负载增加 r=2 倍后，在 Shuffle 阶段便可以通过编码多播的方式将通信开销降低 r=2 倍。其中，Shuffle 阶段的编码多播是通过 OpenMPI[27][28]提供的函数接口的 MPI_Bcast 实现的。MPI_Bcast 是用于并行计算中在多个进程之间广播消息的函数，可以把处于同一个进程组内的一个进程的本地缓冲区的数据作为消息发送出去，其余进程接受消息。

2.4 面向性能的拥塞控制策略

PCC（Performance-oriented Congestion Control）[19]是一种由在线学习驱动的拥塞控制策略，其中每个发送方持续观察观察其操作和实际性能之间的联系，从而持续采用为系统带来高性能的操作。通过采用合适的效用函数以及步长控制算法可以实现较好的性能和收敛性。

假设当前节点以某个速率发送数据出现了丢包事件，那么它在接下类应该如何调整呢？导致丢包事件的发生的原因多种多样，可能是该节点的发送速率太大了，此时该节点应该降低自己的发送速率。也可能是存在其他节点在以较大的速率发送数据，此时该节点的发送速率应当保持不变而让其他节点进行调整。因此，

在发生丢包事件后直接降低本节点的发送速率是不太恰当的。在 PCC 的运行过程中，将以速率 r 发送一小段时间的数据，观察实际运行过程的性能（如传输延迟，丢包率等），将这些性能指标聚合到一个效用函数中，生成效用值 u，借此观察实际的调整动作是否会改善实际的运行性能。不同的效用函数可以使系统实现不一样的目标。比如，如果系统更在意丢包率或者传输延迟，那么可以将效用函数设为反映丢包率或传输延迟的函数，或者二者的组合函数。通过运行单个微实验，PCC 得到了速率 r 发送数据的效用值 u。在运行过程中，PCC 将运行多个微实验，以分别两种不同的速率发送数据，观察实际的性能并将其聚合为效用值进行比较，将发送速率朝着将产生更高性能效用的方向移动。不同于 TCP 对网络环境的预设，PCC 由在线学习算法驱动，通过跟踪实际的运行时性能来寻找最佳的发送速率。PCC 的运行过程图如图 2-5 所示。

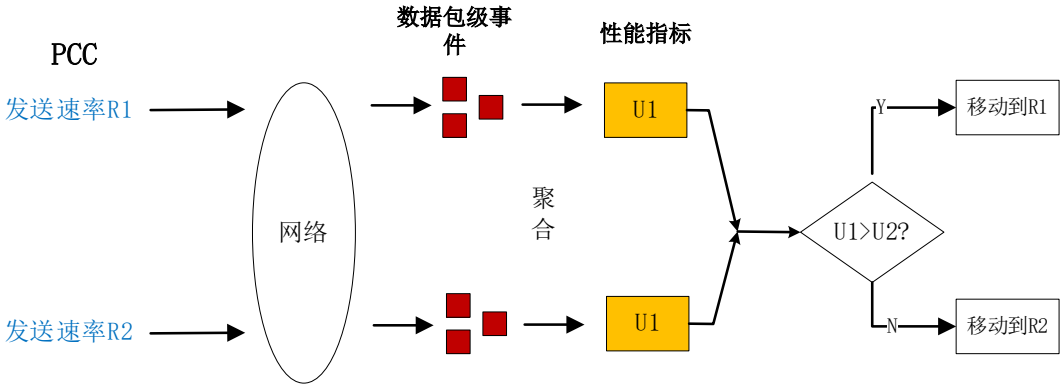

图 2-5 PCC 运行过程图

为了避免网络中的偶然因素的干扰，一组微实验将执行 4 次尝试，其中分为两组，一组采用较大的速率为(1+ε)r，另一组采用较小的速率为(1-ε)r。在发送出 4 个连续的微实验后，PCC 将发送速率恢复为初始值，并继续等待 ACK 信息。当 ACK 信息收集完毕之后，PCC 将根据相关信息计算出 4 个效用值，记作 U^+_1,U^+_2,U^-_1,U^-_2，分别表示采用较大的发送速率的两个和较小的发送速率的两个效用值。之后的调整过程如下所示：

（1）如果 $U^+_1 > U^-_1$ 的同时 $U^+_2 > U^-_2$,此时便把发送速率调整为较大的那个；

（2）如果 $U^+_1 < U^-_1$ 的同时 $U^+_2 < U^-_2$ 则把发送速率调整为较小的那个;

（3）如果两组比较的结果不一致，则修改发送速率的调整步长 ε 为 $\varepsilon+\varepsilon_{min}$，之后重复上述的过程。这一步是为了防止网络环境中的噪声干扰。

2.5 本章小结

本章对本课题接下来将实现的方案中需要用到的相关技术做了介绍。首先介绍了基于 MapReduce 的并行计算框架，描述了 MapReduce 并行计算的一般性过程。之后介绍了网络编码技术和编码分布式计算技术，本课题中将应用这些技术来降低并行计算系统中的通信开销。最后介绍了面向性能的网络拥塞控制策略，本课题将参考该策略对编码分布式计算方案进行动态调节，实现对网络环境的自适应。

3 动态自适应并行计算方案的设计

本章首先对基于三种分布的动态网络环境下的 MapReduce 系统的性能进行简单的调研，阐述了实现对动态网络环境的自适应的必要性。之后将详细介绍如何在 MapReduce 并行计算系统上通过由在线学习驱动的动态编码分布式计算方案，实现对网络环境的自适应。

3.1 对动态网络环境的自适应的必要性

在 MapReduce 并行计算框架中，计算开销主要产生在 Map 阶段，通信开销主要产生在 Shuffle 阶段。在故障频发的网络环境中，当网络的拥塞情况比较严重时，会频繁发生丢包事件，同时数据包的传输延迟会增加，从而使得 Shuffle 阶段的执行时间增加，Reduce 阶段的执行又依赖 Shuffle 阶段从其他节点接收到的中间结果，因此整个计算任务的执行时间将增加。

首先为了观察动态网络环境对基于 MapReduce 并行计算系统执行任务的性能的影响，在分别基于离散型均匀分布，指数分布，Pareto[29]分布三种网络带宽分布情况下，对 MapReduce 系统的执行情况进行了简要的测试和分析。主要收集了 Map 阶段和 Shuffle 阶段的执行时间，将 Map 阶段的执行时间作为计算开销，将 Shuffle 阶段的执行时间作为通信开销，并统计 Shuffle 阶段和 Map 阶段的比值。计算开销和通信开销应当尽可能的平衡，因此希望该比值尽可能的接近 1。测试时在 MapReduce 系统执上行了 128 个计算任务，该系统是一个在 SimGrid 模拟平台上实现的原型系统（见 4.1 节）。在基于离散型均匀分布、指数分布和 Pareto 分布的网络环境下其通信开销和计算开销的比值情况分别如图 3-1、图 3-2 所示和图 3-3 所示。根据该比值的变化情况可见，在三种网络分布下，随着网络环境的变化，通信开销相对计算开销会产生比较严重的波动，在链路拥塞严重的情况下该波动情况将给 Shuffle 阶段的运行造成很大的延迟，进而降低整个计算任务的执行效率。因此，在本课题中实现一个让 MapReduce 系统具有对动态网络环境的自适应性的方案是有必要的。

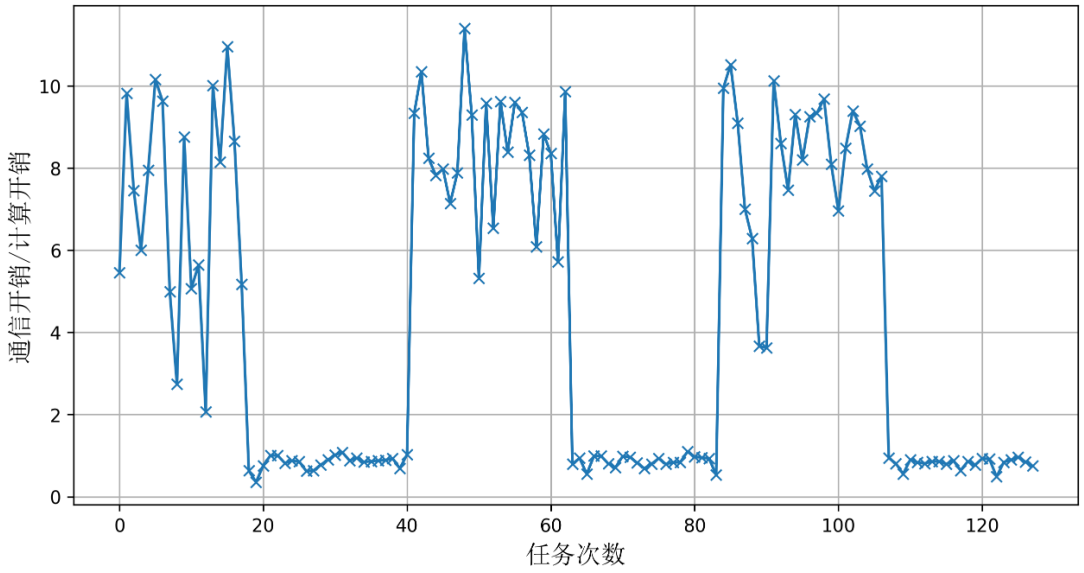

图 3-1 基于均匀分布的网络环境下 MapReduce 的通信开销和计算开销的比值变化情况

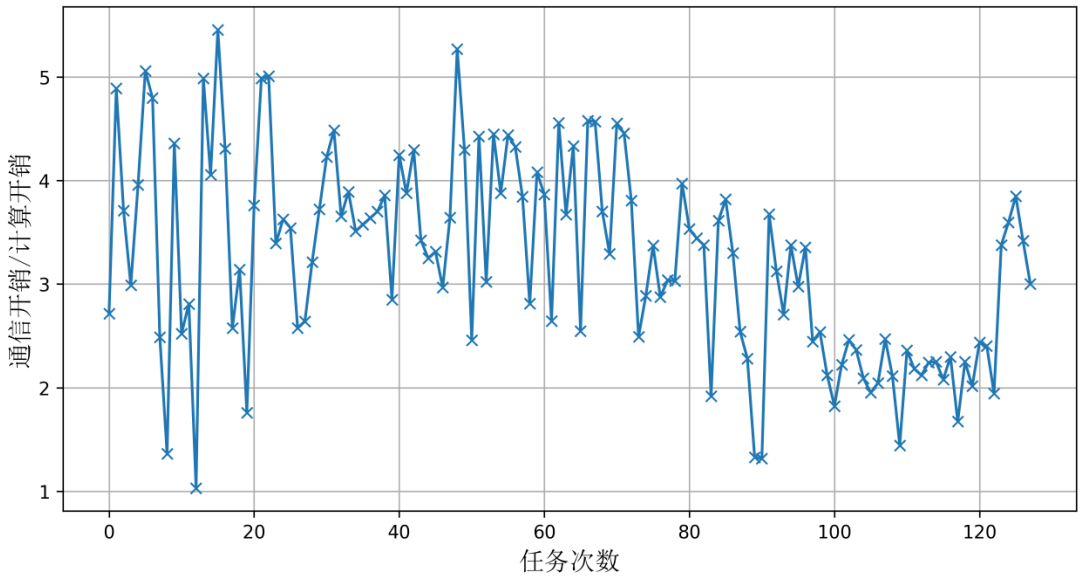

图 3-2 基于指数分布的网络环境下 MapReduce 的通信开销和计算开销的比值变化情况

3.2 需求分析

根据 3.1 节中的调研结果，本课题的方案需要满足以下要求：

（1）在网络资源相对紧张的情况下实时地对环境做出反馈，通过一些技术来降低系统的通信开销，从而避免网络环境对计算性能造成影响，同时缓解系统中

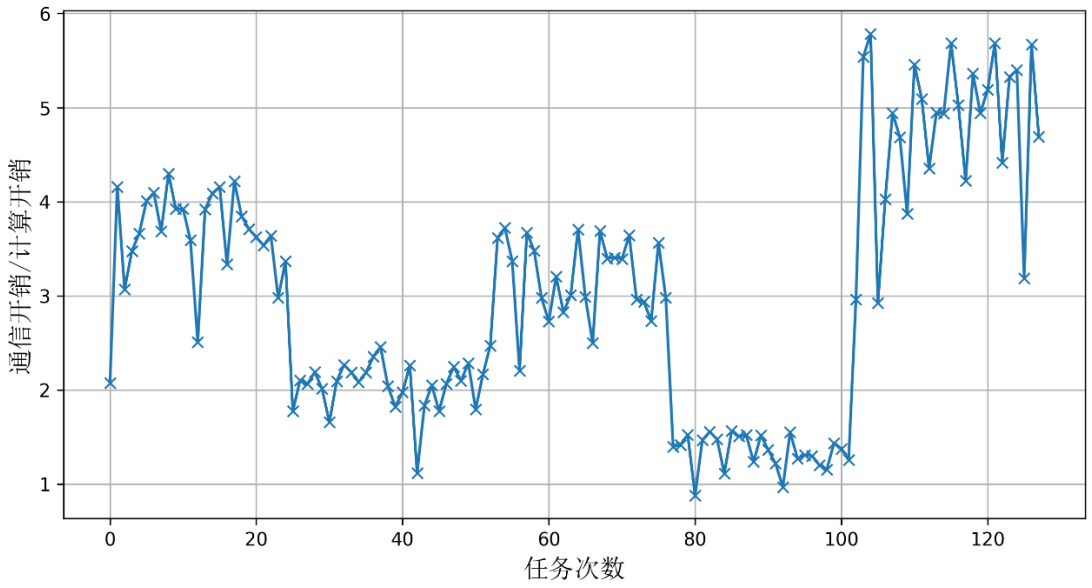

图 3-3 基于 Pareto 分布的网络环境下 MapReduce 的通信开销和计算开销的比值变换情况

的网络拥塞的情况；

（2）该方案应当面向动态变化的网络环境，可以根据网络带宽的变化情况实时地调整计算开销和通信开销之间的平衡；

（3）该方案应当根据系统的计算资源和通信资源的实际情况、对性能的具体要求，合理地对任务执行的性能进行评估；

（4）该方案应当可以避免环境中偶然因素的干扰。

3.3 由在线学习驱动的编码分布式计算方案

[7]阐述了在 MapReduce 中计算开销和通信开销之间的权衡，提出通过增加编码分布式计算方案中的负载因子 r 可以增加 Map 阶段的计算开销，可以在 Shuffle 阶段创造编码多播机会将通信开销也降低 r 倍，从而增加 Shuffle 阶段的执行效率。反之降低参数 r 则会增加 Shuffle 阶段的通信开销。对于计算密集型任务，应当尽可能减少 Map 阶段的计算开销，而对于通信负载相对较大的任务，应当适当增加 Map 阶段的计算开销，从而减少 Shuffle 阶段的通信开销，提升整个计算任务的性能。因此可以考虑在 MapReduce 中应用编码分布式计算方案来调节计算过程的通信开销和计算开销。

而对于动态变化的网络环境，如何适当的设置编码分布式计算方案的参数则不再单取决于计算任务本身的计算复杂度和通信开销，也需要考虑到运行时网络的拥塞情况。静态的应用编码分布式计算方案在适应动态变化的网络环境方面可能无法实现较理想的性能。由于节点故障、链路拥塞等各种情况都会导致实时的网络带宽呈现出频繁变化的趋势，因此需要对编码分布式计算方案进行动态的调整，实现对计算开销和通信开销之间的权衡的动态调节。导致网络环境变化的原因众多，可能是链路拥塞导致丢包率上升，也可能是某个节点出现突发性的故障使得 Shuffle 阶段出现滞后节点导致传输时间延长，硬性的对网络环境变化的原因进行假设可能导致动态适应性较差。因此本课题考虑应用[19]提出的由在线学习驱动的拥塞控制策略到编码分布式计算方案中。

本课题方案的伪代码如算法 3-1 所示。该方案将动态的运行编码分布式计算方案，并在运行过程进行实时的调整。假设在系统运行时正处于状态 r，即正在运行以负载因子为 r 的编码分布式计算。此时将在状态 r 的基础上运行微实验，从任务列表中接收两个任务，分别以状态 r^+和状态 r^-运行这两个任务。状态 r^+和状态 r^-表示以将负载因子 r 分别增大和减小一定的幅度运行编码分布式计算方案，状态 r+和状态 r-的负载因子 r 分别根据 3-1 和 3-2 所示的公式求得，其中ϵ表示对

$$r^+ = r(1 + \epsilon) \tag{3-1}$$

$$r^- = r(1 - \epsilon) \tag{3-2}$$

编码分布式计算方案的调整步长，目前本课题实现的系统在运行时采用的ϵ为 0.5。之后节点负载网络感知模块将监视这两个任务的执行情况并收集反馈信息。收集的具体信息为 Map 阶段的执行时间和 Shuffle 阶段的执行时间，分别用于反映了节点的计算负载和传输中间结果的网络负载。本课题的方案的流程图如图 3-4 所示，当两个任务的反馈信息都收集完毕，便将其根据效用函数聚合成效用值。具体的效用函数则可以根据并行计算系统的实际情况进行设置。例如如果系统本身的网络带宽资源相对紧张的情况下，便可以将效用函数函数中通信开销的权重设置的较大些，反之如果系统的计算资源相对紧张时便可以将计算开销的权重设置的大些。目前本课题的实现系统采用的聚合公式如 3-3 所示，α暂时定为 0.5，旨在实现计算开销和通信开销的平衡。

$$U = \alpha NetworkLoad + (1-\alpha)ComputationLoad \quad (3\text{-}3)$$

算法 3-1：由在线学习驱动的编码分布式算法

输入：job_queue // 任务队列，可从中读取任务

r // 当前状态下的负载因子 r

输出：result // 任务的输出结果

```
1.function OnlineLearningMapReduce(job_queue,r):
2.   // 该函数在 master 节点上运行
3.   Master←GetFreeMaster() // 获取一个空闲的 Master
4.   r+←r*(1+ε)
5.   r-←r*(1-ε)
6.   jobs←job_queue.Pop() // 从任务队列中获取任务用于微实验 一次获取 4 个
7.   UtilityInfos←[]
8.   // 该循环的几个任务的执行过程可并行执行
9.   for job in jobs:
10.     Master←GetFreeMaster() // 获取一个空闲的 Master
11.     job.SetR(r+,r-) // 设置该任务的负载因子为 r+或 r-
12.     Master.DoJob(job)
13.     UtilityInfos.Collect() // 收集系统中的性能评估数据
14.  根据 UtilityInfos 决定负载因子的调整方向
15.  调整系统的负载因子 r
16.  输出任务的执行结果
```

在获得了两组微实验的效用值之后进行比较，系统便可以获知在当前网络环境下哪种调节方向更有利于计算任务的执行，之后系统便根据此方向调节编码分布式计算的负载因子 r。

根据 2.4 节中的描述，为了避免执行微实验时偶然因素的干扰，一般一次调整操作将执行两组微实验，当两组微实验的效用值比较结果一致时才进行状态的迁移。当两组微实验结果不一致时，例如第一组微实验的结果为状态 r^+的性能优于状态 r^-，第二组微实验的结果相反，此时便不进行状态的迁移，在状态 r 的基础上继续重复该过程。

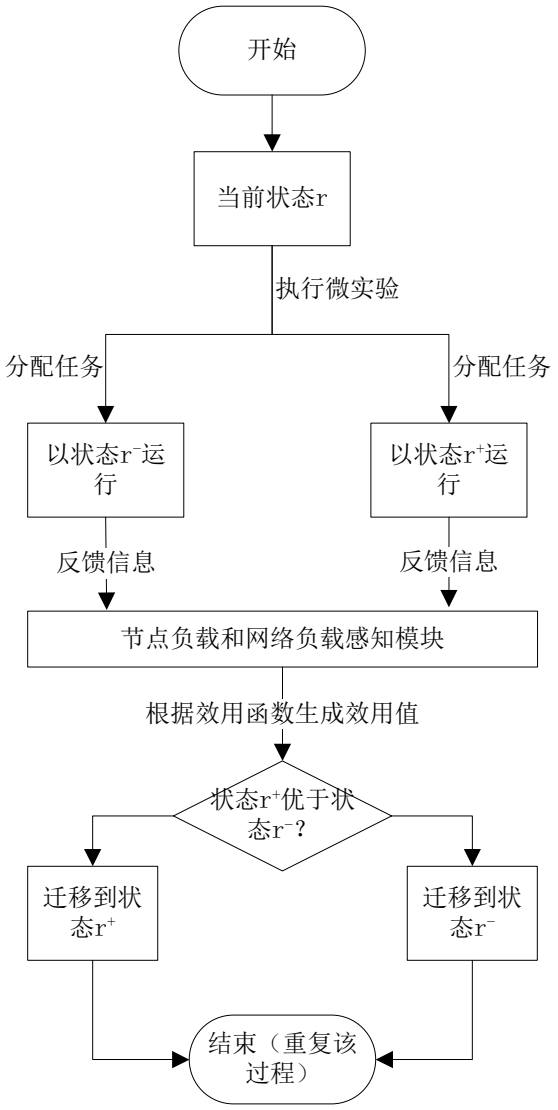

图 3-4 动态网络环境自适应方案流程图

3.4 成本估计

本课题的方案的主要工作集中在实现由在线学习驱动的动态调整模块，以及基于 MapReduce 的原型系统和动态网络配置模块的完成。另外需要对编写一些和模拟平台相关的配置文件和节点的配置信息。初步估计本课题的代码行数在 2000 行左右。

根据 Walston-Felix 软件成本估计模型[30]，项目的成本可以用如 3-4 和 3-5 所示的公式来估算。其中 KLOC（Kilo Source Line of Code）为代码行数（以千行为单位），E 为工作量，以 PM 估计。D 为项目持续时间，以月估计。

$$E = 5.2 \times (KLOC)^{0.91} \quad (3\text{-}4)$$

$$D = 4.1 \times (KLOC)^{0.36} \quad (3\text{-}5)$$

根据此成本估计模型，可以计算得到本课题的工作成本和时间成本，分别为：

$$E \approx 9.77$$

$$D \approx 5.26$$

3.5 本章小结

本章首先对在动态网络环境中运行的 MapReduce 并行计算系统的性能进行了一些调研，在三种网络带宽分布情况下统计了计算任务的计算开销和通信开销的具体情况，展示了实现并行计算系统对动态变化的网络环境的自适应的必要性。之后明确了本课题的方案的具体需求和目标。然后本章详细介绍了如何通过由在线学习驱动的编码分布式计算方案，在 MapReduce 并行计算系统的基础上实现对动态变化的网络环境的自适应。

4 动态自适应并行计算方案的实现

本章将详细介绍 3.3 节中的动态自适应并行计算方案的具体实现过程。首先阐述了一个基于 MapReduce 的并行计算原型系统的实现过程，之后为该系统添加了动态网络调节模块，使其可以在指定的网络分布环境下运行。然后在这个系统的基础上阐述了 3.2 节中提出的方案的实现过程。本课题的实现代码在该代码仓库[1]中可见。

4.1 原型系统的实现

4.1.1 整体框架

该原型系统的整体框架如图 4-1 所示。主要包括 MasterManager，WorkerManager 和网络环境配置模块。其中 MasterManager 包含着一组 Master，

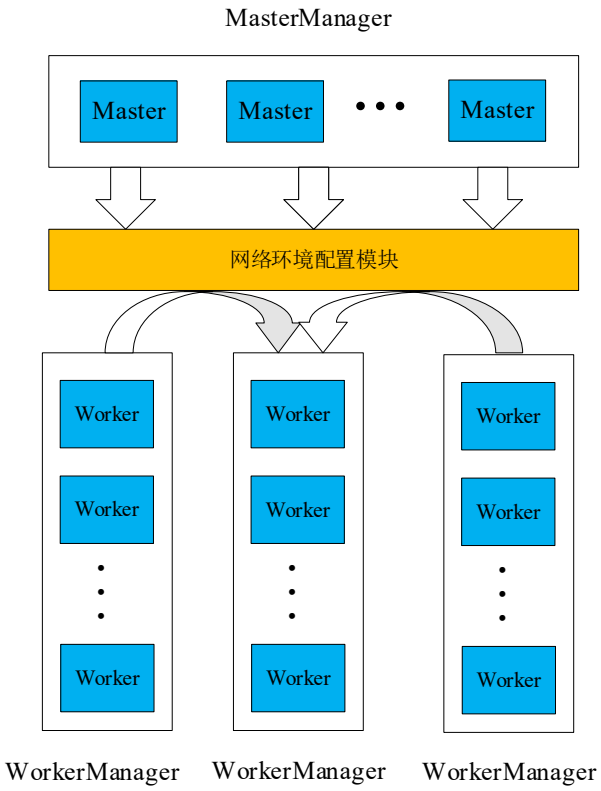

图 4-1 原型系统整体框架图

[1] https://github.com/limingqihust/bishe.git

负责调度和管理这些 Master 的运行。Master 则负责执行一个具体的计算任务的调度工作，Master 和 Master 之间并行地执行各自的计算任务，互不相关。类似的，WorkerManager 包含一组 Worker，负责调度和管理这些 Worker。网络环境配置模块负责根据配置文件中指定的网络分布情况模拟动态的网络环境，节点之间进行通信时的带宽将向该模块查询获得。

该系统在 SimGrid[31][32]平台上编程实现。SimGrid 是一个用于开发分布式应用的模拟器框架，提供了一系列用于模拟分布式程序的工具，比如网络资源、存储资源等，可被用于评估集群、网格、P2P 算法等。该系统在 SimGrid 上利用其提供的 Actor 模型来模拟多个节点的并行计算过程，一个 Actor 可以看作分布式应用程序中的一个进程。这些 Actor 显式地使用 SimGrid 提供的 S4U 接口来执行它们的计算、通信、磁盘读写等行为，SimGrid 将预测每个行为所需的时间，并相应地调度这些 Actor，等待这些活动的完成。

在系统启动时，需要在配置文件中指定节点配置信息和网络平台配置信息。节点配置信息保存在 my_deploy.xml 中，它提供的配置信息的一部分如下所示：

```
<actor host="node0" function="my_master">
    <argument value="5" />                <!-- master num of master node -->
    <argument value="6" />                <!-- worker host num -->
</actor>
<actor host="node1" function="my_worker">
    <argument value="node0" />            <!-- host name of master -->
    <argument value="5" />                <!-- num of workers in a worker node -->
    <argument value="1" />                <!-- worker host id -->
</actor>
```

其中一个 actor 描述了一个节点的相关配置信息，它指定了节点的名字，节点启动时将运行的 main 函数的函数名以及一些必要的命令行参数。比如上述的代码制定了 node0 和 node1 两个节点的 main 函数分别为 my_master 和 my_worker 和它们启动时的命令行参数。另外，它也可以指定一些其他配置信息，比如节点的 cpu 数量等。网络平台配置信息保存在 small_platform.xml 中，它提供的配置信息的一部分如下所示。它用于告知模拟平台存在哪些链路，这些链路的最大带宽值

```
<zone id="zone0" routing="Full">
```

```
    <host id="node0" speed="98.095Mf"/>
    <host id="node1" speed="76.296Mf"/>
    <link id="9" bandwidth="70000Bps" latency="1.461517ms"/>
    <link id="loopback" bandwidth="498MBps" latency="15us" sharing_policy="FATPI
PE"/>
    <route src="node0" dst="node0">
      <link_ctn id="loopback"/>
    </route>
    <route src="node1" dst="node1">
      <link_ctn id="loopback"/>
    </route>
    <route src="node0" dst="node1">
      <link_ctn id="9"/>
    </route>
</zone>
```

和延迟值，节点的最大带宽值，节点之间的路由方式，需要经过哪些链路等信息。比如上述代码则指定了 node0 和 node1 两个节点之间进行路由时需要经过哪些链路和这些链路的最大带宽值。如果程序运行时在一个节点向另一个节点发送消息时 SimGrid 无法在网络平台配置信息中寻找到这里一条链路，程序就会抛出异常。这里的带宽值都是指的链路的最大带宽值而并不是一次通信所采用的带宽值。

4.1.2 MapReduce 的实现

本课题方案的实现基于一个 MapReduce 并行计算原型系统。其中一个节点为 master 节点，在该节点上将运行 MasterManager；其余节点为 worker 节点，每个 worker 节点上各自运行 WorkerManager。每个节点对应 SimGrid 中的一个 actor。MasterManager 和 WorkerManager 具体包含的成员如下所示。

```
class MasterManager {
public:
    // receive a job, alloc a master to execute this job
    void Run(JobText& job_text);
private:
    int FindFreeMaster();
    simgrid::s4u::Mailbox* mailbox_;
```

```
    std::vector<simgrid::s4u::Mailbox*> worker_host_mailboxs_;
    std::vector<std::shared_ptr<Master>> masters_;
};

class WorkerManager {
public:
    void Run();
private:
    int FindFreeWorker();
    int worker_num_;
    int id_;
    std::vector<std::shared_ptr<Worker>> workers_;
    simgrid::s4u::Mailbox* mailbox_;                    // receive message from master manager
    simgrid::s4u::Mailbox* master_host_mailbox_;// send message to master manager
};
```

在这些类成员中，Worker 中的 id_用于唯一标识一个 worker 节点。Master 指针数组和 Worker 指针数组分别指向它们负责管理和调度的这些 Master 和 Worker。simgrid::s4u::Mailbox 型指针将用于实现节点和节点之间的通信，以及每个节点下的 Master 和 Worker 之间的通信。通信的信息主要包括任务请求信息，Shuffle 的阶段要传输的数据，节点间用于同步的信息等。该变量的相关接口由 SimGrid 模拟平台提供，基于共享内存实现。该接口提供了 Put 和 Get 等方法，消息以字符串的方式传递，当接收消息时未读取到数据时进程便会阻塞。在进行通信时可以指定此次通信拟采用的网络带宽，该带宽值最大不能超过 4.1.1 节网络平台配置文件中配置的链路的带宽最大值。系统运行时所需要的动态网络环境主要通过 simgrid::s4u::Mailbox 的这些特性实现。

系统需要执行的的计算任务组织为任务队列，供 MasterManager 读取并运行。该原型系统运行一个计算任务的过程如图 4-2 所示。当 MasterManager 从任务列表中接受到一个计算任务后，便从其管理的 Master 中寻找空闲的 Master，同时向其他 WorkerManager 发送请求以寻找空闲的 Worker。如果所有的 Master 和 Worker 都处于工作状态，则等待直至一组 Master 和 Worker 为空闲状态。Master 将会通知每个 Worker 负责的输入文件和计算任务类型，并且负责调度调度各个

Worker 完成 Map 阶段和 Reduce 阶段的计算，以及 Shuffle 阶段的数据传输工作，其中计算任务表示为 JobText 类型的变量，指定了该计算任务需要执行的函数和所有的输入文件。该变量包含的成员如下所示。其中 type 表示该任务需要执行

```
struct JobText {
    JobType type;
    int input_file_num;
    std::string input_file_prefix;
};
```

的函数名，input_file_num 和 input_file_prefix 分别表示输入文件的数量以及输入文件的前缀名。

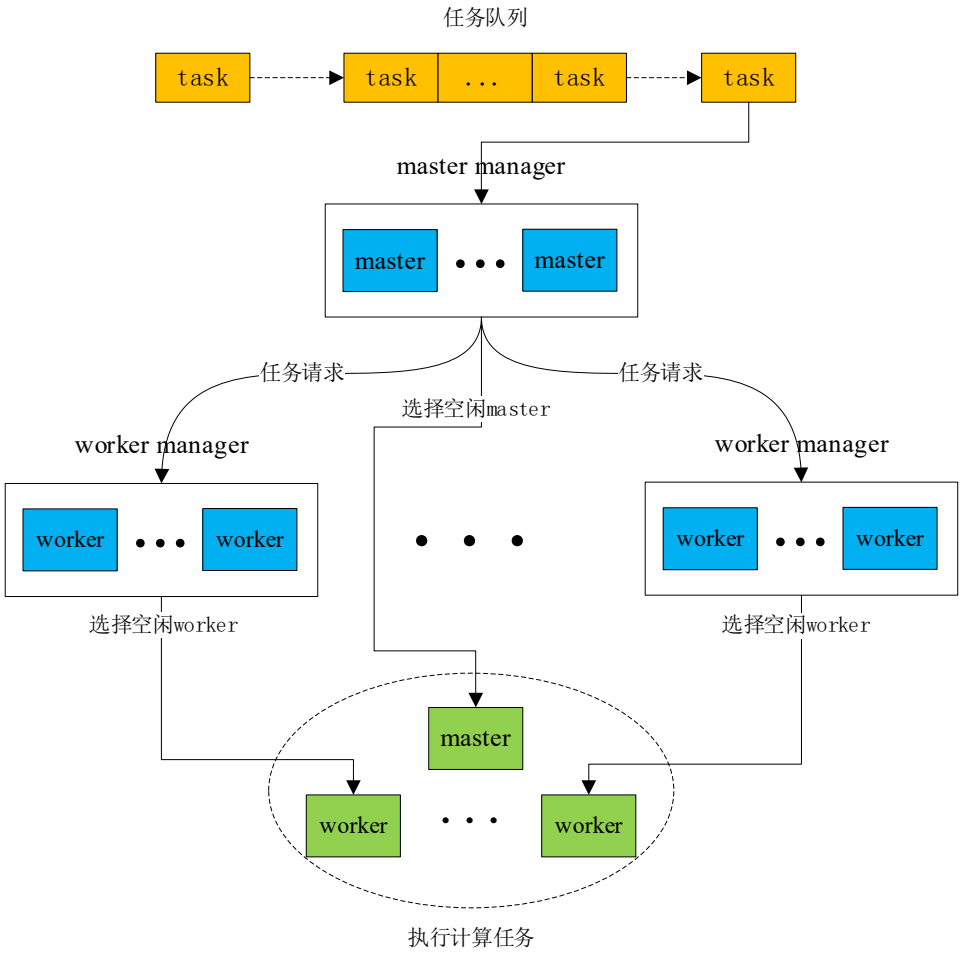

图 4-2 计算任务执行过程图

4.1.3 动态网络环境的模拟实现

动态网络环境的模拟过程在 BandWidthConfigModule 中实现。该模块可供所

有的节点访问，通过共享指针调用。在程序启动后，该模块初始化时将读取名为 bandwidth.config 的配置文件来获取网络环境的配置信息，这些信息包括网络分布情况的类型，网络带宽的更新频率，以及各种分布模型的具体参数比如指数分布中形状参数等。该模块包含成员变量和成员函数如下所示。

```
class BandWidthConfigModule {
public:
    BandWidthConfigModule(const std::string& path);
    ~BandWidthConfigModule();
    std::atomic<double> m_w_bw_;          // bandwidth between master and workers
    std::atomic<double> w_w_bw_;          // bandwidth between worker and worker
    std::atomic<double> broadcast_bw_;    // bandwidth when worker broadcasts
    std::atomic<double> max_bw_;          // max bandwidth, used by barrier
    double GetBW(BWType bw_type);
private:
    void DynamicAdjustBandWidth();
    // responsible for dynamic adjustment of bandwidth
    std::thread dynamic_adjust_bw_thd_;
    BandWidthDistributionType bw_distribution_type_;
    int interval_;
};
```

该模块的几个关键成员 m_w_bw_，w_w_bw_，broadcast_bw_，max_bw_分别用来表示 master 节点和 worker 节点之间的通信带宽，worker 节点之间的通信带宽，节点间的广播带宽，以及一个最大带宽值。其中广播带宽表示进行编码多播时的网络带宽，前三个带宽值主要表示传输 Shuffle 阶段中间结果时采用的带宽值。最大带宽值主要用于运行时传递一些与计算任务和 Shuffle 阶段都无关的、用于同步的必要信息时的带宽。节点在通过 simgrid::s4u::Mailbox 型成员向其他节点传递消息之前将根据其通信类型向 BandWidthConfigModule 通过 GetBW 成员函数查询此次通信的网络带宽。

动态网络环境的调节工作由 BandWidthConfigModule 的成员线程 dynamic_adjust_bw_thd_负责完成。该线程将执行 DynamicAdjustBandWidth 函数，它每隔一个时间间隔便被唤醒，根据指定的网络带宽分布类型，对上述提到的四

种类型的带宽值进行动态调整。BandWidthConfigModule 模拟的网络分布情况在配置文件中进行配置，主要支持三种分布：均匀分布，指数分布和基于 Pareto 分布。根据[29]，指数分布和 Pareto 分布可以模拟真实的网络带宽分布情况。唤醒的时间间隔为成员变量 interval_。

4.2 由在线学习驱动的编码分布式计算方案的实现

由在线学习驱动的编码分布式计算方案通过动态地调节编码分布式计算方案实现对动态变化的网络环境的自适应，调节的方向根据运行时执行的微实验的性能分析决定。这一部分的工作由 OnlineLearningModule 完成，该模块包含的成员如下所示。OnlineLearningModule 从任务队列中接收任务，任务队列表示为成

```
class OnlineLearningModule {
public:
    void DoWork();
private:
    void DoMicroExperient();
    void UpdateRAccordingToABTest(
        const UtilityInfo& u_1_a, const UtilityInfo& u_1_b,
        const UtilityInfo& u_2_a, const UtilityInfo& u_2_b);
    double GetUtility(const UtilityInfo& utility_info) const;
    std::shared_ptr<ConcurrencyQueue> job_queue_;
    std::shared_ptr<MasterManager> master_manager_;
    int r_;
    double eta_ = 0.1;
    double eta_min_ = 0.01;
    double eta_max_ = 0.05;
    int max_r_;
    double interval_ = 10.0;
};
```

员变量 job_queue_，它是一个一致性队列，可供多个线程互斥访问。OnlineLearningModule 通过 MasterManager 共享指针将任务传递给 MasterManager 进行执行。该方案的执行过程图如图 4-3 所示。OnlineLearningModule 在运行过程中会将一组用于微实验的任务交由 MasterManager 执行，同时对这些任务分别

指定负载因子 r 的数值，每一组微实验分别尝试一个在当前系统运行的负载因子 r 的基础上的一个更大的和更小的值，具体的计算方式可见 3.3 节。

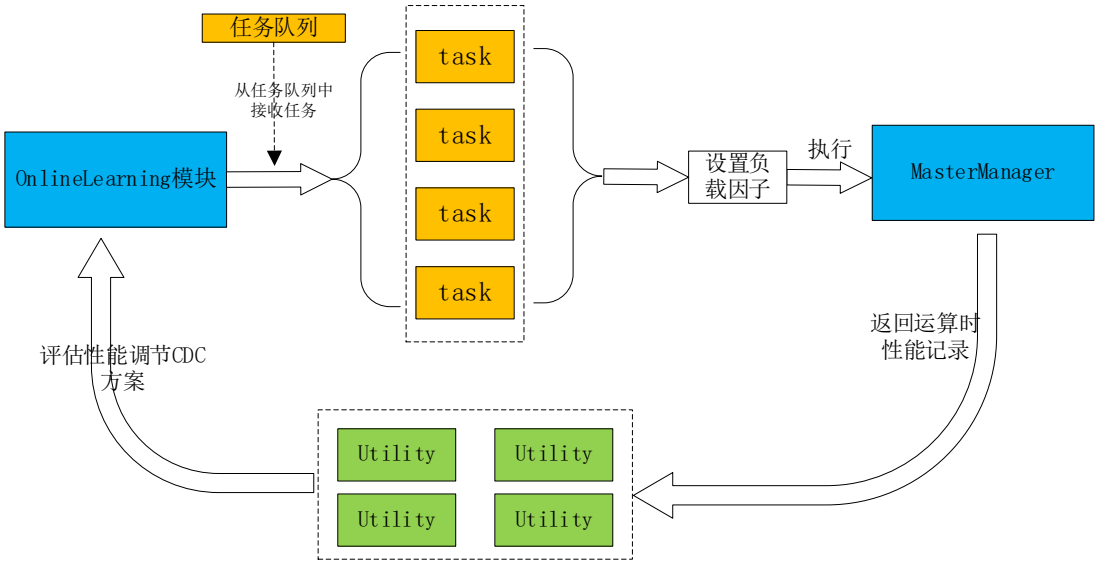

图 4-3 由在线学习驱动的 CDC 方案执行过程图

在将用于微实验的任务交由 MasterManager 之后，OnlineLearningModule 将等待这些任务的运行结束，此时仍然可以从任务队列中将接收任务交由其余的空闲 Master 执行正常的计算流程。MasterManager 在根据指定的任务上下文执行任务后将把执行时的性能指标记录保存为一个 UtilityInfo 对象，一个 UtilityInfo 变量包含的主要成员如下所示，这些变量分别表示执行该计算任务的总执行时间，

```
struct UtilityInfo {
    double time;
    double network_load;
    double computation_load;
};
```

计算开销和通信开销。其中计算开销由 Map 阶段的执行时间表示，通信开销由 Shuffle 阶段的执行时间表示。之后 OnlineLearningModule 将根据 GetUtility 函数这些性能记录依据聚合公式（见 3.3 节）聚合为一个效用值，并调用 UpdateRAccordingToABTest 成员函数根据指定的比较规则决定接下来的调整方向。效用值的聚合方式和比较规则可见 4.2 节。在本次调整结束之后，将继续重复执行下一轮的实验和调整流程。

4.3 本章小结

本章详细介绍了本课题的动态自适应并行计算方案的实现过程。首先介绍了在 SimGrid 模拟平台上如何实现了一个基于 MapReduce 的并行计算系统。之后介绍了该系统的动态网络环境是如何模拟实现的。然后，在此系统的基础上完成了第三章提出的方案，实现了由在线学习驱动的动态编码分布式计算方案，该方案监控和收集计算任务在执行过程中的计算和通信开销，通过执行微实验的方式探查调节动作对系统的实际执行性能的影响，从而向更适应网络环境的方向对系统进行调整。

5 测试与实验结果分析

本章节将首先介绍本课题的测试环境，并对 4.1 节和 4.2 节中实现的原型系统和优化方案进行性能的测试，并且展示测试结果。

5.1 测试环境

本课题的方案在 SimGrid 模拟平台上编程实现并进行测试。具体的编程环境，测试环境以及模拟平台的具体配置信息如表 5-1 所示。

表 5-1 环境配置信息

配　置	具体信息
CPU	Intel i7-10510U (8) @ 2.304GHz
内存	16GB
操作系统	Ubuntu 20.04.6 LTS
模拟平台	SimGrid v3.35
编译器	cmake 3.27.0-rc1
开发语言	c++14

5.2 功能性测试

参照[7]中的测试部分，本课题的测试环节运行的计算任务为 Hadoop 排序基准测试 TeraSort。TeraSort[2]是一种分布式的排序算法，排序数据以键值对的形式保存，TeraSort 根据输入数据的键以及指定的键的比较规则进行排序。计算任务的数据通过 TeraGen 程序生成，TeraGen 是 Apache Hadoop 开源软件框架中的一个 map/reduce 程序，生成的每个键值对由 10 个字节的键和 90 个字节的值构成。

首先为该系统进行功能性测试，展示其可以正常地指定给定的计算任务。在该系统上执行一个 TeraSort 任务的一部分输出结果如图 5-1 所示，只展示了节点 1 和节点 2 的部分输出结果。其中最左侧的编号为节点 id，最右侧的 10 字节为键值对中的键，键值对中的值并未展示在输出结果中。可见通过执行 TeraSort 任

[2] https://github.com/apache/hadoop/tree/trunk/hadoop-mapreduce-project/hadoop-mapreduce-examples

务可将输入数据成功按照键的字节序排列。

```
[1] [key] 2a 60 87 28 20 e9 fe 4d 72 8b
[1] [key] 2a 66 ea d4 52 90 af d8 03 b4
[1] [key] 2a 6f 51 ff 19 6d 39 99 0a b2
[1] [key] 2a 72 ea 17 f2 00 14 b8 0c 81
[1] [key] 2a 7d 87 af 72 c0 7e 8c 7e d9
[2] [key] 2a 7e 41 1e 7f e0 00 01 0a 55
[2] [key] 2a 7f 36 68 03 a2 3c aa f5 74
[2] [key] 2a 82 2d eb 99 3d 58 85 da 8a
[2] [key] 2a 89 65 e6 1d 23 1f 5e e0 cd
[2] [key] 2a 8f 1b 7d 83 66 b8 0c ac e6
```

图 5-1 TeraSort 运行结果

5.3 性能测试

接下来将进行性能测试。每次测试将运行 128 个 TeraSort 计算任务。分别在基于均匀分布、指数分布、Pareto 分布三种网络带宽分布情况下，对 MapReduce 方案，编码分布式计算方案（以下简称 CDC 方案）和本课题实现的由在线学习驱动的动态编码分布式计算方案（以下简称 DCDC 方案）的性能进行测试。

首先在基于离散型均匀分布，指数分布和 Pareto 分布的网络环境下，测试了系统在执行计算任务的过程中通信开销和计算开销的比值的变化情况，这三种网络分布情况的运行结果分别如图 5-1，图 5-2 和图 5-3 所示。在三种网络分布

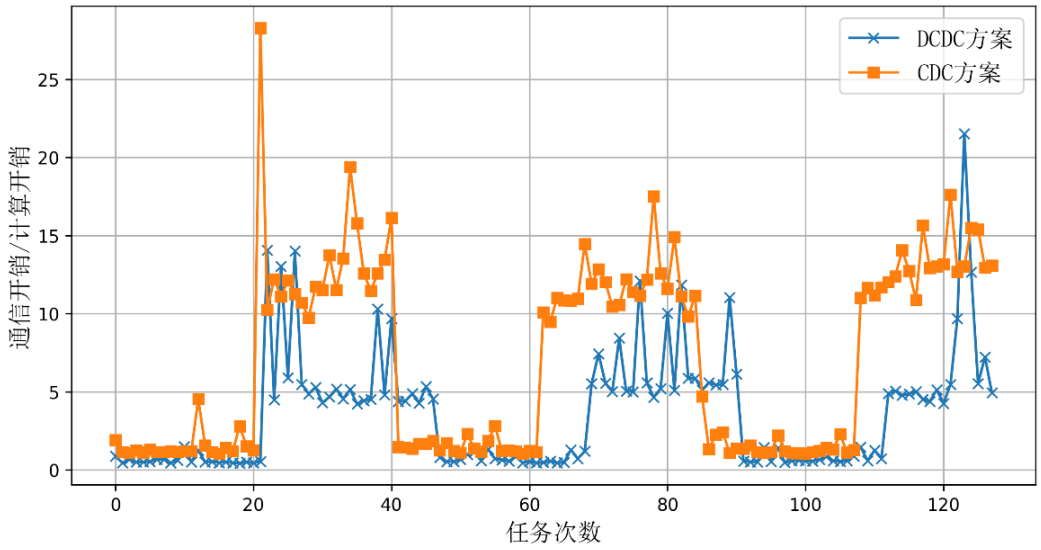

图 5-2 基于离散型均匀分布的网络环境下 CDC 方案和 DCDC 方案的通信开销的比值变化情况

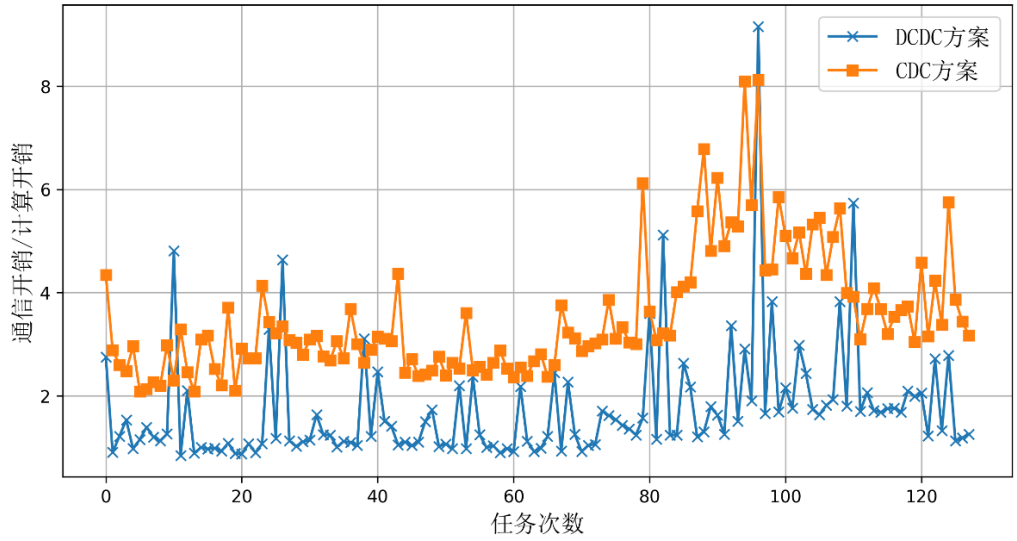

图 5-3 基于指数分布的网络环境下 CDC 方案和 DCDC 方案的通信开销与计算开销的比值变化情况

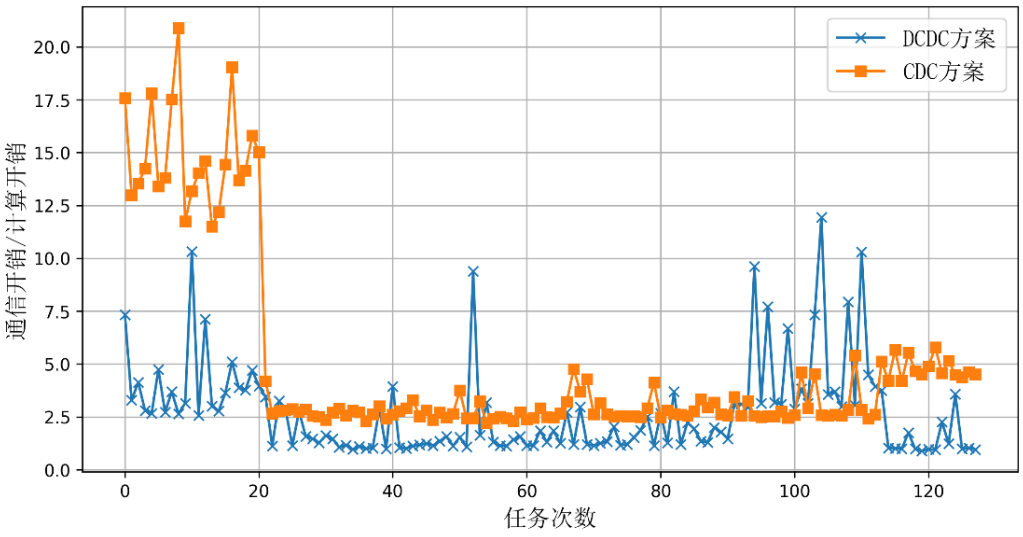

图 5-4 基于 Pareto 分布的网络环境下 CDC 方案和 DCDC 方案的通信开销与计算开销的比值变化情况

情况下，相比 CDC 方案，DCDC 方案在执行时大部分的计算任务的通信开销和计算开销的比值更接近于 1。随着网络带宽的动态变化，两种方案的比值都产生一定程度的波动，会比较频繁地出现骤增和骤降的现象。但是经过动态的调节，DCDC 方案的通信开销和计算开销的比值可以快速地回落到比较理想的状态。

更具体地来看，在三种网络分布环境下，这两种方案的通信开销于计算开销的比值的方差如表 5-2 所示。方差的比较结果显示，在三种动态网络环境下，相

比 CDC 方案，DCDC 方案的通信开销和计算开销的比值更加的稳定。

表 5-2 通信开销和计算开销的比值的方差

网络分布情况	均匀分布	指数分布	Pareto 分布
CDC 方案	35.88	1.39	20.51
DCDC 方案	14.00	1.26	4.75

为了更详细地测试本方案在计算性能上的提升，分别在三种网络分布情况下，统计了 MapReduce 方案，CDC 方案和 DCDC 方案执行 128 个计算任务的时间。具体的计算任务的数据同样通过 TeraGen 生成。三种方案的运行时间的对比情况如图 5-4 所示，更详细的测试数据如表 5-3 所示。分别在三种网络带宽分布情况下，DCDC 方案相比 MapReduce 方案在任务执行时间上分别提升了 56.33%，51.63%和43.51%，相比CDC方案执行时间分别提升了29.95%，33.69%和32.88%。由此可见，在指定分布情况的动态网络环境下，相比于 MapReduce 方案和静态的应用编码分布式计算方案，对它进行动态的调整可以显著地提升系统的计算性能。

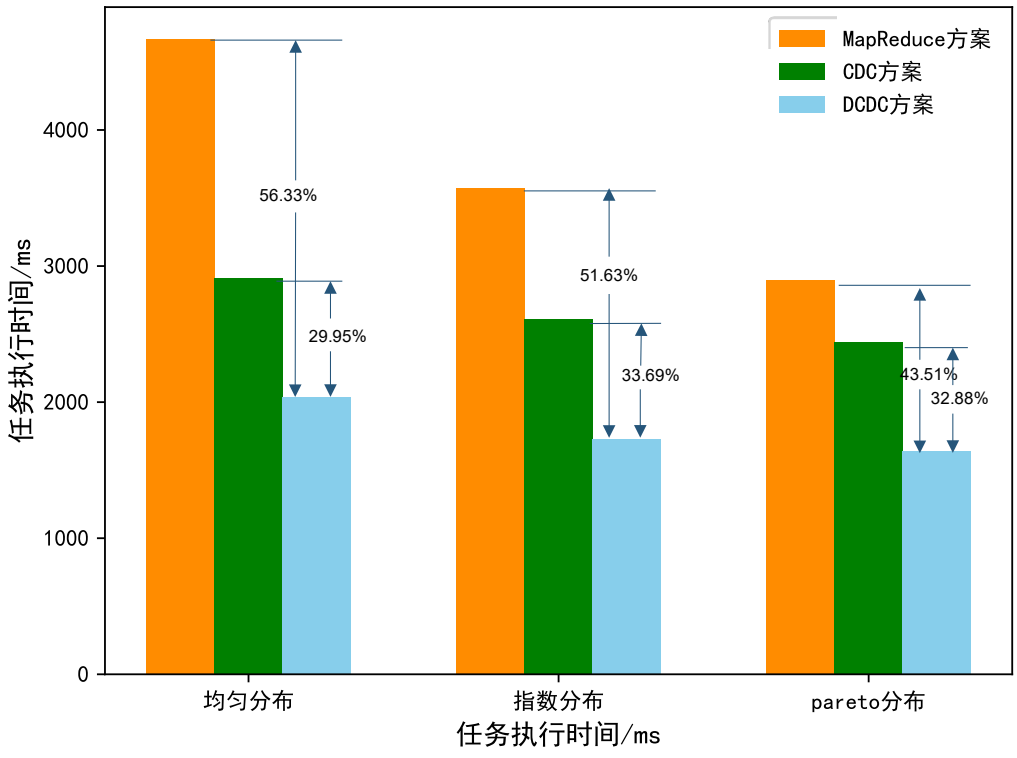

图 5-5 任务执行时间对比图

表 5-3 三种网络分布情况下的运行时间

网络分布情况	均匀分布	指数分布	Pareto 分布
MapReduce 方案执行时间/ms	4669.495	3572.682	2898.367
CDC 方案执行时间/ms	2910.585	2606.088	2439.630
DCDC 方案/ms	2038.824	1727.899	1637.348
DCDC 方案相比 MapReduce 方案	56.33%	51.63%	43.51%
DCDC 方案相比 CDC 方案	29.95%	33.69%	32.88%

5.4 本章小结

本章首先介绍了方案的测试环境的配置信息。之后具体阐述了方案的测试过程以及测试结果。分别在基于离散型均匀分布，指数分布，Pareto 分布三种网络环境下，对 MapReduce 方案，CDC 方案和 DCDC 方案的性能进行测试。测试结果表明：

（1）在任务的执行时间方面，DCDC 方案的计算性能相比 MapReduce 方案提升了 50%左右，相比 CDC 方案提升了 30%左右，可见对编码分布式计算的动态调整是有利于任务在动态网络环境中的执行的；

（2）通信开销和计算开销的比值情况折线图表示，DCDC 方案相比 CDC 方案具有对网络更好的适应性，在网络环境进行变化时可以更好的对系统进行自适应的调节。比值的方差的比较结果也表示，DCDC 方案的二者的开销比更加稳定。在动态网络环境下本课题的调节方案可以使二者的开销尽可能的更加接近，从而达到通信开销和计算开销的更好的平衡。

6 总结与展望

面向动态变化的网络环境的并行计算方案聚焦于大规模并行计算系统中由于故障频发的网络环境造成的计算性能降低的问题。网络带宽不稳定的环境下，在基于 MapReduce 并行计算框架的系统中传输中间结果的过程会产生延迟。课题的关键在于如何降低传输中间结果的开销，以及如何根据系统实际的计算开销和通信开销对动态变化的网络环境实现自适应。为了解决这个问题，主要做了以下工作：

（1）首先介绍了 MapReduce 计算框架和当前并行计算领域内的相关技术，主要是关于通信负载和计算负载不均衡方面的优化。并且介绍了本课题的方案网络编码和编码分布式计算技术；

（2）在 SimGrid 并行计算模拟平台编程实现了一个基于 MapReduce 的并行计算原型系统，并在此基础上添加动态网络调节模块，并在基于离散型均匀分布、指数分布、Pareto 分布的网络环境下对该系统的计算开销和通信开销的变化情况做了调研；

（3）为了使系统的计算开销和通信开销达到动态的平衡，在该原型系统上实现了由在线学习驱动的动态编码分布式计算方案。该方案通过执行微实验的方式实现对动态网络环境的自适应；

（4）在三种网络分布环境下，分别对 MapReduce 方案，静态的编码分布式计算方案和本课题的方案的性能进行了测试。测试结果表明，相比静态的编码分布式计算方案，本课题的方案在动态网络环境下可以更高的实现计算开销和通信开销之间的平衡。在计算性能方面，本课题的方案相比 MapReduce 方案提升了约 50%，相比静态的编码分布式计算方案提升了约 30%。

然而，本课题仍然存在着很多问题。在统计节点的通信开销和计算开销时，只是简单地将 Map 阶段的执行时间和 Shuffle 阶段的执行时间代指开销，这在一些情况下这些数值并不能反映系统的实时信息。在评估任务的执行性能方面，采用的效用函数只是计算开销和通信开销的线性组合，没有进行更多种类的函数的尝试和比较。受限于机器的性能，在模拟平台上进行测试时系统的节点数也没有

达到很大的规模。

总的来说，本课题通过由在线学习的编码分布式计算方案，实现了MapReduce 并行计算系统对动态网络的自适应。自己对并行计算领域内的相关技术有了一定程度的了解和认识，希望在将来的工作中能进一步完善自己的不足。

致 谢

至此毕业设计工作即将完成。首先要感谢我的导师胡燏翀教授。在完成毕业设计的过程中，他从选题、方案的确定、方案的测试和论文的撰写等方面给了我细致的指导和意见。

在华中科技大学的四年中，我收到来自老师和同学的许多帮助和支持，让我顺利地度过了本科生活。

感谢我的父母和家人，他们自我幼时给我提供了家庭的温暖和呵护，让我成长为一个健康、积极的人。

最后，希望自己可以在之后的研究生生涯中克服自己的缺点，弥补自己的不足，在崭新的篇章中做出新的成绩。

参考文献

[1] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[2] 肖嘉豪，李颂华."基于 Hadoop 的 MapReduce 架构研究".中国计算机用户协会网络应用分会 2021 年第二十五届网络新技术与应用年会论文集.Ed..《计算机科学》编辑部, 2021, 408-411.

[3] White, Tom. Hadoop: The definitive guide. " O'Reilly Media, Inc.", 2012.

[4] Hedayati, Soudabeh, et al. "MapReduce scheduling algorithms in Hadoop: a systematic study." Journal of Cloud Computing 12.1 (2023): 143.

[5] Chowdhury, Mosharaf, et al. "Managing data transfers in computer clusters with orchestra." ACM SIGCOMM computer communication review 41.4 (2011): 98-109.

[6] Zhang, Zhuoyao, Ludmila Cherkasova, and Boon Thau Loo. "Performance modeling of mapreduce jobs in heterogeneous cloud environments." 2013 IEEE Sixth International Conference on Cloud Computing. IEEE, 2013.

[7] Li, Songze, et al. "A fundamental tradeoff between computation and communication in distributed computing." IEEE Transactions on Information Theory 64.1 (2017): 109-128.

[8] Lee, Kangwook, et al. "Speeding up distributed machine learning using codes." IEEE Transactions on Information Theory 64.3 (2017): 1514-1529.

[9] Li, Songze, Mohammad Ali Maddah-Ali, and A. Salman Avestimehr. "Coded distributed computing: Straggling servers and multistage dataflows." 2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE, 2016.

[10] Guo, Leitao, Hongwei Sun, and Zhiguo Luo. "A data distribution aware task scheduling strategy for mapreduce system." Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1. Springer Berlin Heidelberg, 2009.

[11] Borthakur, Dhruba. "HDFS architecture guide." Hadoop apache project 53.1-13

(2008): 2.

[12] Wang Y, Zhang Y, Su Y, et al. An adaptive and hierarchical task scheduling scheme for multi-core clusters[J]. Parallel computing, 2014, 40(10): 611-627.

[13] Gufler B, Augsten N, Reiser A, et al. Load balancing in mapreduce based on scalable cardinality estimates[C]//2012 IEEE 28th International Conference on Data Engineering. IEEE, 2012: 522-533.

[14] Li J, Liu Y, Pan J, et al. Map-Balance-Reduce: An improved parallel programming model for load balancing of MapReduce[J]. Future Generation Computer Systems, 2020, 105: 993-1001.

[15] 李冬月,尹铁源."基于 MapReduce 的负载均衡策略研究." 信息与电脑(理论版) 36.02(2024):177-179. doi:CNKI:SUN:XXDL.0.2024-02-043.

[16] Fan Y, Liu W, Guo D, et al. Shuffle scheduling for MapReduce jobs based on periodic network status[J]. IEEE/ACM Transactions on Networking, 2020, 28(4): 1832-1844.

[17] Hou X, Kumar T K A, Thomas J P, et al. Dynamic workload balancing for hadoop mapreduce[C]//2014 IEEE Fourth International Conference on Big Data and Cloud Computing. IEEE, 2014: 56-62.

[18] Karandikar, Shrikrishna, et al. "TCP rate control." ACM SIGCOMM Computer Communication Review 30.1 (2000): 45-58.

[19] Dong M, Li Q, Zarchy D, et al. {PCC}: Re-architecting congestion control for consistent high performance[C]//12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). 2015: 395-408.

[20] Dong M, Meng T, Zarchy D, et al. {PCC} vivace:{Online-Learning} congestion control[C]//15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18). 2018: 343-356.

[21] Jay N, Rotman N H, Godfrey P, et al. Internet congestion control via deep reinforcement learning[J]. arXiv preprint arXiv:1810.03259, 2018.

[22] Kyzy, Aliyeva Matanat Latif, and Misirli Roza Rizvan Kyzy. "A REVIEW OF BIG DATA MANAGEMENT, BENEFITS AND CHALLENGES." Journal of Monetary Economics and Management 1 (2023): 8-14.

[23] Zaharia, Matei, et al. "Spark: Cluster computing with working sets." 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10). 2010.

[24] Sleeman IV, William C., and Bartosz Krawczyk. "Multi-class imbalanced big data classification on spark." Knowledge-Based Systems 212 (2021): 106598.

[25] Matsuda, Takahiro, Taku Noguchi, and Tetsuya Takine. "Survey of network coding and its applications." IEICE transactions on communications 94.3 (2011): 698-717.

[26] Ahlswede, Rudolf, et al. "Network information flow." IEEE Transactions on information theory 46.4 (2000): 1204-1216.

[27] Gabriel, Edgar, et al. "Open MPI: Goals, concept, and design of a next generation MPI implementation." Recent Advances in Parallel Virtual Machine and Message Passing Interface: 11th European PVM/MPI Users' Group Meeting Budapest, Hungary, September 19-22, 2004. Proceedings 11. Springer Berlin Heidelberg, 2004.

[28] Zhang, Andrew, et al. "Modin OpenMPI compute engine." Master's thesis, EECS Department, University of California, Berkeley (2021).

[29] Varet A, Larrieu N. How to generate realistic network traffic?[C]//2014 IEEE 38th annual computer software and applications conference. IEEE, 2014: 299-304.

[30] Salam, Abdus, Abdullah Khan, and Samad Baseer. "A comparative study for software cost estimation using COCOMO-II and Walston-Felix models." The 1st International Conference on Innovations in Computer Science & Software Engineering,(ICONICS 2016). 2016.

[31] Saleh, Ehab, and Chandrasekar Shastry. "Simulation and Modelling of Task Migration in Distributed Systems Using SimGrid." International Conference on Modeling, Simulation and Optimization. Singapore: Springer Nature Singapore, 2022.

[32] Casanova H, Legrand A, Quinson M. Simgrid: A generic framework for large-scale distributed experiments[C]//Tenth International Conference on Computer Modeling and Simulation (uksim 2008). IEEE, 2008: 126-131.