论文编号 00000

河南中医药大学

Henan University of Chinese Medicine

本科毕业论文

## 论文题目： 基于Nginx 服务器集群负载均衡技术的研究与改进

|  |  |  |
| --- | --- | --- |
| 姓 | 名： | 苏峻锋 |
| 院 | 系： | 尚真书院 |
| 专 | 业： | 医学信息工程 |
| 年 | 级： | 2020 级 |
| 学 | 号： | 2020189017 |
| 指导教师： | | 张格 |

评定成绩：

2024 年 4 月 18 日

# 毕业设计（论文）诚信声明书

本人声明：我将提交的毕业论文（设计）《 》是我在指导教师指导下独立研究、写作的成果，论文中所引用他人的无论以何种方式发布的文字、研究成果，均在论文中加以说明；有关教师、同学和其他人员对本文的写作、修订提出过并为我在论文中加以采纳的意见、建议，均已在我的致谢辞中加以说明并深致谢意。

|  |  |  |  |
| --- | --- | --- | --- |
| 论文作者 （签字） 时间： | 年 | 月 | 日 |
| 指导教师已阅 （签字） 时间： | 年 | 月 | 日 |

# 毕业设计（论文）版权使用授权书

本毕业论文《 》是本人在校期间所完成学业的组成部分，是在河南中医药大学教师的指导下完成的，因此，本人特授权对河南中医药大学可将本毕业论文的全部或部分内容编入有关书籍、数据库保存，可采用复制、印刷、网页制作等方式将论文文本和经过编辑、批注等处理的论文文本提供给读者查阅、参考，可向有关学术部门和国家有关教育主管部门呈送复印件和电子文档。本毕业论文无论做何种处理，必须尊重本人的著作权，署明本人姓名。

|  |  |  |  |
| --- | --- | --- | --- |
| 论文作者： （签字） 时间： | 年 | 月 | 日 |
| 指导教师已阅 （签字） 时间： | 年 | 月 | 日 |

|  |  |  |
| --- | --- | --- |
| 目 | 录 | |
| [摘要](#_bookmark0)⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [I](#_bookmark0) |
| [ABSTRACT](#_bookmark1) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [III](#_bookmark1) |
| 1 [绪论](#_bookmark2)⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [1](#_bookmark2) |
| 1.1 [研究背景及意义](#_bookmark3) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [1](#_bookmark3) |
| 1.2 [论文结构](#_bookmark4) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [3](#_bookmark4) |
| 2 [相关理论及技术研究](#_bookmark5)⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [4](#_bookmark5) |
| 2.1 [服务器集群](#_bookmark6) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [4](#_bookmark6) |
| 2.2 [负载均衡技术](#_bookmark7) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [9](#_bookmark7) |
| 2.3 [Nginx 服务器](#_bookmark9)⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [13](#_bookmark9) |
| 2.4 [本章小结](#_bookmark11) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [22](#_bookmark11) |
| 3 [网站访问量时间序列预测研究](#_bookmark12)⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [23](#_bookmark12) |
| 3.1 [时间序列预测基本方法](#_bookmark13) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [23](#_bookmark13) |
| 3.2 [访问量的时间序列预测算法](#_bookmark14) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [25](#_bookmark14) |
| 3.3 [两种算法的比较以及选择](#_bookmark18) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [35](#_bookmark18) |
| 3.4 [本章小结](#_bookmark20) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [35](#_bookmark20) |
| 4 [基于时间卷积网络的剩余性能的预测](#_bookmark21)⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [36](#_bookmark21) |
| 4.1 [服务器剩余性能的获取](#_bookmark22) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [36](#_bookmark22) |
| 4.2 [时间卷积网络的关键特征](#_bookmark23) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [39](#_bookmark23) |
| 4.3 [时间卷积网络与负载情况的结合](#_bookmark25) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [45](#_bookmark25) |
| 4.4 [本章总结](#_bookmark27) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [49](#_bookmark27) |
| 5 [预测模型与传统负载均衡的结合](#_bookmark28)⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [50](#_bookmark28) |
| 5.1 [传统负载均衡算法的选取](#_bookmark29) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [50](#_bookmark29) |
| 5.2 [改进的负载均衡算法的提出](#_bookmark32) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [53](#_bookmark32) |
| 5.3 [负载均衡算法的优化](#_bookmark35) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | | [56](#_bookmark35) |

|  |  |
| --- | --- |
| 5.4 [本章总结](#_bookmark39) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | [61](#_bookmark39) |
| 6 [总结与展望](#_bookmark40)⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | [62](#_bookmark40) |
| 6.1 [本文工作总结](#_bookmark41) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | [62](#_bookmark41) |
| 6.2 [工作展望](#_bookmark42) ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ | [63](#_bookmark42) |

[参考文献](#_bookmark43)⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ [64](#_bookmark43)附录 A [附录 A 相关模型算法实现主要代码](#_bookmark91)⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ [67](#_bookmark91)[致谢](#_bookmark92)⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ ⋅ [69](#_bookmark92)

基于 Nginx 服务器集群负载均衡技术的研究与改进

苏峻锋摘要

互联网的发展迅速，且携带的信息量巨大，信息交流速度迅速，自由程度高，并且实现了全球的信息共享。但同时用户增加导致访问量暴增，很大成程度上增加了网络服务器的压力，同时也对服务器的性能提出了更高的要求。由于大量的 Web 高并发访问对后台的服务器造成的压力问题亟待解决。由此应运而生的服务器集群系统和负载均衡技术很大程度上缓解了这个问题，并且这些技术一方面改善了服务器的性能，另一方面很大幅度地缩减了改善服务器性能所需要的开销。现在服务器上应用的负载均衡技术有很多，有的分配策略也存在不足，可能会影响繁忙的服务器处理任务的同时还接收着大量访问请求，而部分服务器却处于空闲状态未分配到请求任务，这种现象会影响整个服务器集群对高并发访问的响应速度，使得用户体验变差。

针对服务器集群中负载均衡策略存在的分配不合理问题，本文研究了基于

Nginx 高并发服务器的负载均衡算法，其主要工作如下：

1. 基于服务器网站访问量的时间序列预测研究。主要工作，对传统负载均衡算法只能被动，不能主动进行防御的缺陷，其访问量激增导致的服务器集群性能下降甚至宕机的问题，通过使用数学分析和机器学习算法对网站未来访问量进行预测，根据预测结果对服务器节点进行调整，提升集群的稳定性。并利用Kaggle 比赛“Web Traffic Time Series Forcasting”所提供的网站访问量数据集，对数据进行分析和预测，并通过两个模型的对比，确定了合适的预测模型，使得负载均衡算法具有主动探测的能力。
2. 对服务器集群各个节点在一天内的总体综合负载指标变化趋势进行预测。选取合适的指标作为能够体现综合负载情况的依据，并在各个节点不断收集负载信息数据，将得到的各个节点的负载数据进行分析和生成时间序列样本数据，使用卷积神经网络建立综合负载预测模型。并将其预测结果作为后面负载均衡策略中权重调整的依据。
3. 基于 Nginx 高并发访问的负载均衡的改进。针对 Nginx 服务器传统的负载均衡无法真正得到不同服务器节点的真实负载情况的问题，提出了一种改进

的动态负载均衡算法对用户的任务进行合理的权重分配。创新的使用两个阶段来解决神经网络初期没有数据的问题，同时保证了集群的稳定运行。另一个阶段采用了对节点的综合负载指标预测结果来判断是否调整某个服务器当前的权重。参考了 TCP 拥塞控制的原理，来对各个服务器节点的权重进行调整，从而达到控制节点负载状态的目的。

关键词：Nginx；时间卷积网络；负载均衡，时间序列预测算法

Research and Improvement of Nginx-based Server Cluster Load Balancing Technology

Su JunFeng ABSTRACT

The rapid development of the internet carries a massive volume of information, enables swift communications, fosters a high degree of freedom, and has achieved global infor- mation sharing. However, the increasing number of users has led to a surge in access demand, significantly burdening network servers and raising the performance standards required of them. The pressure on backend servers due to the high concurrency of web access urgently needs to be addressed. Server cluster systems and load balancing tech- nology have emerged in response to this issue, greatly mitigating it. These technologies not only enhance server performance but also substantially reduce the costs associated with server performance improvements. Many load balancing technologies are cur- rently applied to servers, yet some distribution strategies are insufficient, potentially affecting busy servers that continue to receive an overload of access requests while other servers remain idle with no assigned tasks. This disparity can affect the entire server cluster’s response time to high-concurrency access, leading to a degraded user experience.

In response to the issue of improper distribution in server cluster load balanc- ing strategies, this paper investigates a load balancing algorithm based on the high- concurrency server Nginx, with the following main contributions:

* 1. Research on time-series forecasting of website traffic volume. This addresses the traditional load balancing algorithms’ passive nature by employing mathematical analysis and machine learning algorithms to predict future website traffic. Adjust- ments to server nodes based on these predictions enhance cluster stability. Using the dataset provided by the Kaggle competition ”Web Traffic Time Series Forecasting,” data analysis and prediction are conducted, and through the comparison of two models, an appropriate forecasting model is selected, imbuing the load balancing algorithm with proactive detection capabilities.
  2. Prediction of the overall composite load index trends for each node in a server cluster throughout a day. Suitable indicators reflecting the composite load are selected, and load data are continuously collected from each node. This data is analyzed and used to generate time-series sample data, and a convolutional neural network is employed to establish a comprehensive load prediction model. The results of this prediction serve as the basis for weight adjustments in subsequent load balancing strategies.
  3. Improvements to load balancing under high-concurrency access based on Ng- inx. Addressing the issue of traditional Nginx server load balancing algorithms failing to accurately ascertain the real load conditions of different server nodes, a novel dy- namic load balancing algorithm proposes a rational weight distribution of user tasks. An innovative two-phase approach overcomes the lack of initial data for neural net- works while ensuring cluster stability. One phase involves adjusting the weight of a server based on predicted comprehensive load index results for that node. Drawing on the principles of TCP congestion control, adjustments to the weight of each server node are made to manage the node’s load condition effectively.

**Keywords:** Nginx; Time Convolutional Network; Load Balancing; Time Series Forecasting Algorithm

# 绪论

## 研究背景及意义

* + 1. 研究背景

随着互联网的高速发展和广泛普及，人们的生活与互联网已密不可分。从年轻的数字原住网民到如今老年人的网络化，都在积极参与数字世界，展现出强烈的社交化、互动性和创造力，形成了独特的网络文化。根据中国权威调查中国互联网络信息中心调查研究第 52 次《中国互联网络发展状况统计报告》显示，截止 2023 年 6 月，我国网民规模达 10.79 亿次，较 2021 年 12 月增长 3549万，互联网普及率达 75.6%[[1](#_bookmark44)]

网络范围的不断扩大，服务人民的服务器日益增加，利用互联网收发消息，进行娱乐视频的行为越来越多。在这种境地下，对后台服务器的处理能力提出了严峻的挑战。前人发明了许多方法来应对这种挑战。疏导，引流，集群，但是随着集群规模的增加，也就是服务器设备的不断增加。服务器的不同应用还会频繁产生互动行为, 在这种互动行为中，集群中服务器的交流，远程调用是需要解决的第一个难题，而且当用户提交的请求不断增加时，怎么样保持节点的负载状态稳定？当用户发出请求任务，为了保证节点的的负载状态稳定，多个服务器就会采取一种策略对任务进行分配，以合理的调取资源，这就是负载均衡算法。

由于网络请求量不停地增长，要提高服务器处理请求的性能可以从两个方面提升。一方面是硬件，也被称为硬件负载均衡。增加后端服务器节点的数量来达到分摊负载的目的，提高集群整体的负载能力。但是该方法需要的成本巨大，除了一些大型企业有能力购买之外，对于其他中小型企业来说负担太重。而且即便花费了企业资金买进了很多高性能的服务器，可如果无法使这些设备完全展现出它们的能力，也会使得性价比降低。但是显然一些中小型企业是不会这样做的，他们选择从另一方面进行优化，即软件算法上的优化，这样既节省资源还可以提高服务器性能。[[2](#_bookmark45)]

综上所述，除了大型企业，其它类型的企业从软件方面进行改进是一个可行的方案，而且对于大型企业来说进行软件方面的改进可以使庞大的服务器集群的性能更上一层楼。软件方面进行的负载均衡被称为软负载均衡，在一台或者多台服务器上不同的操作系统中安装一个或者多个附加软件以实现负载均衡。

这种方法最大的特点就是配置可能更加简单，价格比较亲民。

对于软负载均衡常见的有 LVS[[3](#_bookmark46)], Nginx[[4](#_bookmark47)], HAProxy[[5](#_bookmark48)]，Ribbon 等等。其中， Nginx 是一款优秀的软件负载均衡器，具有并发量高代码开源等优点，因此常常被用来作为服务器端的负载均衡器。本文也是基于 Nginx 负载均衡系统的研究，在研究的基础上探究负载均衡算法并创造优化算法

* + 1. 国内外研究现状

几十年来，互联网发展突飞猛进，Web 服务器所需要完成的并发请求量日益增加，带来的就是服务器处理压力不断增加，在除了硬负载均衡的进步以外，在软负载均衡研究中国内外也提出了并实现了一系列的优化，出现了一系列服务器软件，最为著名的就是 Nginx。

在国外，XIAONICHI 等人[[6](#_bookmark49)]，分析研究了负载均衡技术，参考前人运用这些技术手段做过的一些成功案例，并且重点研究了 Nginx 的主要功能模块和内核，运用 Nginx 作为反向代理服务器来解决并发难题。运用负载均衡技术和缓存技术来应对出现服务器过载的情况。并且结合实际情况来设计网络拓扑架构，将 Discuz 论坛应用到该网络架构中，实验仿真分析后得出的结果，证明了基于 Web 缓存技术可以很好的解决高并发的问题。

同时，国外的一些企业也一直研究用户集群系统的负载调度器。Cisco 的 LocalDirect 推出最快响应算法和最小连接数算法。IBM 的 NetworkDispatcher 选择当前连接数最少的服务器处理新的请求任务。Intel 公司网擎的负载调度策略是采用快速响应算法[[7](#_bookmark50)]。

在国内，2016 年，王永辉采用集群服务器分组管理算法，在各服务器周期性采集完负载指标信息后向集群控制节点上报，由控制节点向负载均衡器统一反馈，负载均衡器针对负载信息计算每个服务器的加权值，并选出权值最大的三台服务器，以正比于权值的概率随机选择一台服务器处理客户端请求[[8](#_bookmark51)]。

2018 年，王东提出一种基于 UDP 协议多播实现服务器负载指标数据传输的动态负载均衡算法，该算法将一段时间内多核 CPU 的平均工作负载作为阈值对多核 CPU 是否达到最大工作负载进行判断，由此负载均衡器将减少 CPU 处于满载的服务器的任务量[[9](#_bookmark52)]。实验证明该算法有效提升了系统的吞吐率，但 UDP传输方式仍然存在一定的可靠性问题。

2021 年谭畅，胡磊等人[[10](#_bookmark53)]，提出了一种动态调节权重的负载均衡策略。该策略基于加权轮询策略进行改进，同时考虑服务器的本身硬件性能与工作时的

负载情况，经过实验测试结果表明相较于原加权轮询算法，该算法在高并发情况下的响应时间和实际并发数等方面表现更好。

* + 1. 研究目的与意义

在阅读他人或企业组织关于负载均衡的论文时，发现了许多有关于使用机器学习，深度学习，神经网络等方法实现负载均衡算法的预测从而处理高并发，降低负载率。

2012 年通过数学多元回归分析方法计算得到系统的响应时间和反映节点负载状态两个参数之间存在的联系，并通过这个联系建立关系模型进行服务器负载的预测和集群的负载均衡。同年，吴伟实现了一种基于 BP 神经网络的动态负载均衡算法根据服务器的负载情况，预测请求的处理单元时间，并结合当先处理的请求数，进行其他的调度，两种方法其实都是预测的方法。

2020 秦娥基于服务器网站当前访问量的时间序列来预测未来高峰期的访问量[[2](#_bookmark45)]，对服务器集群中各个节点在一天内总体剩余性能变化的趋势进行预测，从而调整服务器集群的节点进行调整，提升了集群稳定性。

根据所学习的相关内容，并结合大学所进行的课程，使用人工智能分析访问量，并发量，并分析研究内置负载均衡算法；通过比较使用两者后的测试结果，看是否能降低服务其的负载度和响应时间，进而给用户良好的网络体验，具备在实际应用中的参考价值。

## 论文结构

本文详细结构如下：

第 1 章首先论述了本论文的研究背景和意义，国内外 Nginx 负载均衡算法研究现状，以及自己研究的目的与意义。最后论述论文的大体结构。

第 2 章对所使用的相关技术进行介绍与研究，其中包括服务器集群技术，负载均衡技术的类型和常见的负载调度策略，并对 Nginx 重要模块和服务器集群中的反向代理技术做了介绍。

第 3 章对人工智能算法的的分析与探究

第 4 章搭建集群系统，利用相关工具对两种方法进行测试第 5 章分析结果

第 6 章总结，参考文献，附录，致谢

# 相关理论及技术研究

## 服务器集群

* + 1. 服务器集群定义

服务器集群简称集群（Cluster），由一组线性的服务器组成。通常将集群中的服务器称作一个节点，集群中各节点之间可以相互通信、协作，共同组成一个高性能的服务器系统[[11](#_bookmark54)]。来自 Client 的请求由 Cluster 中各服务节点共同处理，同时产生的业务数据通过数据库或者队列等存储形式在各个服务节点之间共享。从Client 的角度来看，服务器一整个集群被视作一个整体，服务器集群向客户端忽略了集群中各服务节点之间的通信细节，各服务节点对外实现逻辑同一。

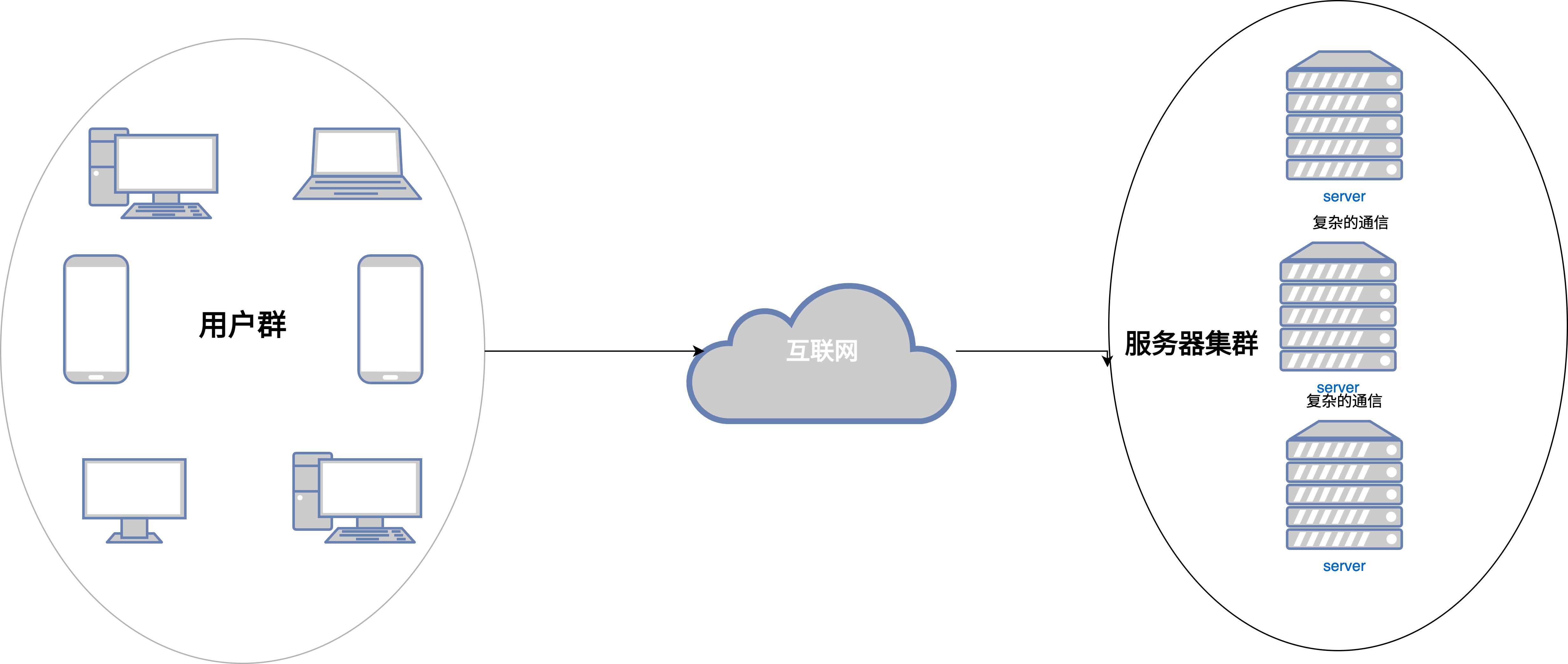


图 2.1 服务器集群与客户端

* + 1. 服务器集群的架构优势
       1. 高性能

服务器集群通过协调不同物理硬件性能的服务器，在服务器服务节点处理客户端的请求时，尤其是高并发的场景下，相比于单个服务器而言，集群架构表现了优越的性能。这种优越的性能不仅仅是不同性能设备能力的叠加，更是因为通信细节的互相协作，紧密联系造就的。

* + - 1. 高可用

服务器集群的高可用性体现在服务器集群的容灾和故障转移之上[[12](#_bookmark55)]。对于

大型开发，企业业务来说，服务器的因为处理器负载过高或者其他导致服务器出现问题而重启崩溃的问题是不可容忍的。集群能够处理这种单独服务器而引发的问题，即使集群中某个服务节点出现故障，也不能够导致集群不可用，集群中其他服务节点需要立即分摊异常节点的工作。而只有当集群中所有服务节点同时不可用时，集群才会停止服务。

* + - 1. 可伸缩性

可伸缩性实现了对服务器节点灵活管理，并且用户对此并无感知。在负载比较大的情境下，就需要增加对服务器节点的投入，相反如果负载比较小的情景下，就可以减少集群内的服务器节点

而本文主要优化的方向就是可伸缩性方面，判断不同情境下可能的负载程度，通过对负载情况的预测，能够对实现可伸缩性较大程度的优化。

* + 1. 服务器集群的分类

服务器集群的分类标准有多种，依据不同的标准可以得到不同的集群类别。例如以多业务结构划分集群，除去常见的同构集群类型外还有目前大量商用化的异构集群。若以集群实际功能结构划分的话，有三类集群是比较常见的，依次是高性能计算集群、高可用集群以及负载均衡集群[[13](#_bookmark56)]。

* + - 1. 高性能计算集群

目前高性能计算集一个分支，一直以来都在不断研究多机并行算法及相关软件的开发，致力于打造超级计算机用于进行复杂的科学运算[[14](#_bookmark57)]。通过使用并行计算技术，HPC 集群将海量运算问题分解为若干个小部分进行独立运算，最终结果将由各服务器的计算结果的汇总得到，即通过多台机器提高数据运算的能力，降低数据运算的时间成本。

伴随着当前大数据人工智能技术的发展，HPC 集群已经逐渐渗透到科学研究的各个领域，给海量数据运算工作提供了强有力的保障。高性能计算集群在地形分析、生物制药、数据挖掘以及图像处理等领域发挥着十分重要的作用

(2) 高可用集群

高可用集群顾名思义，即使在高负载、高并发的场景下，可以将该服务器中的服务、资源、IP 等转移到另外一台服务器上，从而满足业务的持续性；这两台或多台服务器构成了服务器高可用集群。简单来说就是京东淘宝 24 小时不断买买买，微信 QQ 不断发短信，保证了服务器的不间断运行。然而永远的不间断运行几乎是不可能的，常见的算法并不能彻底处理突然之间的高并发和高负载，

比如在双十一期间，某购物网站无法处理突然增加的大量订单用户刷新造成的

DDoS 行为，具体衡量标准请看下面这份表。

表 2.1: HA 衡量标准[[15](#_bookmark58)]

|  |  |  |  |
| --- | --- | --- | --- |
| 描述 | 通俗叫法 | 可用性级别 | 年度停机时间 |
| 基本可用性 | 2 个 9 | 99% | 87.6 小时 |
| 较高可用性 | 3 个 9 | 99.9% 8.8 小时 |  |
| 具有故障自动恢复能力的可用性 | 4 个 9 | 99.99% | 53 分钟 |
| 极高可用性 | 5 个 9 | 99.999% | 5 分钟 |

（3）负载均衡集群

当大量用户并发访问时，将请求转发到不同的机器止实现负载均衡。负载均衡集群是由前端的负载均衡器与后端的服务器构成负载均衡器介于客户端和服务器之间，通过负载均衡调度策略将负载分发到后端服务器处理，负载均衡集群可以分散单台服务器的访问压力和存储压力降低单台服务器宕机带来的业务影响[[16](#_bookmark59)]。为了保证客户端发送的请求能够成功的发送给后端服务器。处理负载均衡器会判断后端服务器是否正常可用，如果检查到后端服务器状态正常则根据相应的负载均衡策略从这些可用的服务器中选择一台服务器处理对应的请求，但是如果检查到后端服务器状态异常，则该服务器会被自动剔除待其恢复正常再加入集群系统中。

但是负载均衡集群存在一定的缺陷，负载均衡器无法对集群服务器性能进行监控，因而分配请求时容易导致服务器过载或者空闲情况的出现。针对这种情况，该类集群将收集业务服务器动态负载状况的性能监控程序部署在请求分发服务器上，通过对该程序的使用可以时时刻刻掌握上游服务器的负载变化。本论文的研究内容即是负载均衡集群下深度学习算法的使用。

* + 1. 服务器性能指标

负载均衡集群通过使用性能监控程序观察上游服务器的性能情况，要达到这一点，其中最主要的一点是如何选取评价节点当前负载状况的指标，且这些指标获取简单，获取时占用系统资源少，更能有效的反映出的服务器节点的资源使用情况，可以将这些指标称作为负载向量。通过研究发现，反应服务器性能

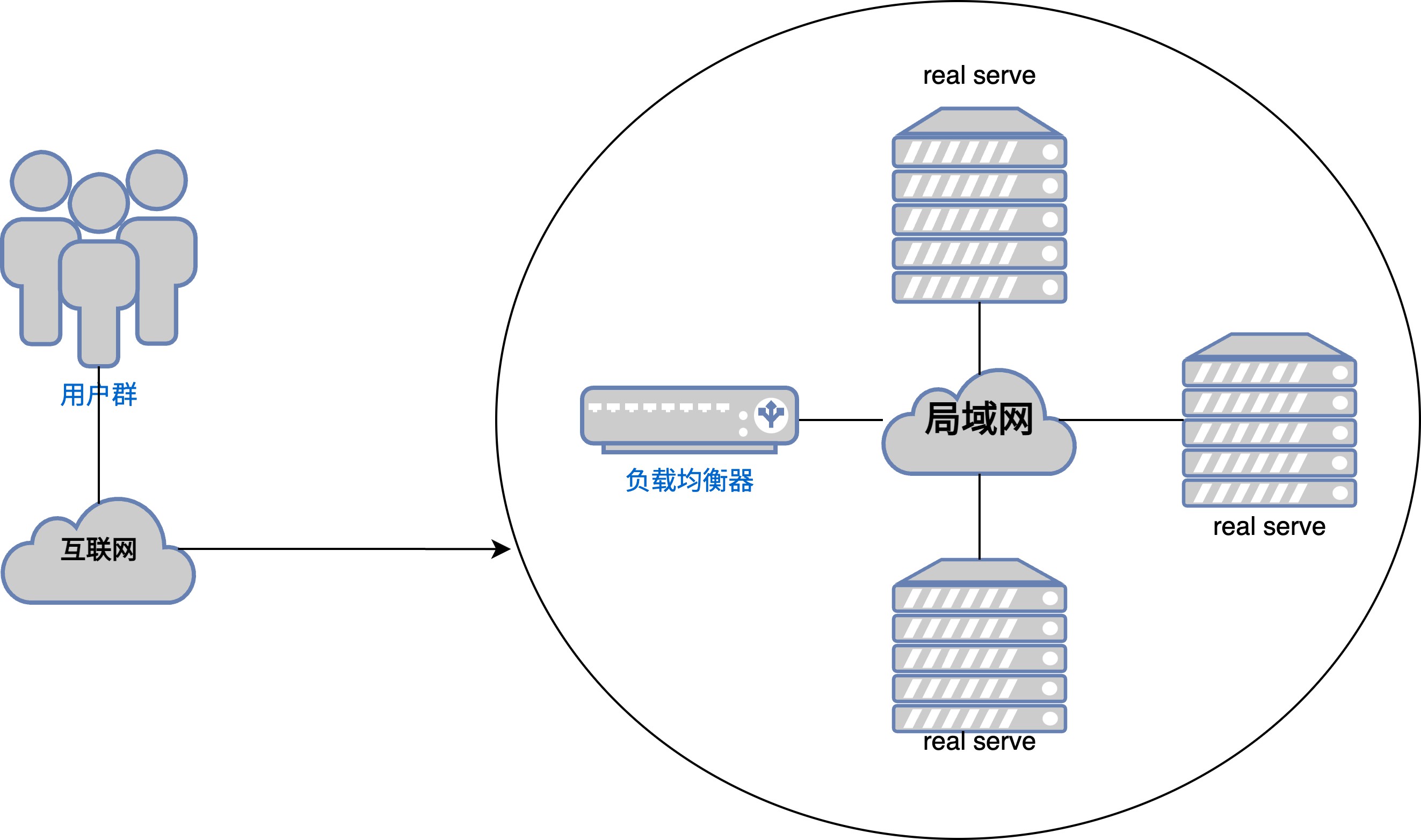


图 2.2 负载均衡集群的基本架构

指标通常从这几个方面考虑：CPU，内存，网络带宽，磁盘 I/O，通过研究发现，采用不同的性能评价指标对负载均衡算法有较大的改变，而随着应用环境的改变，评价指标也会有不同的变化。

通过监控资源利用率（如 CPU、内存、磁盘 I/O 和网络带宽），系统可以确保工作负载均匀分布在所有可用资源上。这样可以确保没有任何单个资源的过度使用，从而最大化整体性能。当业务增长时，可以根据资源利用率数据来决定何时需要扩展硬件或调整系统配置，以优化性能和成本。

计算机一个重要的硬件就是 CPU，通常 CPU 的利用效率就几乎相当于服务器节点的性能情况，如果利用率过高，那么性能就会变低，负载程度变大。每个 CPU 的大小核和频率也是影响 CPU 性能的重要组成条件，每个 CPU 的型号不同，那么需要将 CPU 性能作为负载向量的评价也就越难。服务器节点的内存访问速率对 CPU 的任务处理效率影响较大。而当网络请求量的增加带来的数据访问也会不断增加，在内存中数据请求和移动造成的分页错误或缺页的概率也会不断上升。这样会严重影响 CPU 中任务处理效率，服务器节点处理请求任务的效率下降，导致集群整体的性能下降

如果主要对服务器的计算有着高性能的需求，那么最好将 CPU 和内存作为重要的评价指标，若是处理的任务需要频繁的进行数据的传输和存储，那么网

络和磁盘 I/O 对这种任务类型的影响更大。

上述负载向量指标只是服务器节点实时的工作状态[[17](#_bookmark60)]，而队列的长度是节点处理请求任务的，完成某项任务的整个流程法人方向来思考的，这种方式具有预测性的。此时，如果有任务入队列，那么此时的服务器节点负载程度比较高，并不能接受更多的任务。性能监控程序只能监控当先服务器的实时性能，但是网络任务是作为队列被负载均衡器分发的虽然能够判断队列的长度，但是队列里对任务消耗资源的能力是无法得知的，所以任务队列的长度不能准确反应节点的负载能力。队列一般考虑的是 CPU 的队列长度获取 CPU 的队列长度，可以通过两个视角来呈现，其一是获取 CPU 一段时间内的平均队列长度，其二是获取 CPU 某个时刻的队列长度。

由于不能简单的使用 CPU 性能和任务队列长度作为负载向量指标，所以我们需要一个综合的负载信息评价标准，获取多种不同的指标信息，经过数学运算，得到一个能够从多方面体现节点负载程度的新指标。通过对黄伟华，一种特征加权模糊聚类的负载均衡算法[[18](#_bookmark61)] 的研究，主要有三种方法。第一种方法既是使用队列长度作为主要指标，同时考虑使用 CPU 和磁盘 IO 作为次要指标；另一种是完全结合，同时将队列的长度和资源利用率作为参考指标；最后一种是通过优先级作为主要指标，比如内存占用情况，在内存充足的情况下，在考虑其他次要的指标。

* + 1. 本文选择的参考指标

依据自身经济和技术情况，选择使用观察节点的各种资源利用情况作为负载均衡能力的指标，并选取了 CPU，内存，磁盘 I/O 和网络带宽作为评价各个节点性能的指标。具体的系统状态评价对应的资源利用率情况本文进行了整理，如下图表所示。

表 2.2: 稳定系统的资源状态和评价

|  |  |  |
| --- | --- | --- |
| 性能指标 | 资源利用率 | 状态评价 |
| CPU 利用率 | 70% | 好 |
| 85% | 坏 |
| 90%+ | 很差 |

Continued on next page

表 2.2: 稳定系统的资源状态和评价 (Continued)

|  |  |  |
| --- | --- | --- |
| 磁盘 IO | <30% | 好 |
| <40% | 坏 |
| <50% | 很差 |
| 运行队列 | <2\*CPU 数量 | 好 |
| 网络带宽 | <30% 带宽 | 好 |
| 内存 | 没有页交换 | 好 |
| 每个 CPU 每秒 10 个页  交换 | 坏 |
| 更多的页交换 | 很差 |

设 𝐶𝑗、𝑀𝑗、𝐷𝑗、𝑀𝑗 分别作为第 j 个节点的 CPU 剩余性能，内存剩余性能，磁盘剩余性能，网络带宽剩余性能。在服务器集群处理大量任务和网络请求时，由于分发的请求量和请求所处理的数据量的不同，可能会出现有的节点处于高负载，有的节点处于低负载的情况，这样服务器集群的性能就不能得到充分的发挥，所以需要一直监控集群中各个节点的负载状况，不断调整任务队列的分配方案。下面是计算当前服务器节点的负载状态的数学公式

𝑈𝑗 = 𝑊𝑐𝑝𝑢 ⋅ 𝐶𝑗 + 𝑊𝑚𝑒𝑚 ⋅ 𝑀𝑗 + 𝑊𝑖𝑜 ⋅ 𝐷𝑗 + 𝑊𝑛𝑒𝑡 ⋅ 𝑁𝑗 (1.1)

𝑈𝑗 为当前服务器节点的总体剩余性能。通过对 𝑈𝑗 值的监控，观察其变化范围作为判断是否需要修改请求人文分配方案的依据。

## 负载均衡技术

* + 1. 什么是负载均衡

负载均衡是在支持应用程序的资源池中平均分配网络流量的一种方法。现代应用程序必须同时处理数百万用户，并以快速、可靠的方式将正确的文本、视频、图像和其他数据返回给每个用户。为了处理如此高的流量，大多数应用程序都有许多资源服务器，它们之间包含很多重复数据。负载均衡器是位于用户与服务器组之间的设备，充当不可见的协调者，确保均等使用所有资源服务器。

* + 1. 负载均衡的优势

负载均衡可以定向和控制应用程序服务器与其访客或客户端之间的互联网流量。因此，它可提高应用程序的可用性、可扩展性、安全性和性能。

* + - 1. 应用程序可用性

服务器故障或维护可能会增加应用程序停机时间，使访客无法访问应用程序。负载均衡器可以通过以下方式提高系统的容错能力，自动检测服务器问题并将客户端流量重定向到可用服务器。运行应用程序服务器维护或升级而无需使应用程序停机，为备份站点提供自动灾难恢复，执行运行状况检查并防止出现可能导致停机的问题。

* + - 1. 应用程序可扩展性

可以使用负载均衡器在多个服务器之间智能地定向网络流量。应用程序可以处理数千个客户端请求，防止任何一台服务器出现流量瓶颈，预测应用程序流量，以便可以在需要时添加或移除不同服务器，为系统增加冗余度，使我们可以放心扩展。

* + - 1. 应用程序安全

负载均衡器具有多项内置的安全功能，它们是应对分布式拒绝服务攻击的有用工具，在这种攻击中，攻击者会用数百万个并发请求淹没应用程序服务器，从而导致服务器故障。负载均衡能够做到：监控流量并阻止恶意内容，预测应用程序流量，将攻击流量自动重定向到多个后端服务器，以最大限度减少影响，通过一组网络防火墙路由流量，以提高安全性。

* + - 1. 应用程序性能

负载均衡器通过增加响应时间和减少网络延迟来提高应用程序性能。它们可以执行诸如以下几项关键任务：在服务器之间平均分配负载以提高应用程序性能，将客户端请求重定向到地理位置较近的服务器以减少延迟；确保物理和虚拟计算资源的可靠性和性能。

* + 1. 负载均衡的类型

实现负载均衡的方式有很多种，最常见的时间负载均衡类型有三种。

* + - 1. 软件和硬件负载均衡

基于硬件的负载均衡器是一种硬件设备，可以安全地处理千兆字节的流量并将其重定向到数百个不同的服务器。可以将其存储在数据中心，并使用虚拟化创建多个可以集中管理的数字或虚拟负载均衡器。基于软件的负载均衡器是

执行所有负载均衡功能的应用程序。可以将它们安装在任何服务器上，也可作为完全托管的第三方服务的形式访问。

硬件负载均衡器需要初始投资、配置和持续维护。也可能不会满负荷使用它们，尤其只是为了处理高峰时段的流量高峰。如果流量突然增加到超出其当前容量，这将影响用户，直到能购买并设置另一个负载均衡器为止。相比之下，基于软件的负载均衡器要灵活得多[[19](#_bookmark62)]。它们可以轻松地纵向扩展或缩减，并且与现代云计算环境更加兼容。随着时间推移，它们的设置、管理和使用成本也会降低。

* + - 1. 静态和动态负载均衡

静态负载均衡是根据相关规则制定的负载均衡算法，典型的静态负载均衡策略有加权轮询、IP-HASH 等。静态负载均衡又称作确定性调度，这是由于在请求分发过程中该类策略不会考虑服务器实际运行时的负载性能，这就容易导致出现性能好的服务器处于过载而性能一般的服务器处于轻载甚至空闲状态的不均衡状况，集群资源利用率得不到充分发挥，系统性能被大大削减。静态负载均衡容易实现，配置简单，因此也是比较常见且应用较多的一种负载均衡实现方式。

动态负载均衡的实现需要更为复杂的技术支撑，它在静态负载均衡的实现基础上，需要对集群系统各服务器的运行状态加以评估，通过某些反映服务器运行时负载情况的负载评价指标的变化来动态调节各服务节点的请求分发比例，从而能够将集群系统的资源充分利用在处理高并发客户端请求上，以达到集群负载动态均衡的目标。在实践中，动态负载均衡能够大大提升集群系统的负载性能，但其实现是建立在一定量的系统资源消耗之上，对于一些没有资源的企业来说可能有点困难，但是对于大型企业来说这些成本就可以忽略不计。

根据负载均衡技术类型的不同，下表分析了常见的 Nginx 负载均衡策略。表 2.3: 常见 Nginx 负载均衡算法分析表

|  |  |  |  |
| --- | --- | --- | --- |
| 负载均衡算法分  类 | 算法名 | 优点 | 缺点 |
|  | 轮询算法 | 配置简单 | 不能考虑后端服  务器性能差异 |

静态负载均衡

Continued on next page

表 2.3: 常见 Nginx 负载均衡算法分析表 (Continued)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 加权轮询算法 | 能够考虑不同服务器的性能差异 | 不能考虑负载后服务器的状态变  化 |
| IP 哈希法 | 可以解决 session  问题 | 但在 Nginx 非最  前服务器后失效 |
| 动态负载均衡 | 平均分配认为 | 不能考虑服务器  处理能力不同 |  |
| 加权最小连接法 | 考虑处理能力的  不同 | 无法衡量服务器  负载状态 |
| 最短响应时间法 | 根据服务器响应  时间分配请求 | 通信开销过大 |
| 基于资源的方法 | 性能最大化，控  制分散灵活 | 配置复杂，不一  定反映实际负载 |

* + - 1. 四层到七层负载均衡

大型的网站服务一般会用这样的架构，在业务应用前面加七层负载均衡，然后在七层负载均衡前面加四层负载均衡。当用户发送 HTTP 请求的时候，请求会（经过机房内的路由器，交换机等设备）首先到达四层负载均衡，四层转发给七层，七层转发给应用。四层负载均衡只解析网络包到第四层，根据四层的内容

（比如 TCP port，IP 等）就能确定转发给谁，七层负载均衡解析网络包到第七层，要根据七层的内容（比如 HTTP URL path，HTTP header 等）才能确定转发给谁。

七层即应用层[[20](#_bookmark63)]，要解析 HTTP 内容（不仅仅是 HTTP，其他应用层协议的负载均衡，比如一些数据库 proxy，gRPC 代理，等等，都需要解析完成应用层才能确定转发目标），首先要将 Header 全部读完，读完之后要看下 Content Length是有多长，然后知道 Body 要读到哪里。根据不同的 URL Path，还要确定路由到哪一个 upstream。

四层即传输层，对于用户任务，四层负载均衡器首先通过预先配置好的负载均衡策略在内部网络中挑选一台最好的服务器，接着将请求报文中的目标 IP地址和端口号修改为内部最佳的服务器 IP 和端口号并建立 TCP 链接转发请求，

最后将响应结果返回给用户。用 Nginx 模拟四层负载均衡如下图[2.3](#_bookmark8)所示。

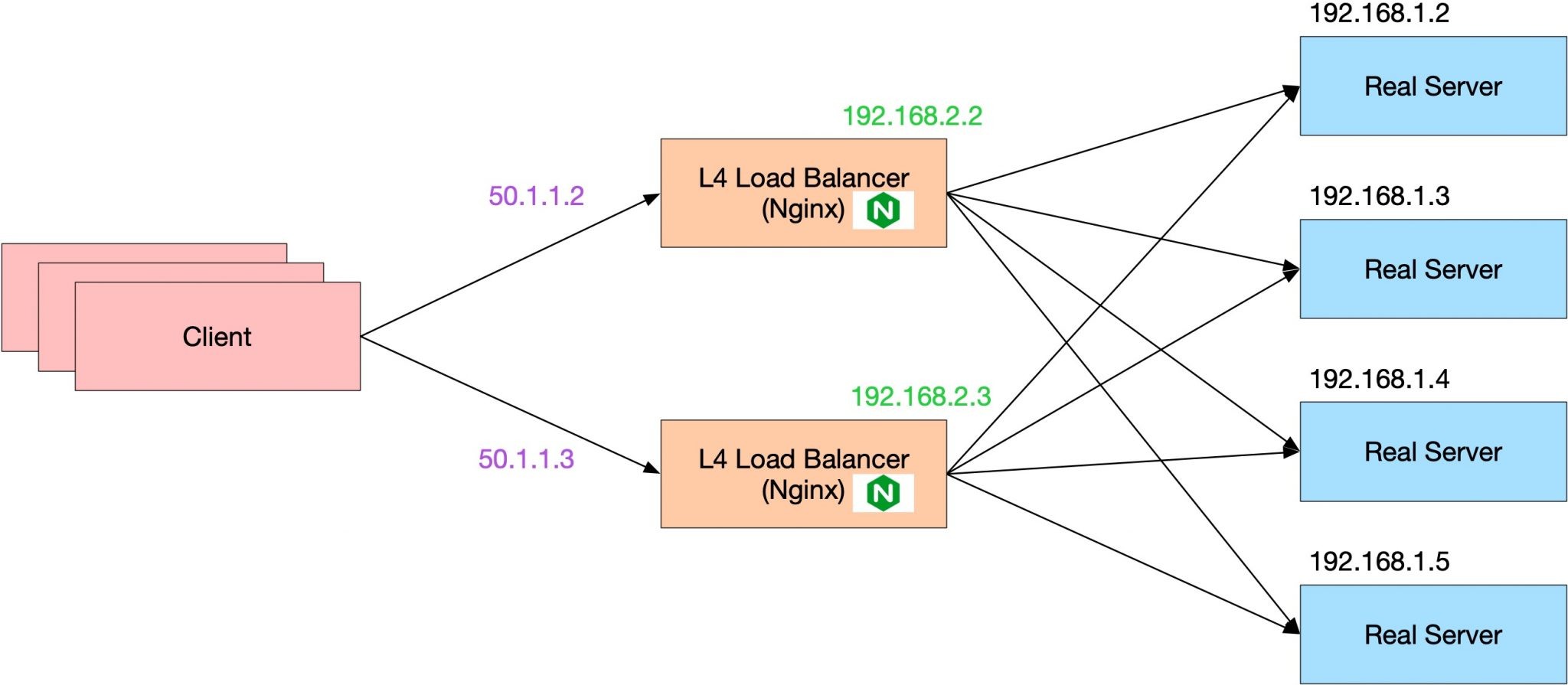


图 2.3 Nginx 作为四层负载均衡使用

## Nginx 服务器

Nginx 是一款轻量级的 Web 服务器/反向代理服务器及电子邮件代理服务器。本身具有占用内存少，并发能力强等特点，其并发能力在同类型的网页服务器中表现较好。Nginx 是由伊戈尔·赛索耶夫为俄罗斯访问量第二的 Rambler.ru站点开发的，第一个公开版本 0.1.0 发布于 2004 年 10 月 4 日。Nginx 相较于之前盛行的 LAMP（Linux Apache MySQL PHP/Python/Perl）由于 Apache 服务器同步多进程的处理方式，而 Nginx 是基于事件驱动架构，在处理任务请求是异步而非阻塞的，因此应对高并发请求是以就能够保持资源的低消耗、响应能力迅速以及稳定性强的特点[[21](#_bookmark64)]。包括百度，京东等众多服务器都是采用 Nginx。

* + 1. Nginx 工作模式和进程模型

Nginx 有单进程和多进程两种工作模式，默认进程为多进程。通常情况下，单进程仅仅只在开发环境下调试使用，对外发布服务时常常使用多进程。多进程模型既是 master-worker 进程模型

1. Nginx 启动后，会产生一个 master 主进程，主进程执行一系列的工作后会产生一个或者多个工作进程 worker
2. 在客户端请求动态站点的过程中，Nginx 服务器还涉及和后端服务器的通信。Nginx 将接收到的Web 请求通过代理转发到后端服务器，由后端服务器进行数据处理和组织
3. Nginx 为了提高对请求的响应效率，降低网络压力，采用了缓存机制，将历史应答数据缓存到本地。保障对缓存文件的快速访问

首先，worker 进程之间是平等的，每个 worker 进程都是从 master 进程 fork过来，在 master 进程里面，先建立好需要listen 的socket（listenfd）之后，然后再fork 出多个 worker 进程。每个 worker 进程，处理请求的机会也是一样的。所有 worker 进程的listenfd 会在新连接到来时变得可读，为保证只有一个进程处理该连接，所有 worker 进程在注册listenfd 读事件前抢accept\_mutex，抢到互斥锁的那个 worker 进程注册listenfd 读事件，在读事件里调用 accept 接受该连接。当一个 worker 进程在 accept 这个连接之后，就开始读取请求，解析请求，处理请求，产生数据后，再返回给客户端，最后断开连接，这样就是一个完整的请求就是这样的了。

具体图示如下

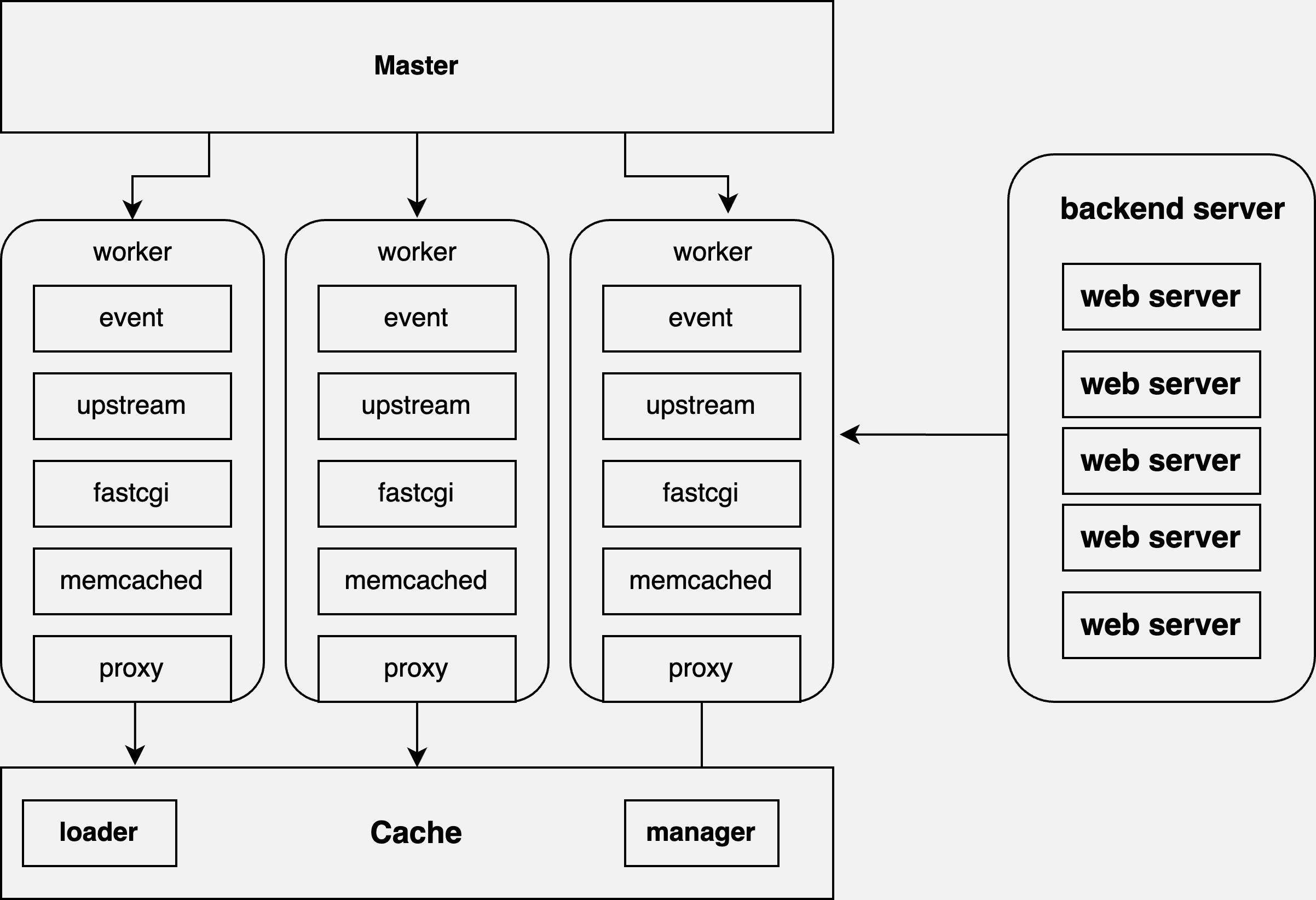


图 2.4 Nginx 多进程工作模式

由于 Nginx 底层基于 epoil 事件驱动模型实现异步非阻塞 IO，一个进程可以监听多个 socket，因此每一个 worker 进程都可以并发多个链接甚至上万个连接[[22](#_bookmark65)]，这保证了 Nginx 的高并发性能，而且消耗的内存非常少。

* + 1. Nginx 的反向代理

代理简单来说，就是如果我们想做什么，但又不想直接去做，那么这时候就找另外一个人帮我们去做。那么这个例子里面的中介公司就是给我们做代理服务的，我们委托中介公司帮我们找房子。弄清楚代理是什么了，既然有反向代理，那么正向代理是什么？先从反向代理解释起。反向代理，其实客户端是没有任何感知的，因为客户端不需要任何配置既可以访问。我们只需要将请求发送到反向代理服务器，由反向代理服务器去选择目标服务器获取数据后，在返回给客户端，此时反向代理服务器和目标服务器对外就是一个服务器，暴露的是代理服务器地址，隐藏了真实服务器 IP 地址。

理解这两种代理的关键在于代理服务器所代理的对象是什么，正向代理代理的是客户端，我们需要在客户端进行一些代理的设置。而反向代理代理的是服务器，作为客户端的我们是无法感知到服务器的真实存在的。总结起来就一句话：正向代理代理客户端，反向代理代理服务器[[23](#_bookmark66)]。

用一个图片对比正向和反向代理

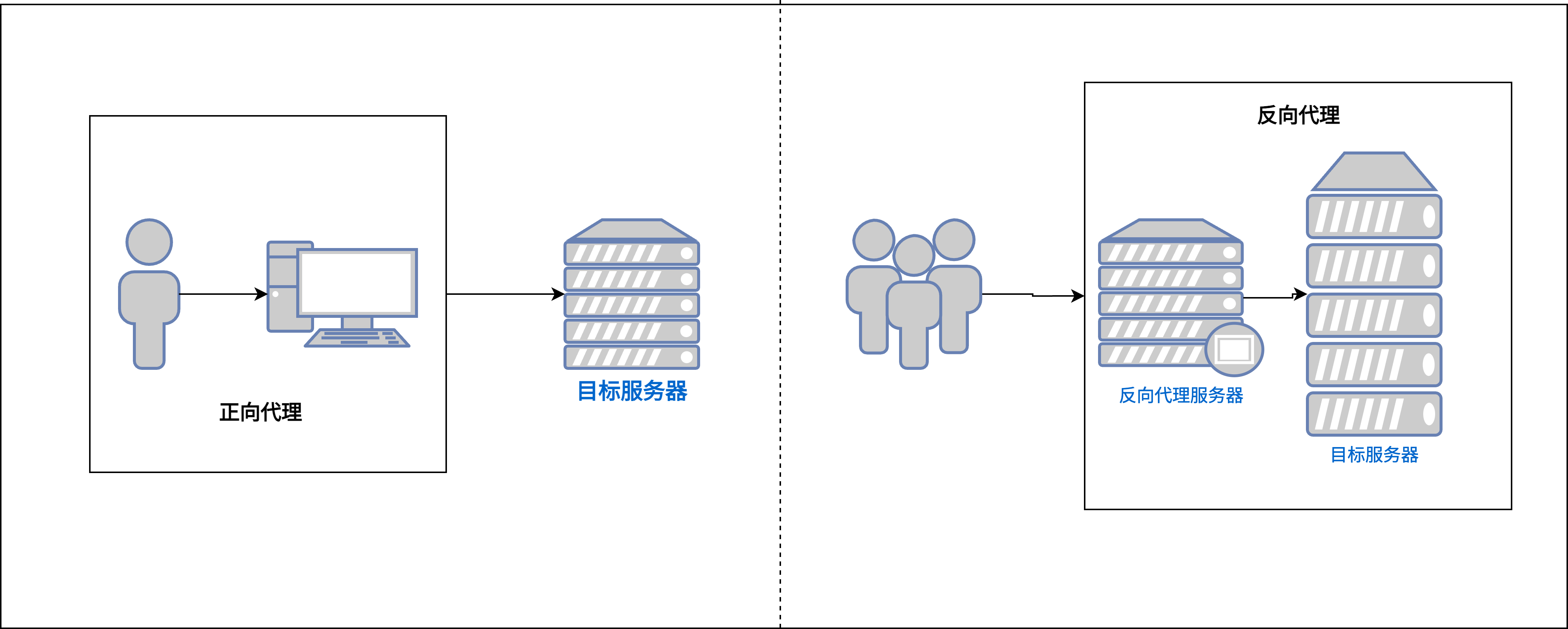


图 2.5 正向代理与反向代理的对比

Nginx 反向代理功能使其能够逾越单机的局限性，并拥有在网络上接收、转发以及处理数据的能力[[24](#_bookmark67)]。通过 proxy\_pass 指令可以配置 Nginx 反向代理， proxy\_pass 的语法结构为 proxy\_passURL，这个 URL 即表示 Nginx 所代理的 上游服务器地址信息。Nginx 反向代理的一个配置实例如下所示，配置中的 “proxy\_set\_headerX-Real-IP”表示定义了一个值为“$remote\_addr”的首部“X- Real-IP”，也即客户端的 IP 地址。当上游服务器接收到客户端请求时，该自定义的首部将以客户端 IP 的形式被打印到服务器访问日志中，上游服务器便能够依

据真实客户端的 IP 地址而非 Nginx 的 IP 地址做日志统计分析[[25](#_bookmark68)]。

1

location \index{

set $upstream\_url "";

proxy\_set\_header X-Real-IP $remote\_addr; rewrite\_by\_lua\_file /home/${HOME}/luafile/log.lua proxy\_pass http://upstream\_url;

}

2

3

4

5

6

Listing 2.1 Nginx 反向代理默认配置

在客户端与Nginx 通信过程中，当请求到达 Nginx 服务器时，Nginx 并不能立即与目标服务器建立 TCP 连接而转发请求工作，而是将任务请求储存在内部缓存队列中，之后在与目标服务器通信发送请求，这种工作模式大大降低里服务器某一节点的负载压力。另外，由于客户端与 Nginx 是在公网上进行网络通信传输，属于慢速连接，而 Nginx 与上游服务器一般是在内部网络进行通信，属于高速连接，考虑到 HTTP 连接具有无状态性，客户端与 Nginx 可以开启Keep-Alive功能，Nginx 与上游服务器之间的Keep-Alive 则可以关闭，因此更能将占用上游服务器的系统资源释放掉，进一步减轻服务器负载。

当上游服务器针对客户端 HTTP 请求处理结束并生成响应报文发送给 Ng- inx 反向代理服务器后，Nginx 首先将拆开报文进行相关处理，二次封装完成响应给客户端。在这个过程中，Nginx 是在接收上游服务器传来的响应报文的同时向客户端发送响应报文，而不像客户端请求资源时 Nginx 先接收，接收完成后再发送给上游服务器的情况一样，Nginx 边接收边转发的工作模式可以极大地减小客户端响应的延迟[[26](#_bookmark69)]。

* + 1. Nginx 负载均衡

Nginx 自身支持多种负载均衡算法，除了使用源码自带的内置调度算法之外，还可以支持第三方扩展的负载均衡技术[[27](#_bookmark70)]。如果想要开启源码自带的调度算法可以则可以把指定调度算法的模块打开。一般来说，不需要单独下载某一模块，需要使用时在配置文件中指定即可。Nginx 内置的调度算法有加权轮询算法、最小连接算法、IP 哈希算法。第三方扩展算法则需要安装第三方模块，将模块放在指定的扩展算法模块路径，在编译 Nginx 过程中，扩展算法会一并编译在二进制程序中，最后使用便可以依据配置指令使用指定的第三方调度算法。常见的第三方调度算法有 fair 响应时间比算法（需要安装 ngx\_http\_upstream\_fair\_module）， URL 哈希算法（需要安装 ngx\_http\_upstream\_hash\_module）等。下面将对常见的

Nginx 调度算法进行分析。

* + - 1. 加权轮询算法

默认轮询算法（Round Robin）的策略是：将请求“依次”分发到候选机器。如下图所示，轮询负载均衡器收到来自客户端的 6 个请求，编号为 1、4 的请求会被发送到服务端 0；编号为 2、5 的请求会被发送到服务端 1；编号为 3、6 的请求会被发送到服务端 2。

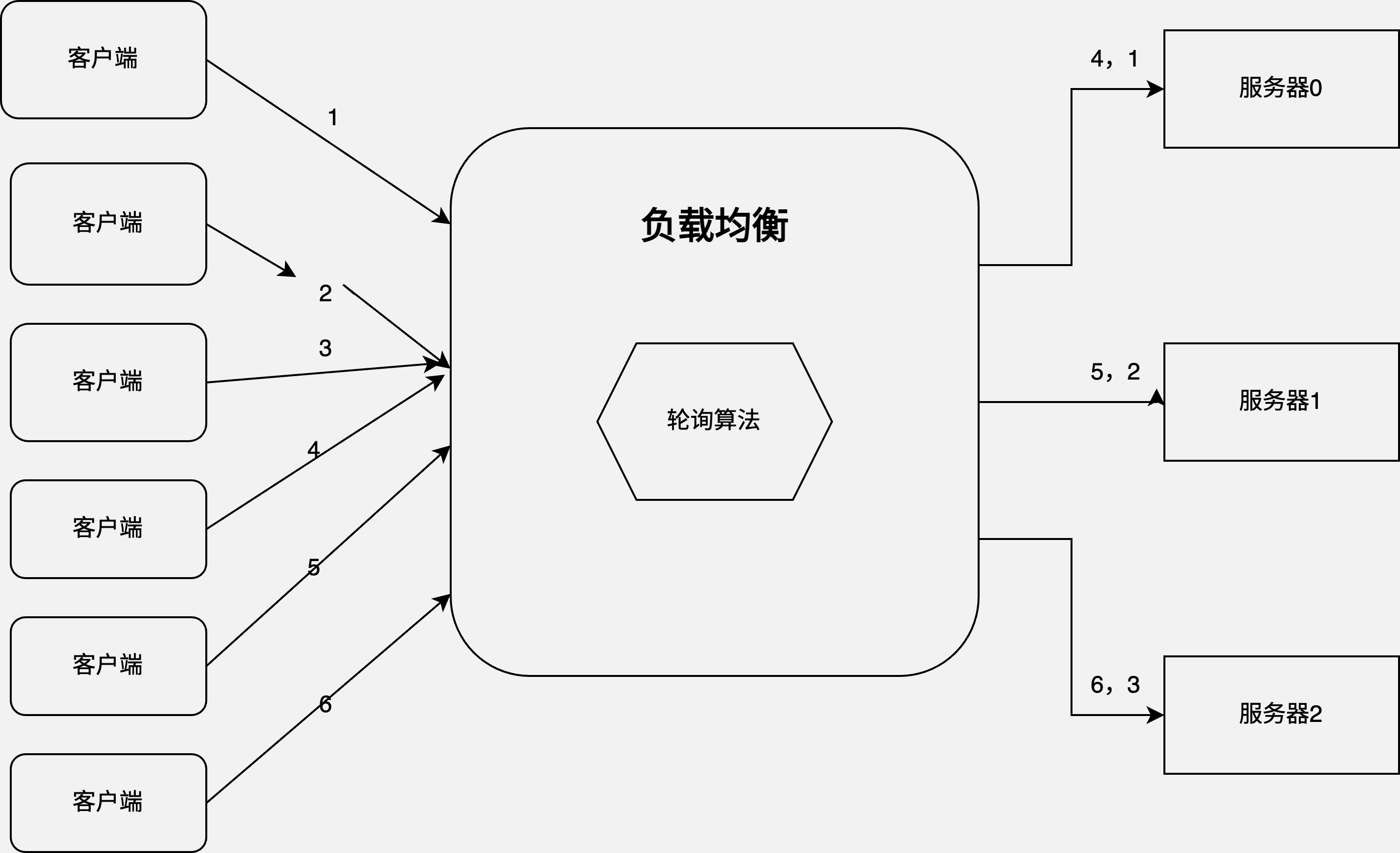


图 2.6 默认情况下的负载均衡算法

默认的轮询算法是等值轮询，即按照一比一的比例向不同的服务器节点分发请求，当配置文件没有任何 weight 权值参数是，则采用默认的轮询算法。

1



默认的轮询算法

upstream backend {

server 127.0.0.14：80 max\_fails=2 fail\_timeout=10s; server 127.0.0.15：80 max\_fails=2 fail\_timeout=10s;

}

2

3

4

5

轮询算法，算法的复杂度比较低，执行效率较高，但是不能体现出集群节点内不同的性能差异和负载情况。

* + - 1. 最小连接算法

最小连接算法（least\_conn）一句话概括就是：按 nginx 反向代理与后端服

务器之间的连接数，连接数最少的优先分配[[28](#_bookmark71)]。要根据机器连接数分发，显然要先维护机器的连接数。因此，最少连接数算法需要实时追踪每个候选机器的活跃连接数；然后，动态选出连接数最少的机器，优先分发请求。最少连接数算法会记录当前时刻，每个候选节点正在处理的连接数，然后选择连接数最小的节点。该策略能够动态、实时地反应机器的当前状况，较为合理地将负责分配均匀，适用于对当前系统负载较为敏感的场景。

1



最小连接算法

upstream backend { least\_conn;

server 127.0.0.14:80 max\_fails=2 fail\_timeout=10s; server 127.0.0.15:80 max\_fails=2 fail\_timeout=10s;

}

2

3

4

5

6

最小连接数调度算法每一个上游集群节点分别维护一个计数变量 number，每次负载均衡器分配个某一节点一个任务时，就将其对应的 number 值上加上 1，当任务完成之后，再将 number 值减去 1。Nginx 根据集群各服务器计数变量的变化，每次都会挑选 number 数值最小的服务器进行任务分配。同时最小连接算法与加权连接算法接轨，如果初始为每一个性能各异的服务器分配初始权重，那么当分发任务请求时将选择连接数与权值比例最小的服务器节点作为最佳服务器节点。最小连接算法流程图如图[2.7](#_bookmark10)所示。

最小连接算法根据服务器连接数大小变化的统计结果来动态的选择服务器分发任务，对于服务器的实时负载情况做了些许的考虑，一定程度上解决了加权轮询算法中请求到来时间间隔不一致所带来的负载差异。但由于每条连接上的请求不同，该算法但从连接数大小来反应实时负载情况还是不够全面。

* + - 1. IP 哈希算法

哈希算法（Hash）根据一个key （可以是唯一 ID、IP、URL 等），通过哈希函数计算得到一个数值，用该数值在候选机器列表的进行取模运算，得到的结果便是选中的机器[[29](#_bookmark72)]。哈希算法解决的问题既是人工的判断某个具体任务需要的性能大小，如果该任务消耗的性能较多，负载均衡器将会对该请求连接进行分配到候选的高性能服务器中。这种算法可以保证，同一关键字（IP 或URL 等）的请求，始终会被转发到同一台机器上。哈希负载均衡算法常被用于实现会话粘滞（Sticky Session）。但是，哈希算法的问题是：当增减节点时，由于哈希取模函数的基数发生变化，会影响大部分的映射关系，从而导致之前的数据不可

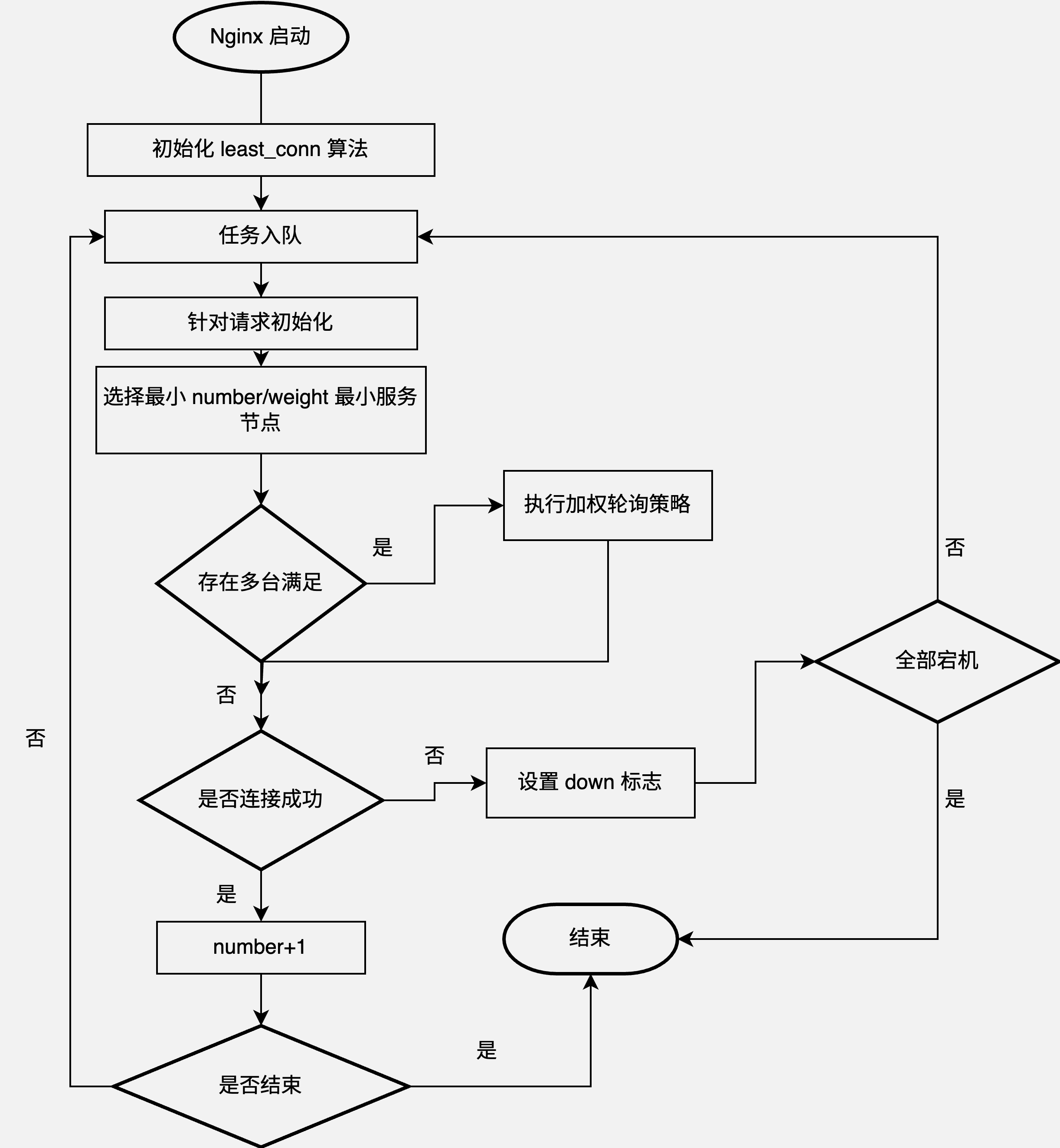


图 2.7 Nginx 最小连接算法

访问。要解决这个问题，就必须根据新的计算公式迁移数据。显然，如果数据量很大的情况下，迁移成本很高；并且，在迁移过程中，要保证业务平滑过渡，需要使用数据双写等较为复杂的技术手段。

Nginx 的 IP 哈希算法则是将来源客户端的 IP 地址作为 key，采用特定的哈希函数作哈希运算，最后将结果请求转发到特定的服务器上进行处理。Nginx 负载均衡器受到客户端请求后，将该客户端的点分十进制前三段作为参数输入到哈希函数计算，哈希值的以保证 IP 地址前三段能够分发导通一个上游服务器，

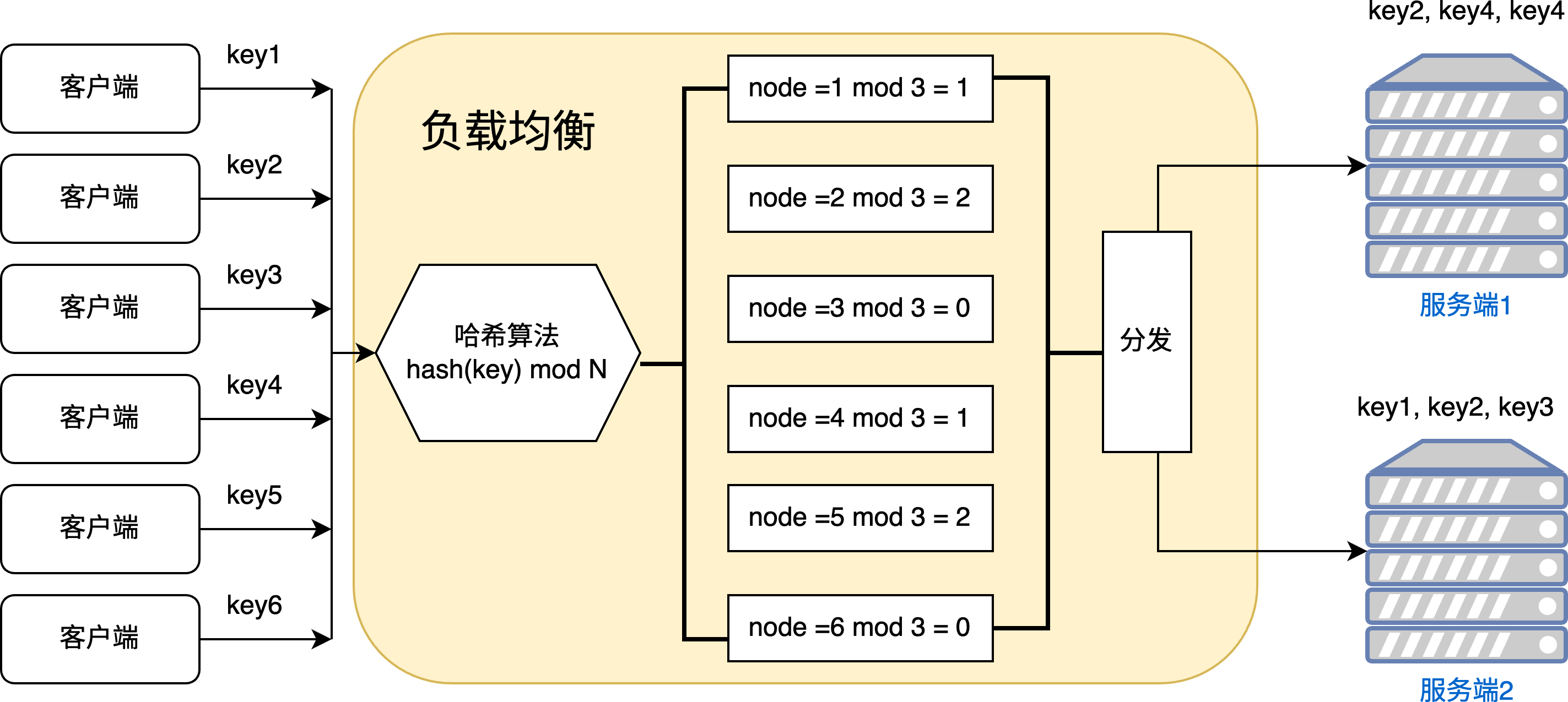


图 2.8 Nginx 哈希算法拓扑图

哈希计算完成之后将此 hash 值对集群中所有正常运行的服务器总数进行趋于，用来确定将任务请求分发到那台服务器上。在确定服务器节点过程中，如果失败次数超过 20 次，将会替换为加权轮询策略。Nginx 配置 IP 哈希算法如下所示。

1



IP-HASH 算法

upstream backend { ip\_hash;

server 127.0.0.14:80 max\_fails=2 fail\_timeout=10s; server 127.0.0.15:80 max\_fails=2 fail\_timeout=10s;

}

2

3

4

5

6

IP 哈希算法流程图如下所示。

IP 哈希算法能满足用户会话保持的要求，因为同一个 IP 的用户请求将映射导通一个服务器节点进行处理。当集群节点出现故障或者集群加入新节点，需要重新进行哈希运算，因此仍旧会出现会话沾性失效问题。在资源占用率比较高的高并发情景下，IP 哈希算法对于不同资源的消耗仍旧无法考虑，还是容易导致某一个服务器节点过载而崩溃而其他节点过于空闲的状态。所以任何不能彻底的明白每一个集群节点中的实时负载的负载均衡算法都无法保证节点是否过载或者过于空闲。

在上述的 Nginx 几种内置的常见调度算法，常见的扩展调度算法还有最小响应比算法。fair 算法在分配客户端请求时依据上游服务器的响应时间作为参考量，将处理请求的高优先级赋予响应时间短的服务器，由于响应时间越长的服务器负载相对越重，因此优先级滞后。fair 算法依据页面大小和加载时间长短能

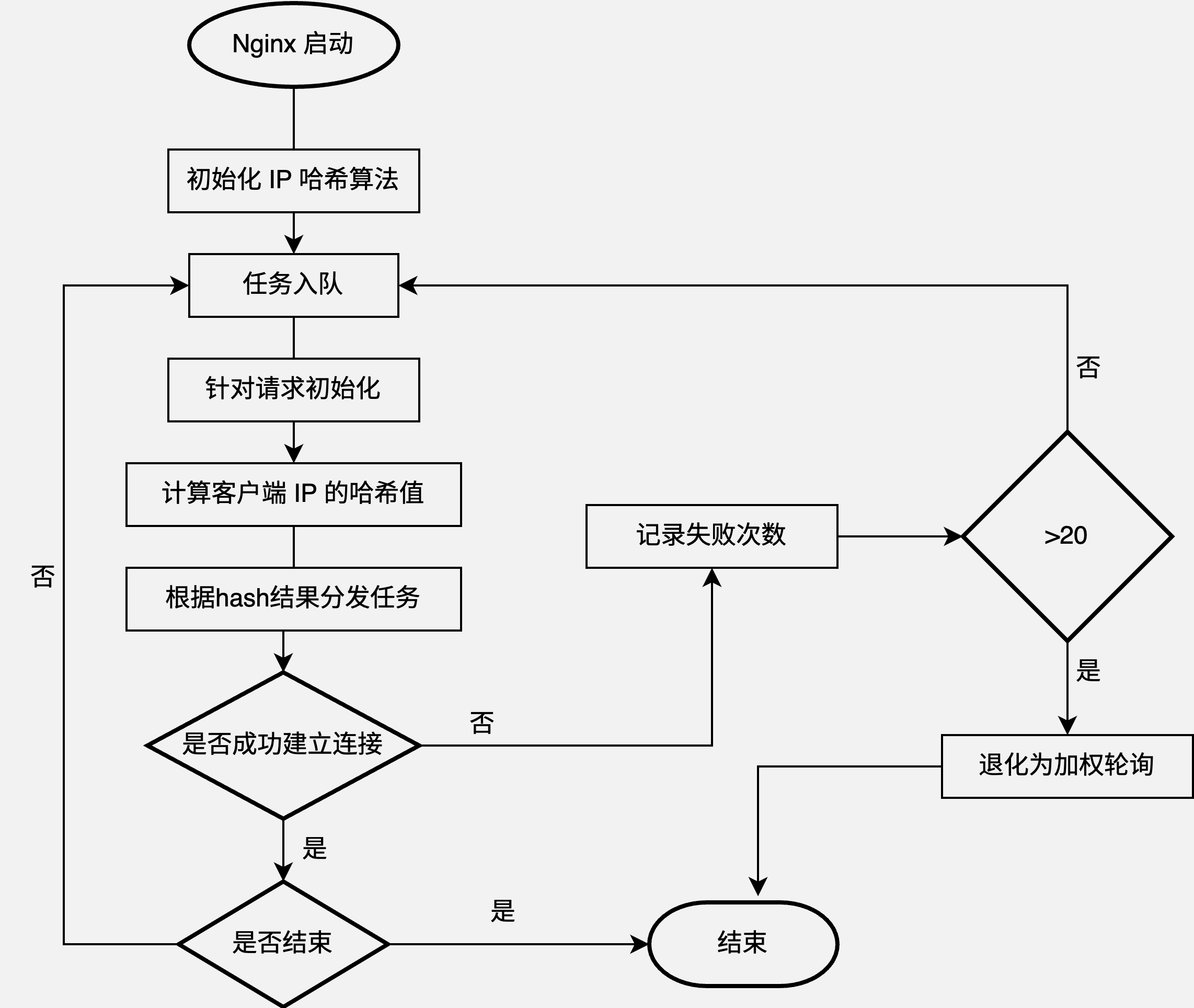


图 2.9 Nginx IP 哈希算法流程图

够自适应地进行负载均衡，在一定程度上考虑到了上游服务器的性能差异。但在高并发情况下，没有考虑到网络拥塞可能带来的耗时延长的情况，且单靠响应时间来判断负载过于片面，不够准确可靠[[30](#_bookmark73)]。

经过上面的对 Nginx 几种内置的负载均衡算法可以得知，静态负载均衡算法配置一遍比较简单，而且执行效率很好，一定程度上能够反应集群节点的负载性能。而动态负载均衡则配置比较复杂，在不考虑多方面的情况下，不够准确可靠，但是相比于静态负载均衡能够更加实时的了解某一节点的性能状态，以至于能够让负载均衡器更好的分配工作。于是本文将从机器学习和神经网络方面探究动态负载均衡中关于网络方面的优化，并分析结果加以与原来对比。

## 本章小结

本章主要阐述了常规负载均衡算法探究涉及的相关理论和技术。首先介绍了服务器集群的概念和分类，同时确定了如何测量服务器剩余节点性能的标准。其次分析了负载均衡技术的意义与优势，在分析负载均衡类型的过程中确认了要突破的方向。最后通过对于 Nginx 源码的了解探究了 Nginx 的工作模式和进程模型，了解了常规的负载均衡算法以及实现，明确了动态负载均衡算法的改进研究方向。

# 网站访问量时间序列预测研究

Nginx 作为一款高性能的 HTTP 和反向代理服务器，其本身和时间序列不是直接相关。但是，在监控 Nginx 性能的过程中，会涉及到时间序列数据。比如， Nginx 的访问日志和错误日志都会记录下请求的时间信息，这些日志数据就构成了时间序列数据。可以通过分析这些时间序列数据来理解 Nginx 的流量模式、性能趋势以及潜在的问题。而且，对于时间序列的处理和分析还可以应用在 Nginx及其它服务的自动化监控和告警系统中。这样的系统可以帮助管理者更有效地了解服务状态，并在出现问题时迅速作出反应。

## 时间序列预测基本方法

时间序列是一组按照时间顺序排列的数据点。这种数据点是通常按照固定的时间间隔收集，比如每天，每周，每月，每年的观测值。间序列数据用在许多领域，包括经济学、气象学、社会学、工程和自然科学等。例如，股票市场价格、气温记录或公司收入都可以被视作时间序列数据。通过对时间序列数据的分析，可以洞察到数据背后的趋势、季节性变化、循环等特征，也可以用来做出未来的预测。在 Nginx 网络访问量的时间序列数据点则可以将每一天某个网页的访问量记录下来，按照日期的对应关系，每一个 datetime 对应一个 record。

时间序列预测方法有很多，简单的方法不多提，主要研究了指数平滑法、趋势延伸法、自回归滑动平均模型（ARMA）、差分整合移动平均自回归模型

（ARIMA）。

1. 指数平滑法。指数平滑法则在历史时间序列的数据给予权重之后的和作出了一些优化改进。指数平滑法根据平滑的次数来进行分类，有一次指数平滑法、二次指数平滑法和三次指数平滑法等。但它们的基本思想与移动平均法类似，都是历史时间序列的数据给予权重之后的和，但是它给不同的信息数据的权重大小不同，与目标预测值时间距离越近，权重越大。趋势延伸法主要趋势延伸法主要是将历史时间序列数据在坐标图上标记处所在的位置，然后画出一条能最好反映历史数据变化的直线或者曲线，并进行延伸得到预测目标。
2. 自回归滑动平均模型[[31](#_bookmark74)]（ARMA）结合了自回归模型（AR）和滑动平均模型（MA），ARMA 模型能有效地捕捉时间序列数据的自相关结构。AR（p）

模型，它预测当前值作为前p 个观测值（lags）的线性组合，模型中的 p 代表模型的阶数，即延迟的观测值的数量。AR 部分的如下所示

𝑋𝑡 = 𝜙1𝑋𝑡−1 + 𝜙2𝑋𝑡−2 + ⋯ + 𝜙𝑝𝑋𝑡−𝑝 + 𝜖𝑡 (3.1)

其中，𝑋 − 𝑡 是时间序列在时刻t 的值，𝜙1, 𝜙2, … , 𝜙𝑝 是模型参数，𝜖𝑡 是白噪声误差项。MA（q）模型，即移动平均模型，它使用当前和过去的预测误差（即观测值与模型预测值之间的差）来预测当前值。模型中的 q 代表误差项的阶数。

𝑋𝑡 = 𝜖𝑡 + 𝜃1𝜖𝑡−1 + 𝜃2𝜖𝑡−2 + ⋯ + 𝜃𝑞𝜖𝑡−1 (3.2)

其中，𝜖𝑡 是模型的白噪声误差项，𝜃1, 𝜃2, … , 𝜃𝑞 是模型参数。把 AR 部分和 MA

部分结合起来就得到了 ARMA 模型。

𝑋𝑡 = 𝜙1𝑋𝑡−1 + ⋯ + 𝜙𝑝𝑋𝑡−𝑝 + 𝜖𝑡 + 𝜃1𝜖𝑡−1 + ⋯ + 𝜃𝑞𝜖𝑡−𝑞 (3.3)

ARMA 模型的参数通常通过极大似然估计（Maximum Likelihood Estimation, MLE）或者最小二乘法来确定。在使用 ARMA 模型之前，往往需要确定模型的 p 和q 值，这通常通过模型的自相关函数（ACF）和偏自相关函数（PACF）来辅助确定。此外，模型需要时间序列数据是平稳的，如果数据不平稳，往往需要通过差分等手段将其转化为平稳时间序列。

网络访问量的时间序列在一定时间内是比较平缓的，比如在每天的凌晨至第二天上午 7 点，七点到下午 5 点。这个时间点以外的访问量就比较复杂。遇见节日天气，访问量的预测就愈发困难。ARMA 模型适合处理和预测短期的平稳时间序列数据，而在处理非平稳时间序列时，经常会使用 ARMA 模型的扩展版-ARIMA 模型。

1. 自回归差分移动平均模型（ARIMA）。

ARIMA 模型是在 ARMA 模型的基础上增加了差分的操作，以处理非平稳时间序列数据。ARIMA 模型的一般形式为 ARIMA(p, d, q)。其中，p 是自回归项的阶数，d 是差分次数，q 是移动平均项的阶数。ARIMA 模型的一般形式为

(1 − 𝜙1𝐵 − ⋯ − 𝜙𝑝𝐵𝑝)(1 − 𝐵)𝑑𝑋𝑡 = (1 + 𝜃1𝐵 + ⋯ + 𝜃𝑞𝐵𝑞)𝜖𝑡 (3.4)

其中，B 是差分操作符，𝜙1, 𝜙2, … , 𝜙𝑝 是自回归项的系数，𝜃1, 𝜃2, … , 𝜃𝑞 是移动平均项的系数，𝜖𝑡 是白噪声误差项。ARIMA 模型的参数通常通过极大似然

估计（Maximum Likelihood Estimation, MLE）或者最小二乘法来确定。在使用 ARIMA 模型之前，往往需要确定模型的 p、d 和 q 值，这通常通过模型的自相关函数（ACF）和偏自相关函数（PACF）来辅助确定。ARIMA 模型的优点是能够处理非平稳时间序列数据，但是它的缺点是需要对模型的参数进行较为复杂的确定。

## 访问量的时间序列预测算法

对于访问量的预测能够有效的对服务器集群的负载能力加以调节，避免访问量过大导致的服务器集群某一节点的负载过高，使得任务处理能力下降，从而达到增强负载均衡的作用。

由于相关条件限制，未使用近几年的数据网页访问量数据作为原始数据。因此在此次预测模型测试中所使用的数据集为 Kaggle 比赛“WebTrafficTimeSeries Forcasting”所提供的的网络访问量数据集，该数据集收集了一些维基词条的日访问量情况，一共将近 145000 篇的维基百科页面。train\_\*.csv 类型的文档包含了流量数据。其中每一行对应一篇特定的文章，每一列对应一个特定的日期。 train\_1.csv 提供了从 2015 年 7 月到 2016 年 12 月一年多时间里共 550 天的的日访问情况，train\_2.csv 提供了从 2015 年 7 月到 2017 年 9 月共 803 天的日访问情况。key\_\*.csv 类型的文件提供用于预测的页名称和缩短的 ID 列之间的映射。

训练数据集由大约 145k 个时间序列组成。每个时间序列代表从 2015 年 7 月 1 日到 2016 年 12 月 31 日不同维基百科文章的每日浏览量。每个时间序列，都有文章的名称以及该时间序列代表的流量类型（全部、移动、桌面、爬虫）。此元数据和任何其他公开可用的数据来进行预测。缺点是该数据集的数据源不区分零流量值和缺失值。缺失值可能意味着流量为零或当天的数据不可用。本文采用了维基百科“泰勒·斯威夫特”词条的页面访问情况，如图 3.1 所示。

为了能够知道模型的准确地，使用了重要指标SMAPE（对称平均绝对百分比误差）一种常用于预测精度评估的指标，可以比较不同规模时间序列的误差。计算公式如下。

𝑛

100

𝑆𝑀𝐴𝑃𝐸 = ∑

𝑛

𝑡=1

|𝐹𝑡 − 𝐴𝑡| (|𝐴𝑡| + |𝐹𝑡|)/2

(3.5)

其中这个公式的 𝑛 是观察总数，𝐹𝑡 是时间 t 的预测值，𝐴𝑡 是时间 t 的实际

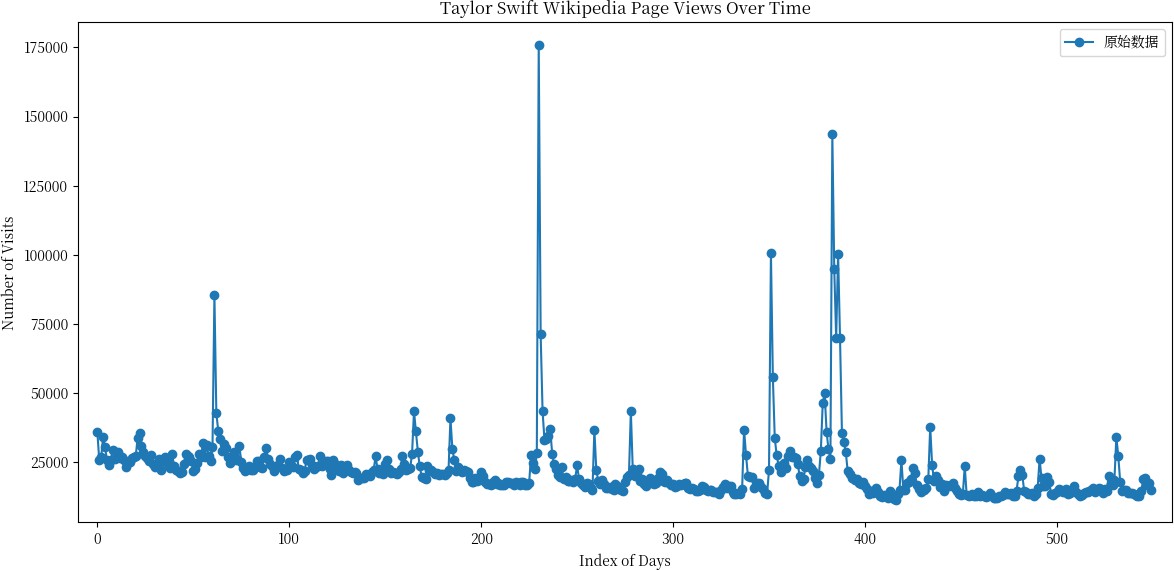


图 3.1 泰勒·斯威夫特日访问量变化

值。SMAPE 的值范围是 0% 到 200%，较低的 SMAPE 值表示较高的预测精度。与其他评估指标相比，它同时考虑了预测值和实际值的误差，避免了过分强调其中一个SMAPE 通过相对误差进行测量，使其不依赖于数据的规模，因此适合比较不同规模的时间序列。

然而，SMAPE 也存在一定的局限性，当时间序列的值接近零时，SMAPE可以变得非常有问题。这是因为当实际值 ( A\_t ) 或预测值 ( F\_t ) 接近于零时， SMAPE 的分母会变得很小，即使是很小的绝对误差也会导致很大的百分比误差。下面图 3.2 是关于 SAMPE 指标的极端测试图。

在图片[3.2](#_bookmark15)中，垂直的线代表了真实值 1, 2, 3 通过不同真实值的表现体现了 SAMPE 的动态分数。当实际值和预测值的差接近于零是，SMAPE 的分数会接近于无穷，所以要避免这个问题。

* + 1. ARIMA 算法访问量预测

ARIME 模型也被称为 Box-Jenkins 模型，可能包含自回归项（AR）、移动平均（MA）、差分运算（I）。

自回归即当前时刻的观察值是之前多个时刻观察值的线性组合加上一个随机误差项。基本上，自回归的核心思想是，过去的值对未来的值有一定的预测能力, 隐含地假设未来将与过去相似。因为这个核心思想，所以必然的会有一定程度上的缺点。

移动平均（MA）是一种统计手段，用于分析时间序列数据。通过创建一系

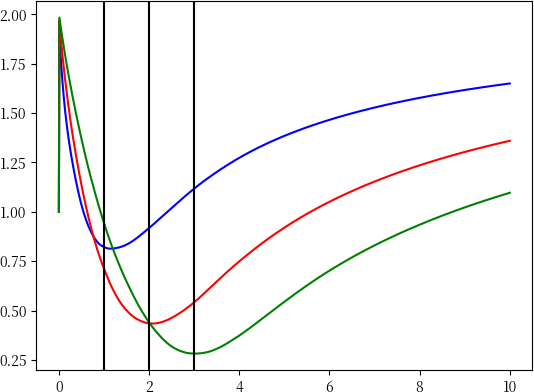


图 3.2 SAMPE 指标极端测试

列平均数来平滑短期波动并突出显示长期趋势或周期。简单的举一个移动平均磨的模型，计算公式为𝑆𝑀𝐴 = 𝑃1+𝑃2+⋯+𝑃𝑛 ，其中P 是不同时间下的价格或者观

𝑛

察的序列，而 n 是时间周期的长度即周期内一共有多少观察的时间数目

差分运算（I）是在时间序列分析中用于使非平稳的时间序列数据变得平稳的一种方法。如果一个时间序列是平稳地，意味着它的统计特性（均值、方差）不随着时间的变化。平稳性是许多统计模型和预测方法的重要前提，非平稳时间序列可以通过差分变成平稳序列，这就是差分运算的作用。差分运算的基本形式是计算时间序列中连续观测值之间的差异。举个例子，一阶差分的计算公式为𝛥𝑌𝑡 = 𝑌𝑡 − 𝑌𝑡−1，很好理解，如果一阶差分的时间序列不平稳，则可以进行更高级的差分，比如二阶差分：

𝛥2𝑌𝑡 = (𝑌𝑡 − 𝑌𝑡−1) − (𝑌𝑡−1 − 𝑌𝑡−2) (3.6)

一般来说，一阶或二阶差分足以处理大多数的非平稳序列。差分运算是构建 ARIMA（自回归积分滑动平均）模型的关键步骤之一

对于 ARIMA 模型来说，可以通过上面三个关键含义来理解这个模型。自回归是指显示变化变量的模型，该变量根据自身的滞后值或者先验值进行回归；差分，表示原始观测值与先前值的差异，用来使时间序列变得平稳；移动平均结合

了观测值与应用于滞后的观测值的移动平均模型之间的残差之间的依赖性

* + - 1. 数据的预处理

由 于 Kaggle 比 赛 中 使 用 的 数 据 命 名 的 混 乱， 比 如 存 在 52\_Hz\_I\_Love\_You\_zh.wikipedia.org\_all-access\_spider 复杂的命名需要进行重新命名和分类，同时需要填充不存在的数值为 NaN 同时由于 ARIMA 模型对数据进行分析和预测要求序列是有一个零均值平稳随机过程产生，因此需要需要判断数据的平稳性[[32](#_bookmark75)]，通过使用 ADF 检验序列中是否存在单位根，如果存在则为非平稳时间序列。ADF 检验的核心是基于一个时间序列模型来测试序列中的单位根。具体来说，ADF 检验通常使用以下的回归模型来进行单位根的测试。

𝛥𝑦𝑡 = 𝛼 + 𝛽𝑡 + 𝛾𝑦𝑡−1 + 𝛿1𝛥𝑦𝑡−1 + ⋯ + 𝛿𝑝𝛥𝑦𝑡−𝑝 + 𝜖𝑡 (3.7)

在这个公式中，𝑦𝑡 是时间序列数据，𝛥 是一阶差分运算符，𝛥𝑦𝑡 = 𝑦𝑡 − 𝑦𝑡−1，𝑡是趋势项，𝛼 是常数项，𝛽 是时间趋势项的系数，𝛾 是滞后一阶的系数，ADF 检验的关键在于检验这个系数；如果 𝛾 = 0，则存在单位根，𝛿1, 𝛿2, ⋯ , 𝛿𝑝 是差分项的系数，𝜖𝑡 是随机误差项，𝑝 是滞后阶数，它可以根据各种信息准则（如 AIC或 BIC）来确定。下表给出了泰勒·斯威夫特词条进行 ADF 检验的结果。

测试临界值

表 3.1: 泰勒·斯威夫特的 ADF 检验参数 值

level 1% -3.442361e+00

level 5% -2.866838e+00

level 10% -2.569592e+00 t 统计量 -7.674316e+00

P 值 1.563457e-11

ADF 的检验结果可知，ADF 值即t 统计量远远小于level 1%、level 5%、level 10% 的三个显著性水平条件下的临界值，故该原始序列为平稳时间序列，非常适合 ARIMA 模型来进行预测。

* + - 1. 参数标定

根据 ADF 的检验结果得知，该时间序列是一个平稳的时间序列，已经不需要进行差分，所以参数d 即原始观测值差异的次数；也称为差异度为 0。而q 和

p 的数值则需要通过自相关及偏自相关函数来进行判断。

在时间序列分析中，p 和q 是 ARIMA 模型（自回归差分移动平均模型）的关键参数[[33](#_bookmark76)]，其中p 代表自回归 (AR) 项的阶数，它是过去值的数量，q 代表移动平均 (MA) 项的阶数，它是预测误差的数量。为了得到合适的p 和q 值，自相关函数（ACF）为移动平均组件 q 提供界限。ACF 图显示了时序和它自己滞后版本之间的相关性。偏自相关函数（PACF）：为自回归组件 p 提供界限。PACF图仅显示不被更早滞后项解释的当前滞后项的相关性。自相关和偏自相关函数系数和滞后值关系如下：

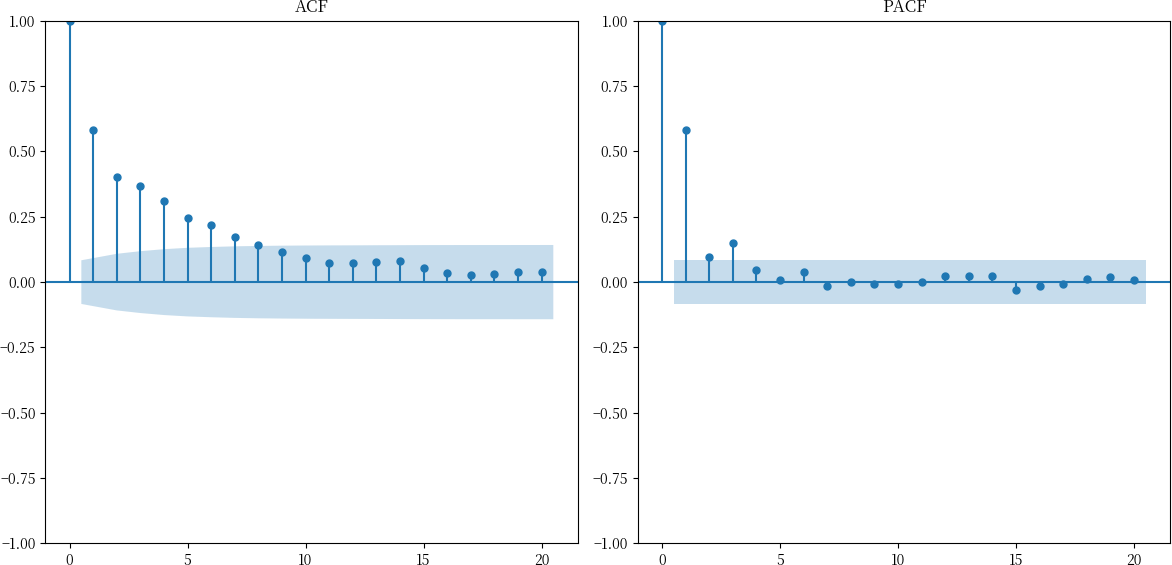


图 3.3 ACF 与 PACF 相关系数和滞后值的联系

如图 3.2 所示，x 值即滞后值，y 值即相关系数，蓝色的区域是显著性水平分界线，如果在蓝色之内则是非显著的，因为我们必须要对数据具有显著性的要求，毕竟如果没有显著性的差异，也就很难有什么变化或者预测了。ACF 图和 PACF 表示的是一个 AR 过程和 MA 过程一个相关性不断减小的过程。当 PACF图在滞后数p 之后截尾，即相关性突然下降到不显著水平，则可以用来辨识AR模型的阶数 p 即为 2；当 ACF 图在滞后数 q 之后截尾即相关性突然下降到不显著水平，则可以用来辨识 MA 模型的阶数 q 即为 1。利用 Python 对该模型的个参数进行估计，得到参数估计结果见表 3.6。

表 3.2: ARIMA 模型拟合结果

|  |  |  |  |
| --- | --- | --- | --- |
| 模型 | 系数 | t 统计量 | P 值 |
| AR（1） | 0.169 | 6.796 | 0.000 |
| AR（2） | 0.107 | -2.310 | 0.021 |
| MA（1） | 0.163 | -4.059 | 0.000 |
| SMA（2） | 3.086 | 3.4e+07 | 0.000 |

其中 𝑝 值是一个统计假设检验的概念，帮助决定观察到的数据是否显著性的与假设矛盾，进而否定这个假设。如果 𝑝 值很小，小于 0.05 那么就说结果具有统计学上的显著性，这表示出现极端数据的概率很低，反之则没有统计学上的相关性，很难找到统计学上的意义。可以看出 ARIMA（2，0，1）在 1% 的显著水平下，AR（1）、AR（2）、MA（1）、SMA（2）的相关系数显著不为 0，回归系数的 𝑝 值均小于 0.05，所以对于泰勒斯威夫特这个词条，通过了显著性检验。

已经确定了𝑝，𝑑，𝑞，的值，则可以直接对泰勒·斯威夫特的时间进行预测，图 3.4 是泰勒·斯威夫特的 ARIMA 预测值与实际值的对比。

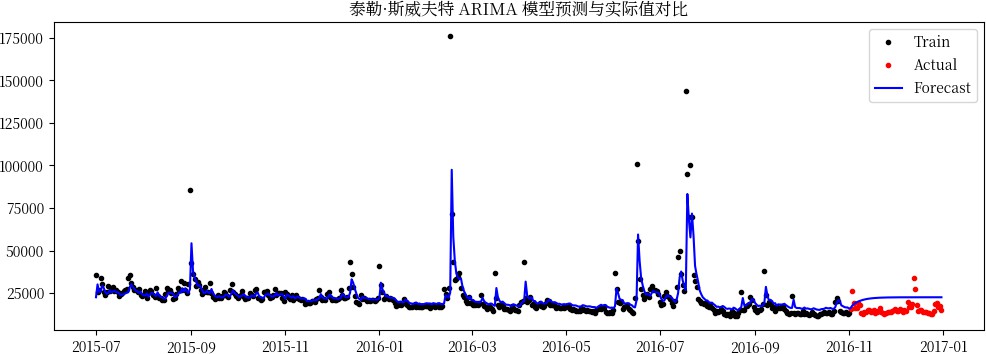


图 3.4 泰勒·斯威夫特词条 ARIMA 模型预测结果

可以看到蓝色的线条基本覆盖掉原始的训练实际值，但是在使用拟合的模型来进行预测的时候，可以看到蓝色的线条并不能很好的覆盖掉真实的实际值即模型仍旧没有很好的预测到真实的实际情况，需要进一步的分析研究。

在某些场景下，ARIMA 模型通常被认为是适用于短期预测的，自回归部分依赖于时间序列的先前值，而移动平均则依赖于预测误差的先前值。因此，模型特别关注数据中的局部波动，而这些往往在短期内更加稳定。但是在长期

预测场景下，可能存在许多不可预测的外部因素和趋势变化，这些因素很难被 ARIMA 模型所捕捉。为了解决上述问题，让时间序列更好的得到预测，笔者使用了 Facebook 的 prophet 模型对词条进行预测。

* + 1. Facebook prophet 算法访问量预测

Facebook 的 Prophet 模型是一个开源的时间序列预测工具，由 Facebook 的 Core Data Science 团队开发[[34](#_bookmark77)]。Prophet 的设计目标是让时间序列的预测工作变得更加简单，并且它对于具有强季节性效应的历史数据尤其适用。它在处理缺失数据和异常值方面具有鲁棒性，并且能容忍时间序列中的一些非常规变化，因此对实际应用场景非常友好。此外，Prophet 还支持添加重要的节假日效应。

Prophet 模型在使用上很灵活，使得用户可以根据自己的需求和熟悉的环境来选择合适的工具进行预测。Prophet 背后的模型基础是一个可加性模型，其中趋势、季节性和节假日效应可以分解非常清晰地展示出来，这使得模型的结果很容易解释。

在时间序列分析领域，有一种常见的分析方法叫做时间序列的分解（Decom- position of Time Series），它把时间序列 𝑦𝑡 分成几个部分，分别是季节项𝑆𝑡 趋势项 𝑇𝑡 ，剩余项𝑅𝑡；一般来说，在实际生活和生产环节中，除了季节项，趋势项，剩余项之外，通常还有节假日的效应。所以，在prophet 算法里面，作者同时考虑了以上四项，也就是：𝑦(𝑡) = 𝑔(𝑡) + 𝑠(𝑡) + ℎ(𝑡) + 𝜖𝑡

其中 𝑔(𝑡) 表示趋势项，它表示时间序列在非周期上面的变化趋势；𝑠(𝑡) 表示周期项，或者称为季节项，一般来说是以周或者年为单位；ℎ(𝑡) 表示节假日项，表示在当天是否存在节假日；𝜖𝑡 表示误差项或者称为剩余项。Prophet 算法就是通过拟合这几项，然后最后把它们累加起来就得到了时间序列的预测值。

（1）趋势项 𝑔(𝑡)

在Prophet 算法里面，趋势项有两个重要的函数，一个是基于逻辑回归函数

（logistic function）的，另一个是基于分段线性函数（piecewise linear function）的。在逻辑回归函数中，一般形式是𝜎(𝑥) = 1/(1 + 𝑒−𝑥) ，它的导数是𝜎′(𝑥) =

𝜎(𝑥) ⋅ (1 − 𝜎(𝑥))，而且 lim𝑥→+∞ 𝜎(𝑥) = 1, lim𝑥→−∞ 𝜎(𝑥) = 0。如果增加一些参数的话，那么逻辑回归就可以改写成如下方程[[35](#_bookmark78)]。

𝐶

𝑔(𝑡) = 1 + 𝑒−𝑘(𝑡−𝑚) (3.8)

方程式中，𝐶 是曲线的最大渐进值，即 𝑔(𝑡) 随着时间增加越来趋向于𝐶；𝑘 为曲

线的增长率；m 为曲线的偏移量。但是在现实环境中，函数 𝑔(𝑡) = 𝐶 的

1+𝑒−𝑘(𝑡−𝑚)

三个参数 𝐶, 𝑘, 𝑚 不可能都是常数，而很有可能是随着时间的迁移而变化的，因此，在 Prophet 里面，作者考虑把这三个参数全部换成了随着时间而变化的函数，也就是𝐶 = 𝐶(𝑡), 𝑘 = 𝑘(𝑡), 𝑚 = 𝑚(𝑡)。在现实的时间序列中，曲线的走势肯定不会一直保持不变，在某些特定的时候或者有着某种潜在的周期曲线会发生变化，这种时候，就有学者会去研究变点检测

假设已经放置了 𝑆 个变点，并且变点的位置是在时间戳 𝑠𝑗, 1 ≤ 𝑗 ≤ 𝑆上，在这些时间戳上，我们就需要给出增长率的变化。如果一开始的增长率我们使用 𝑘 来代替的话，那么在时间戳 𝑡 上的增长率就是 𝑘 + ∑𝑗∶𝑡>𝑠𝑗 𝛿𝑗 通过

一个指示函数 a(𝑡) ∈ {0, 1}𝑆 那么在逻辑回归函数的形式如下，其中 a(𝑡) =

(𝑎1(𝑡), ⋯ , 𝑎𝑆(𝑡))𝑇, 𝛿 = (𝛿1, ⋯ , 𝛿𝑆)𝑇, 𝛾 = (𝛾1, ⋯ , 𝛾𝑆)𝑇。

𝐶(𝑡)

𝑔(𝑡) = 1 + exp(−(𝑘 + a(𝑡)𝑡𝛿) ⋅ (𝑡 − (𝑚 + a(𝑡)𝑇𝛾) (3.9)

在 prophet 程序里面有两种方法，一种是通过人工指定的方式指定变点的位置；另外一种是通过算法来自动选择。在默认的函数里面，Prophet 会选择 n\_changepoints = 25 个变点，然后设置变点的范围是前 80%（changepoint\_range），也就是在时间序列的前 80% 的区间内会设置变点。

在分段线性函数的趋势项中，在每一个子区间上，函数都是线性函数，但是在整段区间上，函数并不完全是线性的。故根据变点的位置和变化量之间的关系，得出的分段线性函数如下。

𝑔(𝑡) = (𝑘 + a(𝑡)𝛿) ⋅ 𝑡 + (𝑚 + a(𝑡)𝑇𝛾) (3.10)

其中𝑘 表示全局增长率（growth rate），𝛿 表示增长率的变化量，𝑚 表示偏移量，𝑎(𝑡) 是一个向量，其中的每一项在t 小于突变点时为 0，在突变点之后为 1。而这两种方法（分段线性函数与逻辑回归函数）最大的区别就是 𝛾 的设置，在分段线性函数中，𝛾 是校正项，其定义为在突变点上的增长率乘以突变点的时间即 𝛾 = (𝛾1, ⋯ , 𝛾𝑆)𝑇，𝛾𝑗 = −𝑠𝑗𝛿𝑗

针对泰勒·斯威夫特词条的数据，在趋势项中没有什么值得改变的，顶多需要指定一下变点的位置，本文选择使用程序算法自己选择。

（2）季节项 𝑠(𝑡)

几乎所有的时间序列预测模型都会考虑这个因素，因为时间序列通常会随

着天，周，月，年等季节性的变化而呈现季节性的变化，也称为周期性的变化。在论文中[[34](#_bookmark77)]，作者使用傅立叶级数来模拟时间序列的周期性，当 𝑇 = 7, 𝑁 = 3时，表示以周为周期，当 𝑇 = 362.25, 𝑁 = 10 表示以年为周期。

𝑁

𝑠(𝑡) = ∑ (𝑎𝑛 cos (

𝑛=1

2𝜋𝑛𝑡

𝑃 ) + 𝑏𝑛 sin (

2𝜋𝑛𝑡

)) (3.11)

𝑃

下图[3.5](#_bookmark16)是对泰勒·斯威夫特词条时间序列的趋势分解图，包括趋势，季节性，残差，经过趋势分解，发现并没有明显的季节性因素。

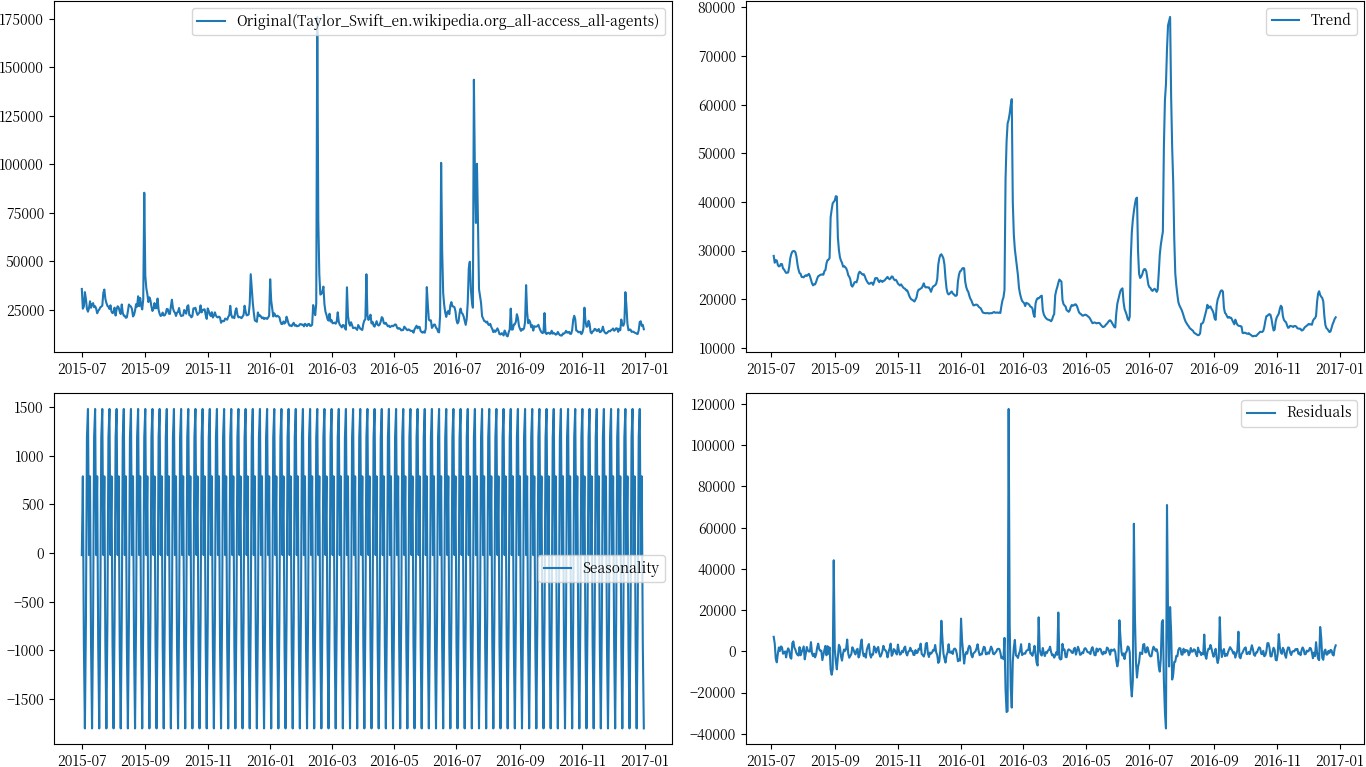


图 3.5 泰勒·斯威夫特趋势分解

(3) 假日项以及特殊因素

ℎ(𝑡) 可用来反映时间序列中某时刻的特殊变动，Prophet 模型根据每个假日项在不同时刻下产生的影响构建独立的模型，并为各个假日项设置不同的前后窗口期，以及产生相应的虚拟变量[[36](#_bookmark79)]。ℎ(𝑡) 的表达形式如下。

𝐿

ℎ(𝑡) = ∑ 𝐾𝑖1 (𝑡 ∈ 𝐷𝑖), (3.12)

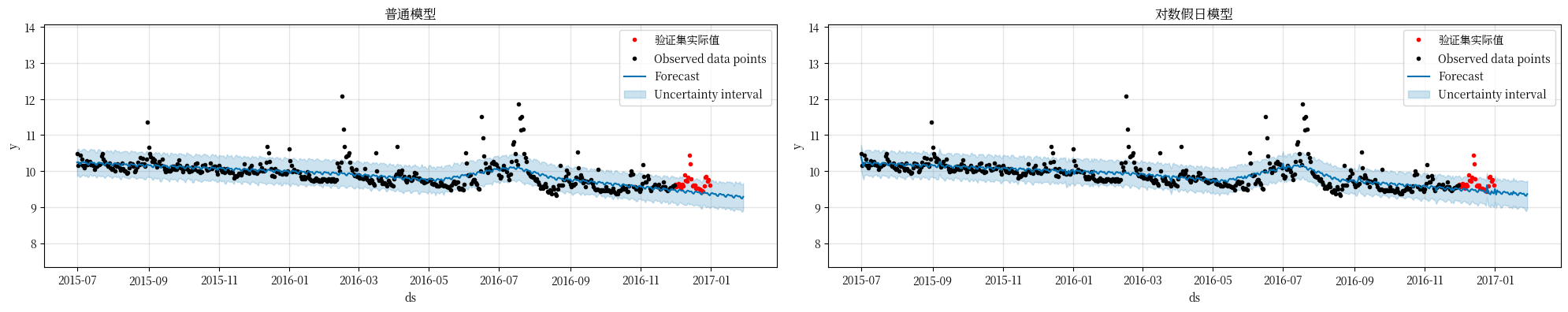
𝑖=1

𝑍(𝑡) = [1(𝑡 ∈ 𝐷1), ..., 1 (𝑡 ∈ 𝐷𝐿)], (3.13)

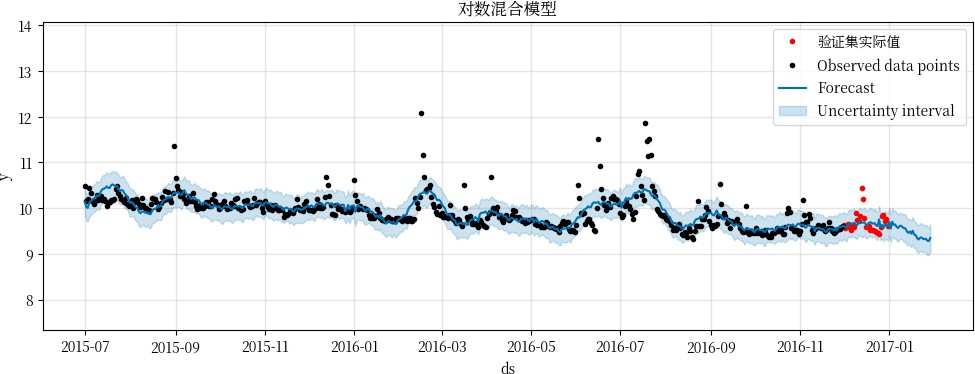
ℎ(𝑡) = 𝑍(𝑡)𝑘, 𝑘 ∼ 𝑛𝑜𝑟𝑚𝑎𝑙(0, 𝛾) (3.14)

在现实环境中，除了周末，同样有很多节假日，而且不同的国家有着不同的假期。在 Prophet 里面，通过维基百科里面对各个国家的节假日的描述，hdays.py收集了各个国家的特殊节假日。针对泰勒·斯威夫特这个词条来说，每一次专辑的发售都是一次重要的影响，发售专辑的时间可以算做一个假日。

通过加上不同影响因素，模型的准确度也会不一样。笔者通过加上不同的选项比如周期性，假日性和普通prophet 模型加以对比，选出了效果比较好的一个模型。



(a) ProPhet 普通模型 (b) Prophet 假日模型



(c) Prophet 混合模型

图 3.6 Prophet 几个模型的结果

图[3.6](#_bookmark17)中，使用普通模型的 SMAPE 的分数为 0.25758，使用假日模型的 SMAPE 的分数为 0.22900，使用混合模型的 SMAPE 的分数 0.14071。SMAPE的分数越低，代表着模型的预测性能越好。三个图片中，小红点是验证集的实际值，小黑点显示训练集的实际值，蓝色的线代表着prophet 模型的预测值，蓝色的区域是置信区间，上面是可能区间上限，下面是可能区间下限。可以很直观的看到，仍旧存在某些没有预测到的特殊因素，需要额外的参数输入。经过查询与泰勒·斯威夫特的新闻有很大关系。总的来说虽然蓝色的线并没有覆盖掉全部的训练集数据，但是在其他的条件下，基本有效的预测了访问量数据。考虑了周

期性和假日性的 prophet 还是没有达到最好的效果，不过勉强也可以了。

## 两种算法的比较以及选择

通过SMAPE（对称平均绝对百分比误差）的比较，得到了了两个模型根据相同数据下最好的参数下的表现。但是仅仅是针对这一条数据下的表现，不具有普遍性，所以需要对每一个词条都要做拟合并预测得到每一个词条的 SAMPE分数。最后比较最后的平均值即可。结果如下图[3.7](#_bookmark19)所示。

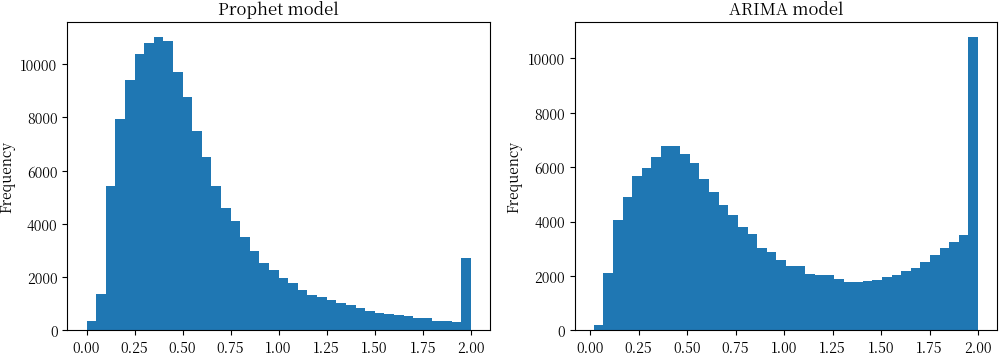


图 3.7 模型 SMAPE 分数的比较

很直观的看到了两个模型的性能，prophet 的最后SAMPE 得分为 0.586509， ARIMA 的最后得分为 0.919387。虽然在某些个词条例如泰勒斯威夫特下表现和 prophet 的模型没有太大差异，但是没有经过一个一个对词条的参数的调整

（笔者最后使用 auto\_arima 自动选择合适的词条参数），拟合结果就不尽人意， ARIMA 的预测准确度会大大下跌。所以最后选择了考虑了周期性（季节性）、假日性、特殊因素、趋势项的 prophet 模型。

## 本章小结

在本章节首先介绍了数据的来源以及数据的特点，并对预测词条的时间序列做了处理。然后探讨当前常见的时间序列预测算法，决定使用 ARIMA 和近年来比较先进的prophet 算法进行实验和探究。最后比较两个算法对所有词条的分数的平均值，结果表明 prophet 算法有更好的预测效果。

# 基于时间卷积网络的剩余性能的预测

服务器集群运行时，由于访问时间，访问任务的不同，某些时间段整个集群可能处于忙碌的状态，某些节点可能处于任务繁忙任务重的而其他节点空闲的问题。这样的两种节点是两种状态，高负载状态和低负载状态，处于低（高）负载状态的需要通过负载均衡器调整为正常负载状态，可以提高集群的总体资源利用率，环节高负载节点的压力。因此本章通过对于时间卷积网络的研究，针对某一个节点的剩余性能的预测，使得对服务器节点的负载分配进行均衡的调整。本文通过对普遍的 Linux 进行研究来获取相关数据。

## 服务器剩余性能的获取

动态负载均衡算法的负载性能会受到服务器负载指标选取的影响。选择合适的负载信息生成合适的负载指标能够真实的反映服务器运行是的负载情况，提升算法的准确度和动态负载均衡的负载效果。在第二章中已经知道了有哪些负载信息值得挑选和收集，但是没有一个真正的合适的负载综合指标来体现真实的负载情况。具体的剩余性能的获取可以使用 Linux 的proc 文件系统获取，该文件储存在内存里，不会占用外存的空间。

1. CPU 信息的获取

要获取Linux 系统中的 CPU 详细使用情况可以使用cat/proc/stat，CPU 详细使用情况如下表。

表 4.1: CPU 详细信息解释

|  |  |
| --- | --- |
| 参数 | 解释 |
| user | 系统从启动到当前时刻，用户态 CPU 时间，不含 nice 值为负的进 |
|  | 程 |
| nice | 系统从启动到当前时刻，nice 值为负的进程占用的 CPU 时间 |
| system | 系统从启动到当前时刻，核心时间 |
| idle | 系统从启动到当前时刻，除硬盘等待外其他等待时间 |

Continued on next page

表 4.1: CPU 详细信息解释 (Continued)

|  |  |
| --- | --- |
| iowait irq  softirq | 系统从启动到当前时刻，硬盘等待时间系统从启动到当前时刻，硬中断时间  系统从启动到当前时刻，软中断时间 |

那么 CPU 利用率的计算公式为：𝐶𝑃𝑈𝑢𝑠𝑔 = 1−(𝑖𝑑𝑙𝑒2 −𝑖𝑑𝑙𝑒1)/(𝑐𝑝𝑢2 −𝑐𝑝𝑢1)通过使用cat /proc/cpuinfo 命令获取服务器 CPU 配置，然后在 benchmark 上可纯到该 CPU 性能的分数作为 𝐶𝑃𝑈𝑚𝑎𝑟𝑘。则一个节点的 CPU 的剩余性能如下。

𝑛∗𝐶𝑃𝑈𝑚𝑎𝑟𝑘

𝐶𝑗

= (1 − 𝐶𝑃𝑈𝑢𝑠𝑔𝑗)

𝑗

∑𝑛 𝐶𝑃𝑈

(4.1)

1 𝑚𝑎𝑟𝑘

1. 内存信息获取获取内存使用情况则可以 cat /proc/meminfo 命令，其中最重要的是 MemFree、Buffers、Cached 三个信息分别是完全空闲的内存大小、缓存文件系统的元数据和跟踪正在读写的块设备、存储页缓存和 slabs 的内存大小。所以内存使用率的计算方式如下。

𝑀𝐸𝑀𝑢𝑠𝑔 = (1 − 𝑀𝑒𝑚𝐹𝑟𝑒𝑒 + 𝐵𝑢𝑓𝑓𝑒𝑟𝑠 + 𝐶𝑎𝑐ℎ𝑒𝑑)/𝑀𝑒𝑚𝐹𝑟𝑒𝑒 (4.2)

1. 磁盘信息获取

磁盘信息可以通过 iostat -x /dev/sda1 命令获取。重要信息如下表。

表 4.2: 磁盘信息获取

|  |  |
| --- | --- |
| 标示 | 说明 |
| Device | 监测设备名称 |
| rkB/s | 每秒实际读取的大小，单位为 KB |
| wkB/s | 每秒实际写入的大小，单位为 KB |
| await | 等待 I/O 平均的时间（milliseconds） |
| svctm | I/O 需求完成的平均时间 |
| %util | 设备带宽的使用率，达到 100% 表示饱和，达到性能瓶颈，如果是支  持处理并发请求的设备则不代表性能瓶颈 |

1. 网络带宽信息获取

获取网络带宽使用 cat /proc/net/dev 命令， 统计一段时间内 Receive 和 Tramsmit 的 bytes 值变化，可以获取网口传输速率 𝑇𝑏𝑦𝑡𝑒𝑠，而网络带宽为 Band- width，则带宽利用率为：𝑁𝐸𝑇𝑢𝑠𝑔 = 𝑇𝑏𝑦𝑡𝑒𝑠/𝐵𝑎𝑛𝑑𝑤𝑖𝑑𝑡ℎ

获取了这几个重要的剩余性能的指标之后，也需要有一个能够体现综合负载情况的评价指标。为了以一种数学方法客观的计算出服务器节点当前的负载量，需要改造一个综合负载决策来组合各项负载指标。常见的指标组合形式有加权和法和加权乘法，通过阅读不同的文献可知，在构建服务器负载状态的数学模型中，加权和法具有不错的效果[[37](#_bookmark80)]。在以往的负载均衡策略研究中这些权重系数通常依靠经验赋值确定，缺乏必要的数学分析和定量依据，为了使得服务器的计算结果更加准确和可靠，本文选取了主成分分析法科学计算综合负载决策函数的权系数向量。

主成分分析法（Principal Componet Analysis, PCA）最是由 Pearson[[38](#_bookmark81)] 提出的一种多元统计方法，该方法通过正交变换将原始数据转换成一组由线性无关变量变量表示的数据，以达到数据降维的目的[[39](#_bookmark82)]。具体步骤如下。

已经收集了服务器节点的 CPU 利用率、内存利用率、设备 IO 使用率，和带宽使用率的数据，将这些数据保存在以个表中，每一列代表一个指标，每一行代表一个观测时间点的数据。由于主成分分析法的要求，将不同的特征量纲统一到相同的尺度，这样就可以公平地比较和组合这些特征，需要对给定数据集进

行标准化。对于每个指标，计算所有数据点的均值。公式为：𝜇 = 1 ∑𝑁 𝑥 其

𝑁 𝑖=1 𝑖

中 𝜇 为均值，N 是观测的个数，𝑥𝑖 每个观测的值。计算标准差：然后计算每个

指标的标准差。公式为：𝜎 = √ 1 ∑𝑁 ( 𝑥 − 𝜇 ( ))2 𝜎 其中 𝜎 是标准差。最后进

𝑁 𝑖=1 𝑖 2

行 Z-score 标准化：对每个指标的每个数据点进行标准化，公式为 𝑧 = (𝑥𝑖−𝜇) 其

𝜎

中 𝑧 就是最后每个指标每个观测时间的标准化后的值。

给定标准化后的数据集 𝑋 = [𝑥1, 𝑥2, ⋯ , 𝑥𝑛]T ∈ ℝ𝑛×𝑑，其中 𝑛 为样本的数量，

𝑑 指标的数量，结合在一起，𝑋 是一个由𝑛 个𝑑-维的实数向量构成的数据集，并且以矩阵形式表示，其中每一行是一个样本，每一列是一个特征。则数据集的协方差公式矩阵如下。

𝑛

𝐴 = 1 𝑋T𝑋 = 1 ∑ 𝑥 𝑥T = 𝑉𝛴𝑉T ∈ ℝ𝑑⋅𝑑 (4.3)

𝑛 𝑛

𝑖 = 1

𝑖 𝑖

对 A 进行特征分解后得到一组降序排列的特征值 𝜆1 ≥ 𝜆2... ≥ 𝜆𝑛 > 0 相对

应的单位特征向量𝑣1, 𝑣2, … , 𝑣𝑛。其中，特征值越大的，其对应的主成分越重要。于是，综合负载决策函数如下。

X = 𝜇1𝐶𝑢𝑠𝑔 + 𝜇2𝑀𝑢𝑠𝑔 + 𝜇3𝐷𝑢𝑠𝑔 + 𝜇4𝑁𝑢𝑠𝑔 (4.4)

通过指定欲保留的主成分比重阈值𝜃 ∈ (0, 1] 来确定目标唯独k 的大小。具

∑

TCN 模型参数组合体来说 ̨ 可以通过

𝑘

𝑖=1

𝑑

∑

𝑖=1

𝜆𝑖

𝜆𝑖

≥ 𝜃 来得到。取前 𝑘 个特征值对应

的特征向量组成的主成分空间 𝕍𝑑×𝑘 = (𝑣1, 𝑣2, ⋯ , 𝑣𝑘)。将标准化后的原始数据

集 X𝑛×𝑑 投影到由前 𝑘 个特征向量构成的主成分空间 V𝑑×𝑘 中，得到了降维后的

𝑘 维数据集。

Z𝑛×𝑘 = X𝑛×𝑑V𝑑×𝑘 (4.5)

在这个公式中，X𝑛×𝑑 表示标准化后的原始数据集，其中 𝑛 是样本数，𝑑 是原始特征数。V𝑑×𝑘 包括了前 𝑘 个特征值对应的特征向量（这些特征向量通常是按照特征值大小从大到小排列的），它们构成了主成分空间。这个空间的维度（𝑘）通常远小于原始特征空间的维度（𝑑）。

当原始数据集 X 与主成分空间 V𝑑×𝑘 相乘时，得到的 Z𝑛×𝑘 就是投影到这个

𝑘 维空间的数据集。在这个 𝑘 维空间中，每一行代表了原始数据的一个样本，每一列则对应一个主成分。Z 的列向量提供了在主成分空间中的坐标，即降维后的数据表示。

其中在进行特征分解后，就已经得到了主成分空间，空间内含有了最重要的 4 个特征值，这个特征值就是权重的系数，通过降维与原始数据值相乘后得到的就是每个指标乘以权重系数后的数据集，只需要对降维的数据集进行每一列的相加，即可对得到时间序列-综合负载指标的数据集。

## 时间卷积网络的关键特征

时间卷积网络由传统的 CNN 改进的专门处理时间序列的深度学习模型[[40](#_bookmark83)]。一开始认为，时序问题（如语言、语音等等）天生就是 RNN 的地盘。然而现在这一观点要成为过去式了。时间卷积网络（Temporal Convolutional Nets, TCNs）作为 CNN 家族中的一员健将，拥有许多新特性，如今已经在诸多主要应用领域中击败了 RNN。看起来 RNN 可能要成为历史了。

卷积神经网络（Convolutional Neural Nets, CNNs）是图像和视频识别领域公

认的主力军，而循环神经网络（Recurrent Neural Nets, RNNs）在自然语言处理领域的地位与其是相似的。但二者的一个主要不同是，CNN 可以识别静态图像

（或以帧分割的视频）中的特征，而 RNN 在文本和语音方面表现出色，因为这类问题属于序列或时间依赖问题。也就是说，待预测的后一个字符或单词依赖于前面的（从左到右）字符或单词，因此引入时间的概念，进而考虑到序列。实际上，RNN 在所有的序列问题上都有良好表现，包括语音/ 文本识别、机器翻译、手写体识别、序列数据分析（预测），甚至不同配置下的自动编码生成等等。但是它的问题是极度地计算密集，因为在整个任务运行完成之前，必须保存所有的中间结果，深度神经网络必须等前一个单词处理完，才能进行下一个单词的处理，于是 TCN 便应运而生。

时间卷积网络需要的数据集和 RNN 有一定的类似，在处理时间序列数据集时都要进行嵌入（Embedding）操作。Embedding 的主要目的是将时序数据映射到一个稠密的连续向量空间中，使得相似的语义信息在该向量空间中也能够彼此接近。这样，神经网络就可以基于这些向量表示学习到输入数据的复杂语义信息，并在相似的向量表示之间进行泛化。图是具体的嵌入过程。

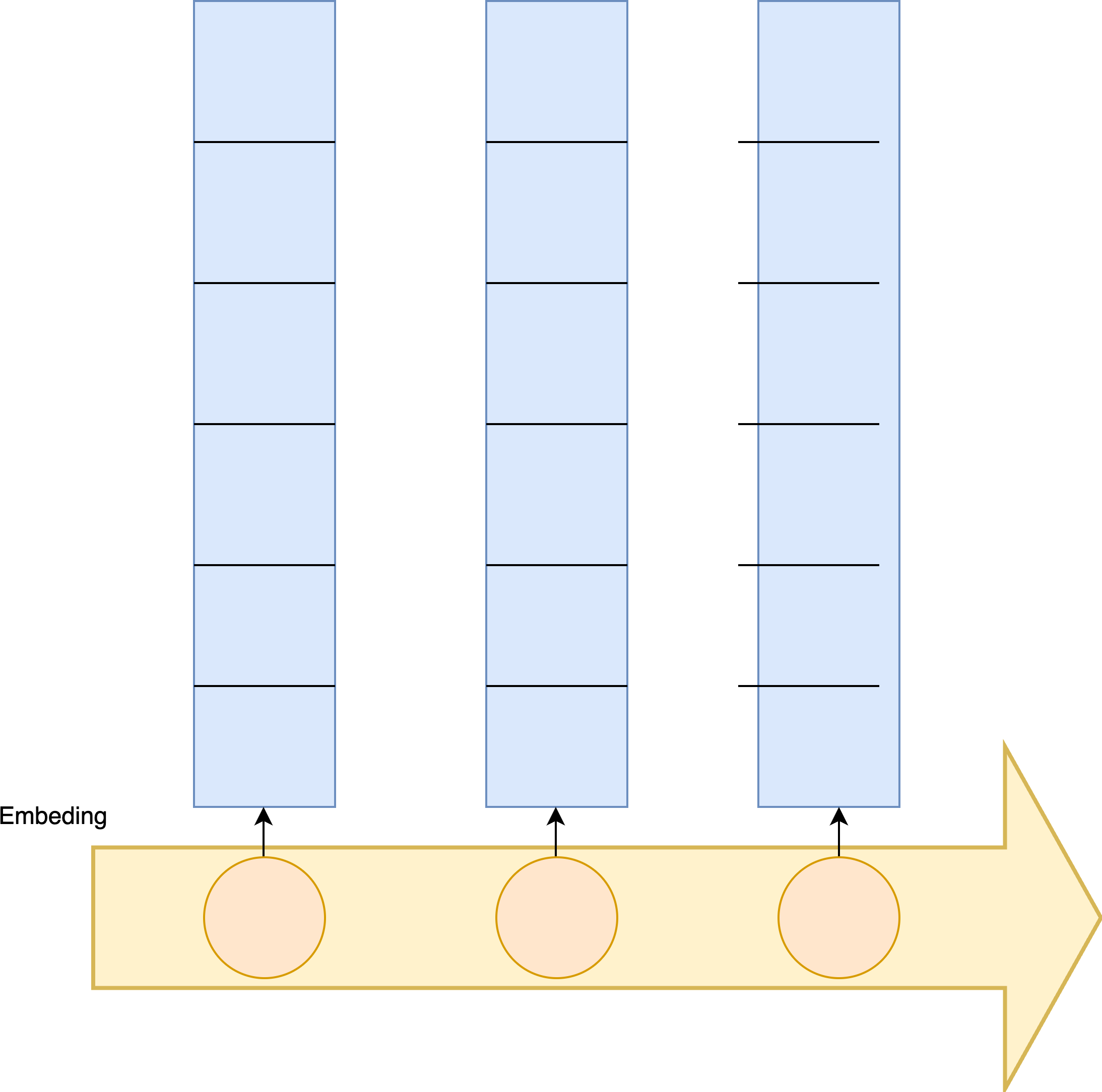


图 4.1 原始时序序列嵌入生成向量

TCN 所依赖的卷积操作在本质上就是一维卷积（1D-CNN）。一维卷积利用多个大小固定的卷积核与输入序列进行卷积运算来生成输出序列。卷积核的形状由输入通道数 𝑖𝑛\_𝑐ℎ𝑎𝑛𝑛𝑒𝑙𝑠 和卷积核大小 𝑘𝑒𝑟𝑛𝑒𝑙\_𝑠𝑖𝑧𝑒 共同决定，卷积核的数量则由输出通道数𝑜𝑢𝑡\_𝑐ℎ𝑎𝑛𝑛𝑒𝑙𝑠 决定。经过 Embedding 后的时序数据的通道数由 1 扩展成了 𝑒𝑚𝑏𝑒𝑑𝑑𝑖𝑛𝑔\_𝑠𝑖𝑧𝑒 ，对于有着 𝑖𝑛\_𝑐ℎ𝑎𝑛𝑛𝑒𝑙𝑠 个通道的时序数据作为输入, 一维卷积使用 𝑜𝑢𝑡\_𝑐ℎ𝑎𝑛𝑛𝑒𝑙𝑠 个大小为（𝑖𝑛\_𝑐ℎ𝑎𝑛𝑛𝑒𝑙𝑠, 𝑘𝑒𝑟𝑛𝑒𝑙\_𝑠𝑖𝑧𝑒）的卷积核进行卷积操作。

假设有一个时间序列，总共有五个时间点，比方说股市，有一个股票的价格波动：[10,13,12,14,15]。TCN 使用一个卷积大小为 2 的卷积核，对上面 5 个数据做一个卷积核大小为 2 的卷积是什么样子如下。

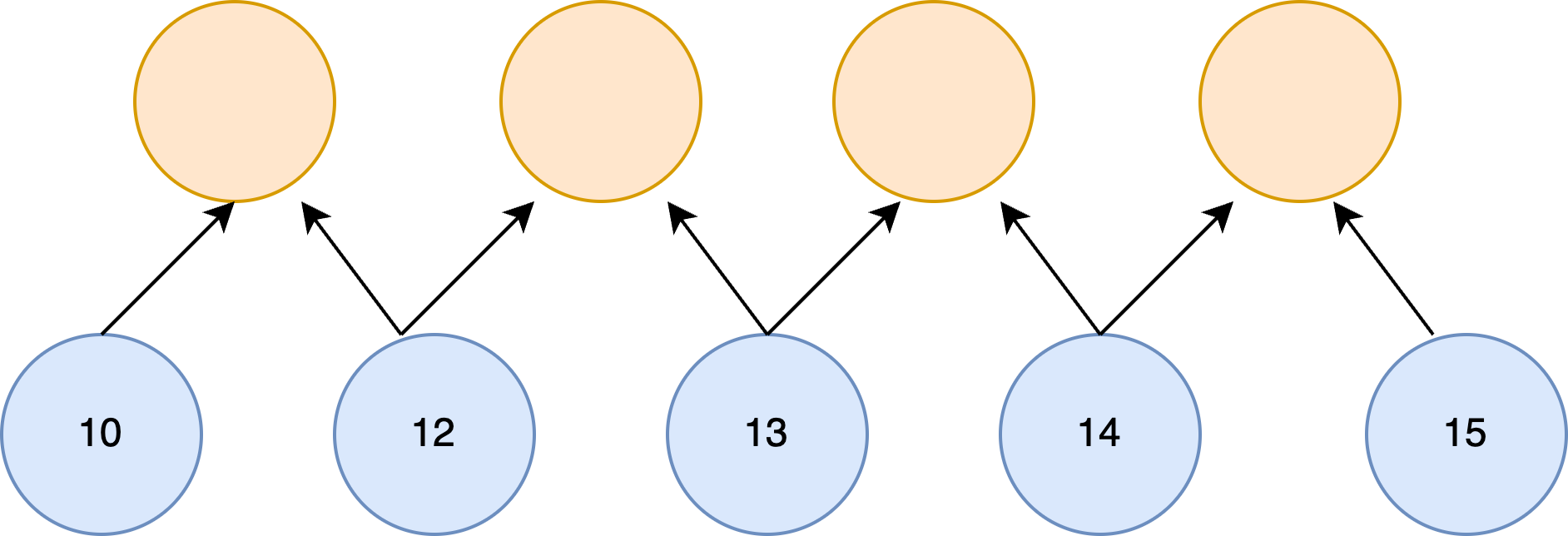


图 4.2 卷积核为 2 的卷积过程

五个数据经过一次卷积，可以变成四个数据，但是每一个卷积后的数据都是基于两个原始数据得到的，所以说，目前卷积的视野域是 2。可以看到是输入是 5 个数据，但是经过卷积，变成 4 个数据了。TCN 网络的第一个原则是输入和输出的形状必须相同。而在进行一维卷积操作时，假设第 𝑖 层的输入长度为𝐿𝑖卷积核大小为 𝐾 则输出长度为 𝐿𝑖+1 = 𝐿𝑖 − 𝐾 + 1。为满足这个原则，TCN 在使用卷积神经网络进行序列处理时，通常需要进行 Padding 操作。通过在输入序列的左侧添加一定数量𝐾 − 1 的 0，实现信号维度的保持，使通过卷积和池化处理后的数据与输入数据的长度相同。在一维卷积中进行的 Padding 操作默认会在左右都进行填充，所以 TCN 进行了额外的裁剪操作。

padding 是左右两头都增加 0，如果 padding 是 1 的话，就是上图的效果，其实会产生 6 个新数据，但是秉着：“输入输出尺寸相同”和“我们不能知道未来的数据”，所以最后边那个未来的 padding，就省略掉了，之后会在代码中会体现出来。

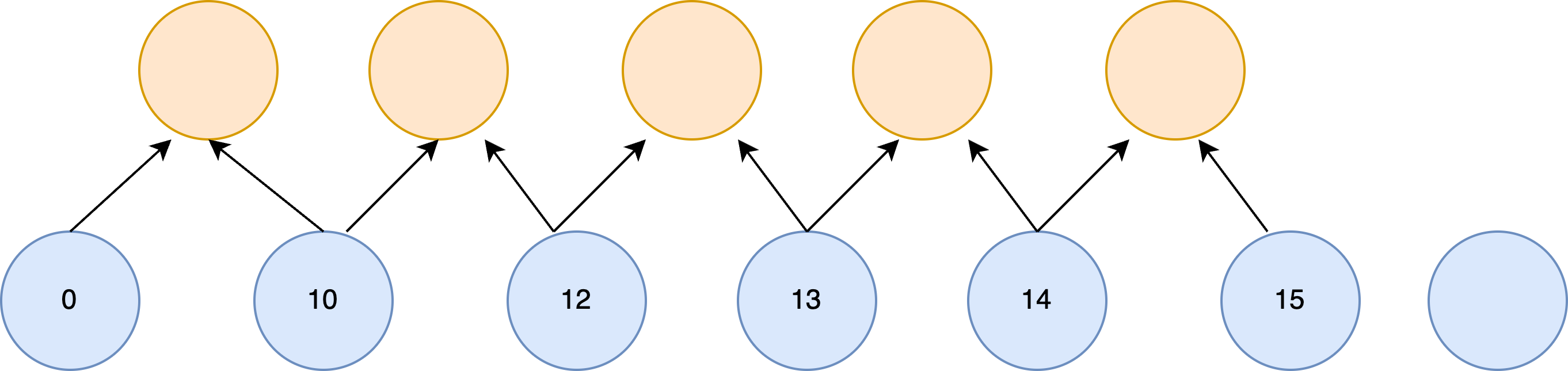


图 4.3 padding 后的卷积

时间卷积网络有两个原则，其一就是上面的 TCN 网络的输入和输出形状必须相同，以确保模型对时序数据的处理不会丢失信息。其二，TCN 网络中的每一时刻的输出仅由该时刻及其之前的输入卷积得到，以确保其在处理序列时具有因果约束。通过使用因果卷积满足原则二，因果卷积是一种只考虑过去时间状态的一维卷积操作。因果卷积的具体步骤如下图所示。

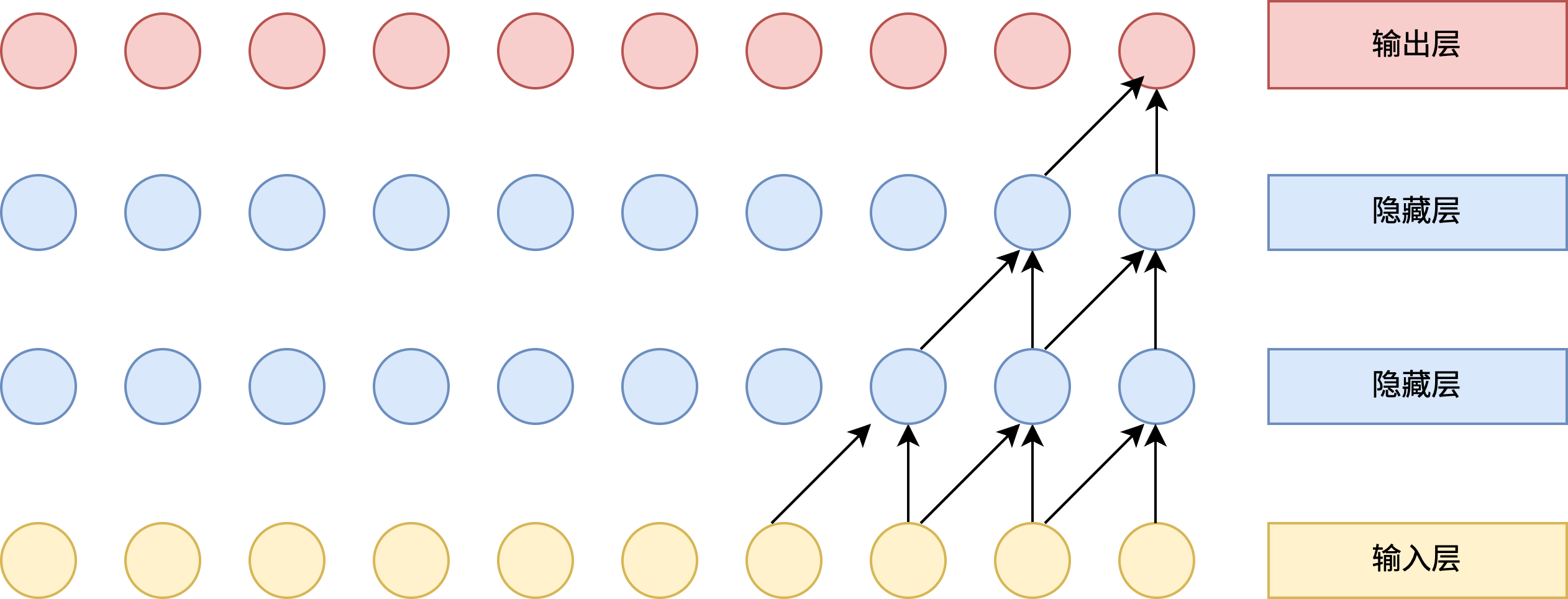


图 4.4 因果卷积

图中可以看到，因果卷积具备了两个特点。只考虑过去的信息。时刻的输出

𝑦𝑡 仅依赖于𝑥0 ... 𝑥𝑡 ，而不依赖任何“未来”的输入𝑥𝑡+1 𝑥𝑇。同时追溯历

史信息越久远，隐藏层越多。图中，输出层期望采集输入层的 4 个时间步，则需要 2 个隐藏层。而业务的需求往往要求采取更多的时间步，确实是“深度”学习了。

由于因果卷积没有循环连接，它们通常比循环神经网络（RNN）训练速度更快，尤其是应用于非常长的序列。但是，针对于一般的时间序列而言，往往是按照分钟记录的，那少说也是十万、百万的数据量，想要考虑之前 1000 个时间点呢？视野域要是 1000，那意味着要 999 层卷积。每经过一层，节点相对于前层减少一个，我们最后的输出只有一个节点，如果输入视野为 1000，需要经过

999 层才能变为最后输出的一个节点。啥计算机吃得消这样的计算。所以引入了膨胀因果卷积。

单纯的因果卷积还是存在传统卷积神经网络的问题，即对时间的建模长度是受限于卷积核大小的，如果要想抓去更长的依赖关系，就需要线性的堆叠很多的层。标准的 CNN 可以通过增加pooling 层来获得更大的感受野，而经过 pooling层后肯定存在信息损失的问题。

膨胀卷积是在标准的卷积里注入空洞，以此来增加感受野。和传统卷积不同的是，膨胀卷积允许卷积时的输入存在间隔采样，采样率受超参数 dilation rate控制，指的是做卷积操作时 kernel 里面的元素之间的下标间隔。空洞的好处是不做pooling 损失信息的情况下，增加了感受野，让每个卷积输出都包含较大范围的信息。下图就是 dilation=2 的时候的膨胀因果卷积的情况，

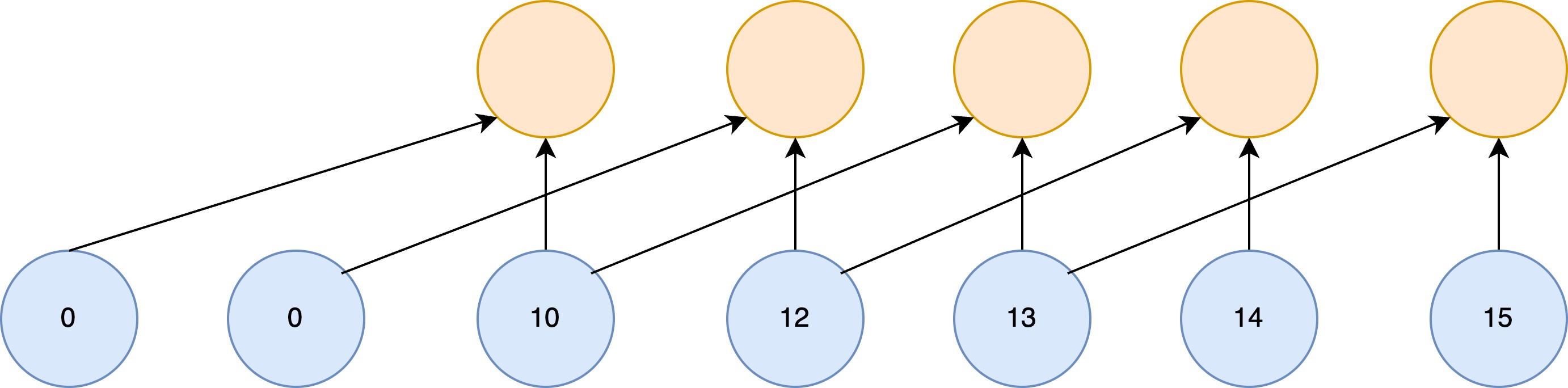
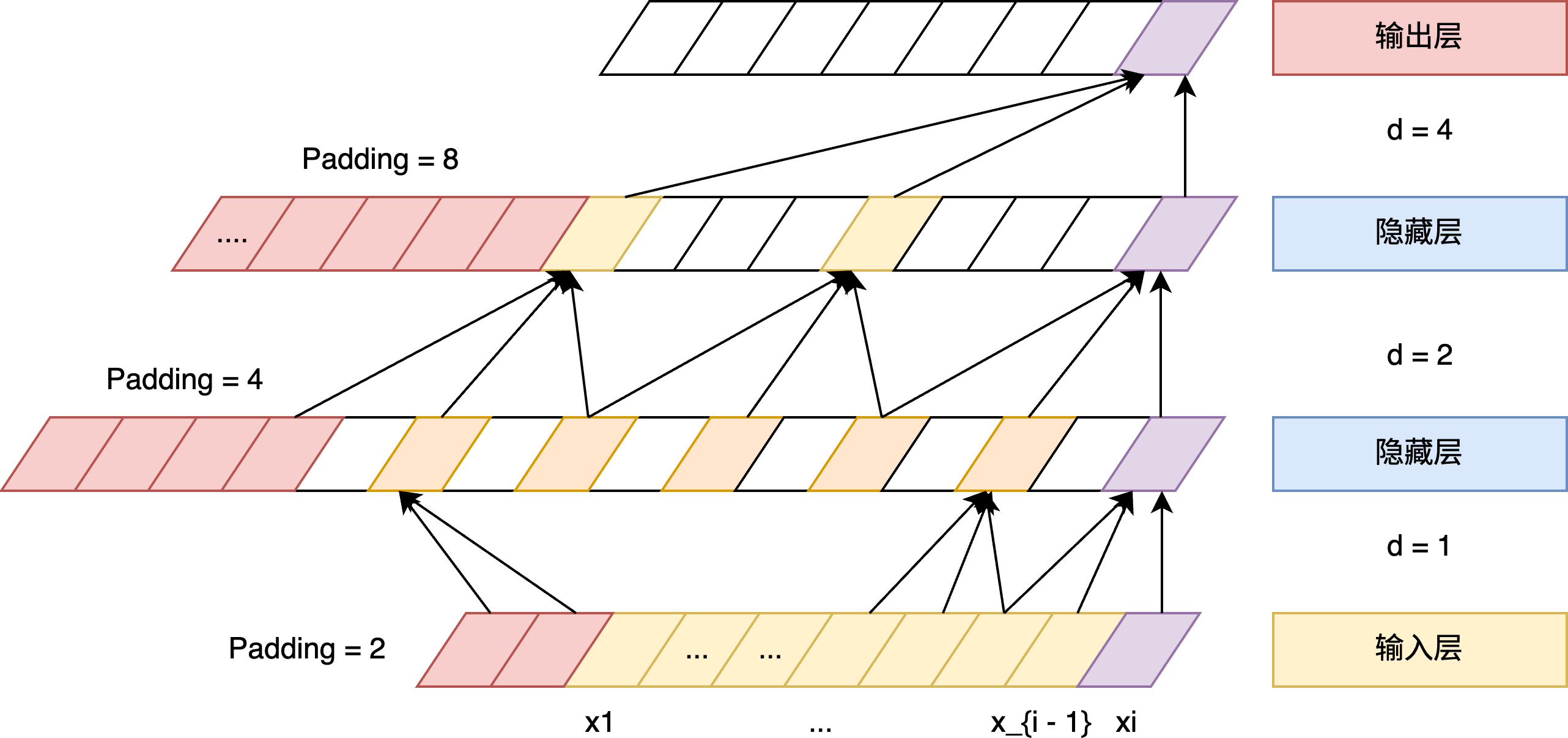


图 4.5 dilation=1 的膨胀因果卷积

图中，可以看到卷积核大小依然是 2，但是卷积核之间变得空洞了，每 2 个点采样一个作为输入；如果dilation=3 的话，那么可以想而知，这个卷积核中间会空的更大，每 3 个点采样一个作为输入。因为 dilation 变大了，所以相应的 padding 的数量从 1 变成了 2，所以为了保证输入输出的特征维度相同，padding的数值在卷积核是 2 的情况下等于 dalition 的数值。一般情况下，𝑝𝑎𝑑𝑑𝑖𝑛𝑔 =

(𝑘𝑒𝑟𝑛𝑒𝑙𝑠𝑖𝑧𝑒 − 1) × 𝑑𝑖𝑙𝑎𝑡𝑖𝑜𝑛，每个卷积核元素之间有𝑑𝑖𝑙𝑎𝑡𝑖𝑜𝑛 − 1 个空洞节点，所以空洞因果卷积的感受野范围大小为 (𝑘𝑒𝑟𝑛𝑒𝑙𝑠𝑖𝑧𝑒 − 1) × 𝑑𝑖𝑙𝑎𝑡𝑖𝑜𝑛 + 1。以输入中的第一个元素作为空洞因果卷积的最后一个元素，则它的左边需要 padding的个数为 (𝑘𝑒𝑟𝑛𝑒𝑙𝑠𝑖𝑧𝑒 − 1) × 𝑑𝑖𝑙𝑎𝑡𝑖𝑜𝑛。于是较为完备的 TCN 训练网络如下。

从图中可以看到，每一层 𝑡 时刻的值只依赖于上一层 𝑡, 𝑡 − 1, … 时刻的值，体现了因果卷积的原则。而每一层对上一层信息的提取，都是跳跃式的，且逐层 dilated rate 以 2 的指数增长，体现了空洞卷积的特性。同时由于采用了空洞卷积，因此每一个隐藏层都需要做 padding，padding 的大小为 𝑝𝑎𝑑𝑑𝑖𝑛𝑔 =



(𝑘𝑒𝑟𝑛𝑒𝑙𝑠𝑖𝑧𝑒 − 1)。

图 4.6 TCN 训练网络骨干

在 CNN 中能够提取不同的特征，网络层数越多，意味着能够提取到不同等级的特征，并且，越深的网络提取的特征越抽象，越具有深层次的信息。但是如果简单的增加深度，就会导致梯度消失或者梯度爆炸。对于这个问题可以使用权重参数初始化和正则化层，这样可以训练几十层的网络[[41](#_bookmark84)]。虽然解决了梯度消失的问题，但是网络退化出现了。高层次的神经网络空间虽然包含了低层次的网络空间，但是由于在训练中使用的是随机梯度下降策略，往往得到的是局部最优解，而不是全局最优解。

假设已经有了一个最优的网络结构，是 18 层。当设计网络结构时，并不知道具体多少层的网络拥有最优的网络结构，假设设计了 34 层的网络结构。那么多出来的 16 层其实是冗余的，希望训练网络的过程中，模型能够自己训练这 16层为恒等映射，也就是经过这 16 层时的输入与输出完全一样。但是往往模型很难将这 16 层恒等映射的参数学习正确，这样的网络一定比最优的 18 层网络表现差，这就是随着网络加深，模型退化的原因。因此解决网络退化的问题，就是解决如何让网络的冗余层产生恒等映射即深层网络等价于一个浅层网络。

为了解决网络退化问题，残差模块应运而生，同时终结图片识别大赛。下面是一个残差模块的结构。

通常情况下让某一层网络学习恒等映射函数𝐻(𝑥) = 𝑥 比较困难，但是如果把网络函数设置为 𝐻(𝑥) = 𝐹(𝑥) + 𝑥 就可以吧恒等映射函数转化为一个残差函数𝐹(𝑥) = 𝐻(𝑥) − 𝑥，只要 𝐹(𝑥) = 0，就构成了一个恒等映射 𝐻(𝑥) = 𝑥。图[4.7](#_bookmark24)中，

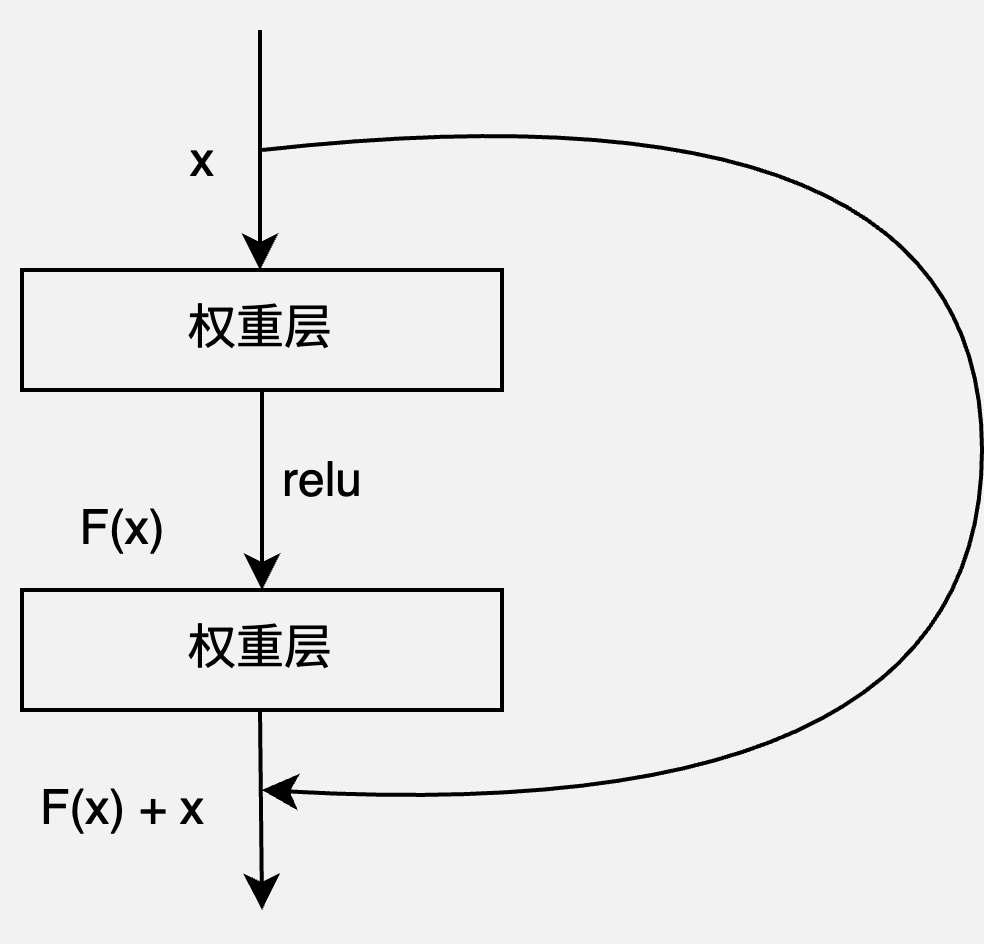


图 4.7 残差模块

iden mapping 被称为 shortcut 连接，residual mapping 即是 F（x）。残差的网络思想，如果网络已经到达最优，继续加深网络，residual mapping 将被push 为 0，只剩下 identity mapping，这样理论上网络一直处于最优状态了，网络的性能也就不会随着深度增加而降低了。

实验证明，残差模块往往需要两层以上，单单一层的残差模块并不能起到提升作用[[42](#_bookmark85)]。shortcut 连接有两种，如果是同等维度的映射则𝐹(𝑥) = 𝑊2𝜎(𝑊1𝑥 +

𝑏1) +𝑏2 𝐻(𝑥) = 𝐹(𝑥) +𝑥，如果维度不同则 𝐹(𝑥) = 𝑊2𝜎(𝑊1𝑥 +𝑏1) +𝑏2 𝐻(𝑥) =

𝐹(𝑥) + 𝑊𝑠𝑥。虽然残差模块刚开始是基于全连接层的表示，实际上残差模块可以用于卷积层。加法变为 channel 间的两个 feature map 逐个元素相加。下图[4.8](#_bookmark26)是时间卷积网络的残差模块。

于是整体的思路已经清晰，首先对时序数据集进行嵌入，嵌入使用指定大小的卷积核设置指定的膨胀因数，开始进行训练，进行归一向量化，和 ReLU 激活函数 Dropout 池化来防止梯度爆炸，最后用一个 Resnet 残差连接来避免梯度消失。

## 时间卷积网络与负载情况的结合

得到了综合负载评价指标和对应的时间就形成了一个常规的时间序列数据集，通过时间卷积网络深度学习即可较好的学习到时间序列内蕴藏的信息，通过蕴藏的信息预测未来的综合负载情况，返回给 Nginx 集群内的负载均衡器，其以此来分配不同的不同的任务给空闲的服务器。

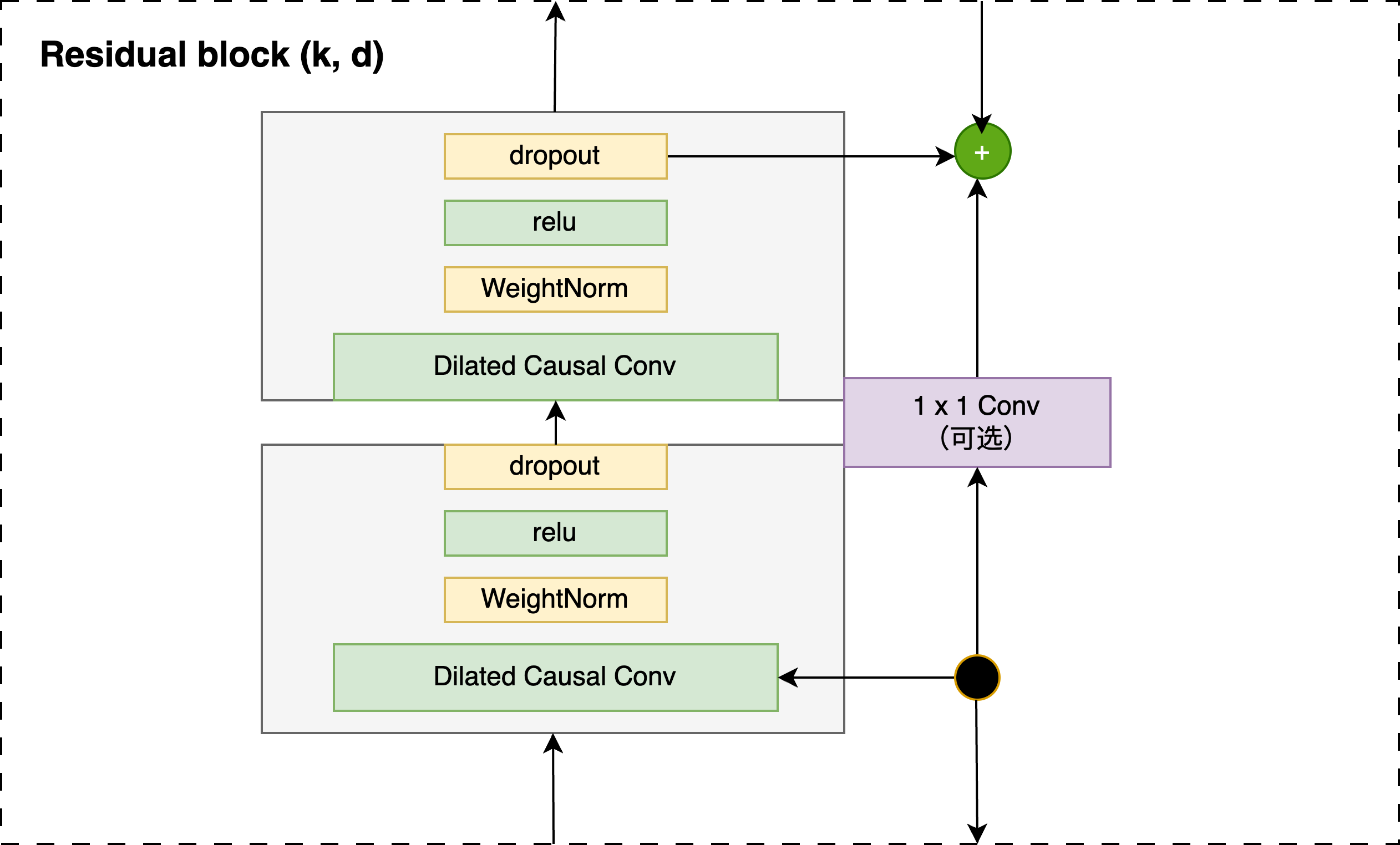


图 4.8 TCN 的残差模块设计

综合相关文献，对比了常规 RNN、LSTM、GRU 与 TCN 等循环神经网络在多种典型序列模型预测问题的性能表现上，结果表明 TCN 通常可以获得更好的预测精度，可以作为时间序列预测建模的有效手段[[43](#_bookmark86)] [[44](#_bookmark87)]。

1. 准备数据通过使用监控脚本，周期性的收集 CPU、内存、磁盘 I/O 和网络带宽的利用率，并把这些数据保存到一个 CSV 文件中。由于作者本身的技能问题，不能使用高效率的监控程序，使用了 bash 脚本来监控这些参数。

1



!/bin/bash

设置本地 CSV 文件路径

OUTPUT\_FILE="/path/to/your/output\_file.csv"

写入 CSV 文件的标题行

echo "Timestamp,CPU\_Usage,Mem\_Usage,Disk\_Read,Disk\_Write,Network\_In,

Network\_Out" > $OUTPUT\_FILE

设置网络接口名称

NETWORK\_INTERFACE="eth0"

捕获信号，以便优雅地退出

trap "echo 'Script terminated'; exit" SIGHUP SIGINT SIGTERM

开始收集数据

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16



while true; do

获取 CPU 使用率

CPU\_USAGE=$(top -bn1 | grep "Cpu(s)" | awk '{print $2 + $4}')

获取内存使用率 (使用的内存比例)

MEM\_USAGE=$(free -m | awk 'NR==2{printf "%.2f", $3\*100/$2 }')

获取磁盘读写数据 (需要 sysstat 包安装)

DISK\_IO=$(iostat -dx | awk 'NR>3 {print $6,$7}' | awk '{read+=$1; write+=$2} END {print read,write}')

获取网络带宽使用数据 (需要 ifstat 包安装)

NETWORK=$(ifstat -i $NETWORK\_INTERFACE 1 1 | awk 'NR==3 {print $6,

$8}')

获取当前时间戳

TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')

写入数据到 CSV

echo "$TIMESTAMP,$CPU\_USAGE,$MEM\_USAGE,$DISK\_IO,$NETWORK" >>

$OUTPUT\_FILE

设置延迟时间，例如每隔 1 分钟收集一次数据

sleep 60 done

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

这个脚本会无限循环到直到我手动暂停，可以设置循环次数，也可以设置循环大周期，比如使用 nohup 来进行循环次数的限制，使用每日，每周，每月作为一个大周期获得较为长远的数据，来进行更加准确的预测。通过主成分空间分析得到的原始数据集，然后分为按照 9 ∶ 1 分为训练集和测试集。

1. TCN 参数设置

由膨胀因果卷积的原理可知，TCN 模型中的扩大因子𝑑 和卷积核大小𝐾 是决定 TCN 模型预测性能的主要参数。此参数的选择目前尚无理论指导，一般均需通过实验对比取得[[45](#_bookmark88)]。由于实验数据以及实验环境的限制，无法通过实验来确定当前不同配置的 TCN 模型的 𝑀𝐴𝐸 值和 𝑅𝑀𝑆𝐸 值来确定最佳参数。但是可以列举几个比较通用的模型参数，其中，扩大因子 d 分别取值为 2、4 、8 和 16

；卷积核 K 分别取为 1×3、1×5 和 1×7 ，两组参数共构成 12 种组合形式，如下表所示。

表 4.3: TCN 模型参数组合

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 𝑑 | 𝐾 | 序号 | 𝑑 | 𝐾 |
| 1 | 2 | 1x3 | 7 | 8 | 1x3 |
| 2 | 2 | 1x5 | 8 | 8 | 1x5 |
| 3 | 2 | 1x7 | 9 | 8 | 1x7 |
| 4 | 4 | 1x3 | 10 | 16 | 1x3 |
| 5 | 4 | 1x5 | 11 | 16 | 1x5 |
| 6 | 4 | 1x7 | 12 | 16 | 1x7 |

通过使用不同参数的 TCN 模型可以得到 MAE（平均绝对误差）即观测值与预测值绝对差的平均值、RMSE（均方根误差) 即观测值与预测值差的平方的平均值的平方根。来确定最佳参数。通过指定训练参数 epoch 以及 lr 来获得预测值，在通过与观测值的比较即可选择到较好的模型参数。

1. 特征学习训练以及优化网络模型

指定时间卷积的参数扩大因子 𝑑 和卷积核尺寸 𝐾, 以及指定的 epoch 和 lr开始训练；使用均方根误差作为目标函数计算预估值和真实值的误差，并利用 Adam 优化算法更新网络中的参数[[46](#_bookmark89)]。下面是重要的训练代码。

1



定义均方根误差为损失函数 class RMSELoss(nn.Module): def init (self):

super(). init ()

def forward(self, yhat, y):

return torch.sqrt(torch.mean((yhat - y) \*\* 2))

初始化损失函数

loss\_function = RMSELoss()

开始进行训练

for epoch in range(num\_epochs): for batch in train\_loader:

inputs, targets = batch optimizer.zero\_grad() 清除过往梯度 outputs = model(inputs) 获取模型预测结果

loss = loss\_function(outputs, targets) 计算损失 loss.backward() 反向传播，计算梯度 optimizer.step() 使用 Adam 更新模型权重

2

3

4

5

6

7

8

9



10

11

12

13

14

15

16

17

18

19

1. 输出预估值

在 PyTorch 中，可以直接调用模型对象并直接将输入数据传递给它以获得预测值。

1



切换模型到评估模式

model.eval()

不计算梯度来加速计算和减少内存消耗

with torch.no\_grad():

for inputs in test\_loader:

输入新数据集

predictions = model(inputs)

输出预测值

print(predictions)

2

3

4

5

6

7

8

9

10

在上述的代码中，prediction 变量包含了模型基于 inputs 给出的预测结果。如果是预测未来一个时间步，prediction 可能只是一个数字。如果是预测未来多个时间步，prediction 可能是一个Numpy 数组或者PyTorch 张量，其中包含一系列预测值。如果输入的一条数据，那么输出的就是单一的预测值，如果输入的是一个序列，那么输出的就是包含多个值的序列。可以通过预测后的序列值观察综合负载情况，然后进行判断。

## 本章总结

本章首先探讨了如何获取到剩余性能的数据，并根据科学的主成分分析法来选取合适的权重参数，用合适的权重参数得到了能较好体现负载情况的综合负载指标数据集。

接着讨论了时间卷积网络与其他循环神经网络的不同之处，得到了其在时间序列预测性能的特征和具体原理和具体过程。最后将综合负载指标已经对应的时间戳形成对应的时间序列数据集。通过设置 epoch 和 lr 来暂时确定MAE 和 RMSE 的走势情况，以获取训练的最佳参数。设置了最佳参数后可以进行更大的 epoch 来实现更好的预测性能。总体描述使用时间卷积网络来进行训练的步骤，分化训练集和测试集，然后根据上面得到的最佳参数设置，进行网络训练拟合，最后输出我们想要的预估值。

# 预测模型与传统负载均衡的结合

回顾第二章的讨论，我们已经详细分析了传统负载均衡策略的工作原理，虽然这些传统策略在多种场景下都能提供稳定的性能，但在面对动态变化和未预见负载的情况时，仍然显得力不从心。本章旨在探讨一种新的解决方案，即预测模型与传统负载均衡的结合，以期望克服现有方法的不足。我们将首先介绍结合方案的设计理念与实施策略，随后通过实验验证其有效性，并对模型的表现进行深入分析。

## 传统负载均衡算法的选取

在第二章的讨论中，发现了加权轮询算法原理简单，而且配置比较简便，权重可以很好的反映出集群内服务器节点的性能和负载情况，是一种非常经典的负载均衡算法。本文选择以加权轮询算法为基础，分析其原理并进行改进，实现一种检测服务器节点负载状况，并以此为根据动态地调整各个节点权重以合理分配请求任务目的的算法。

相比于默认的轮询算法，加权轮询算法则考虑了不同性能特点的服务器节点，性能好的服务器则设置较大的权值，性能不好的服务器则设置较小的权值。当任务入队列时负载均衡器则可以按照设置好的权值比例合理的悬着服务器执行任务请求。

通过对 Nginx 源码的研究，在 ngx\_http\_upstream\_round\_round\_robin.c 这个文件下 ngx\_http\_upstream\_get\_round\_robin\_peer 函数是对加权轮询算法的实现。在具体分配任务的环节，为了记录上游集群节点服务器的权值公设有四个变量，依次是 weight、effective\_weight、total 和 current\_weight。

表 5.1: 加权轮询算法变量及描述

|  |  |
| --- | --- |
| 变量 | 描述 |
| weight | 初始权值，固定不变 |
| effective\_weight | 发生错误，权值减小 |

Continued on next page

表 5.1: 加权轮询算法变量及描述 (Continued)

|  |  |
| --- | --- |
| total | 集群服务器权值总和 |
| current\_weight | 实时权值，初始为零 |

在一次任务中，负载均衡器勉励上游服务器节点的current\_weight 加上起对应的有效权值 effective\_weight，负载均衡器将会选择 current\_weight 最大的服务器节点为本轮最佳服务器。当该服务器正在进行任务时，负载均衡器会降低其 current\_weight。Nginx 定义了两个事件状态，NGX\_OK 表示成功选择最佳服务器，NGX\_BUSY 表示服务器选择失败或者通信发生错误。加权轮询算法详细流程如下图[5.1](#_bookmark30)所示。

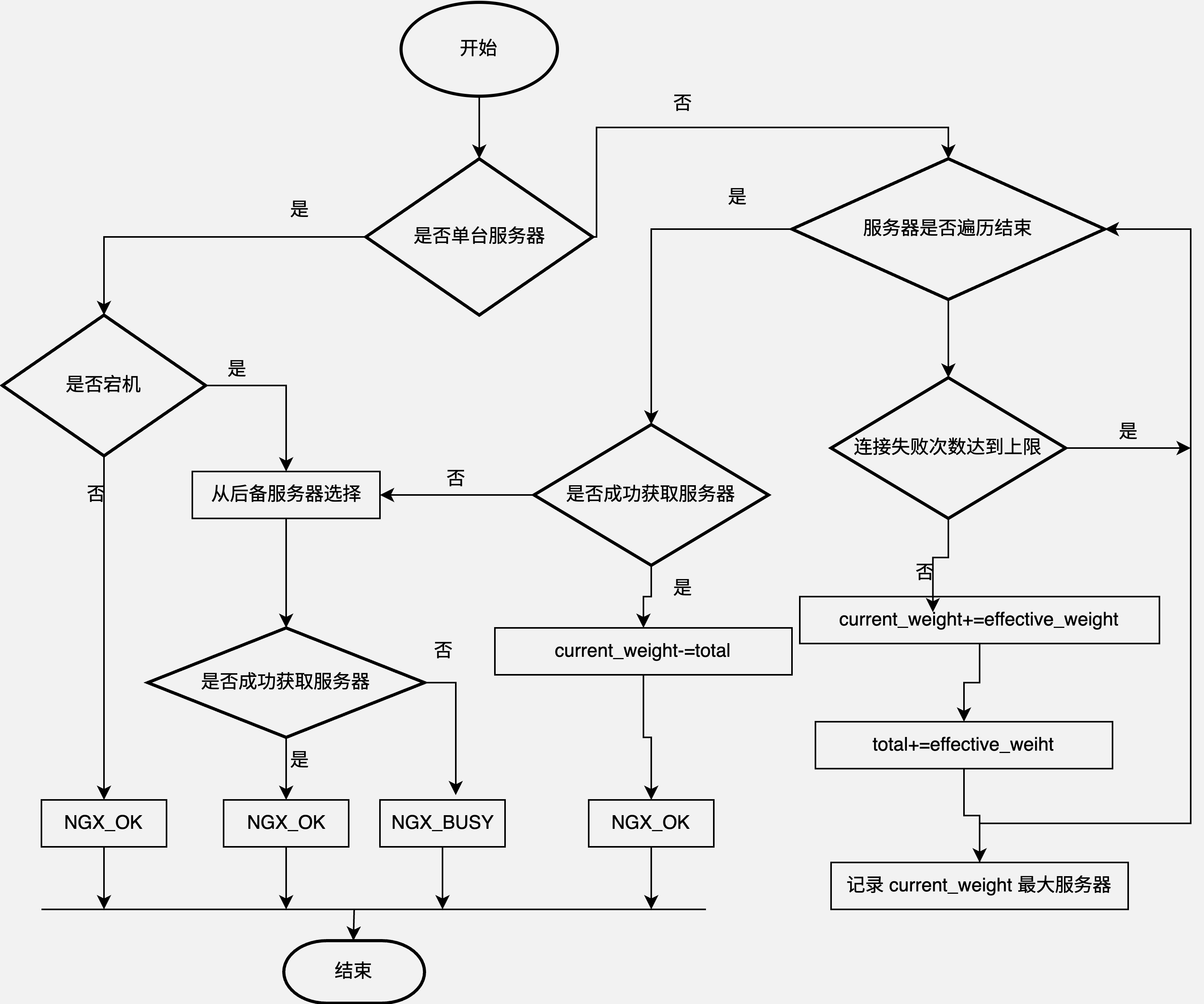


图 5.1 Nginx 加权轮询算法流程图

加权轮询算法，虽然考虑了上游服务器节点的不同性能差异，执行效率较高，节点选择频次相对平滑，配置简单，但是在具体任务过程中，有些任务比较

繁忙，有些任务比较轻松，这种静态的权重调节无法根据真实负载情况进行动态调整。为了解决这个问题，诞生了动态权值的加权轮询算法，基于手机各个后台服务器节点工作时的 CPU 利用率、内存利用率、网络性能和磁盘 IO 等性能情况，动态的调整后端服务器节点权重[[10](#_bookmark53)]。该算法在高并发的场景下，响应时间和实际并发方面表现更好。

常规的加权轮询算法在某些特殊的权重下，加权轮询调度会生成不均匀的实例序列，这种不平滑的负载可能会使某些实例出现瞬时高负载的现象，导致系统存在宕机的风险。为了解决这个调度缺陷，就提出了平滑加权轮询调度算法。如图[5.2](#_bookmark31)所示为 Nginx 中该算法的流程图。

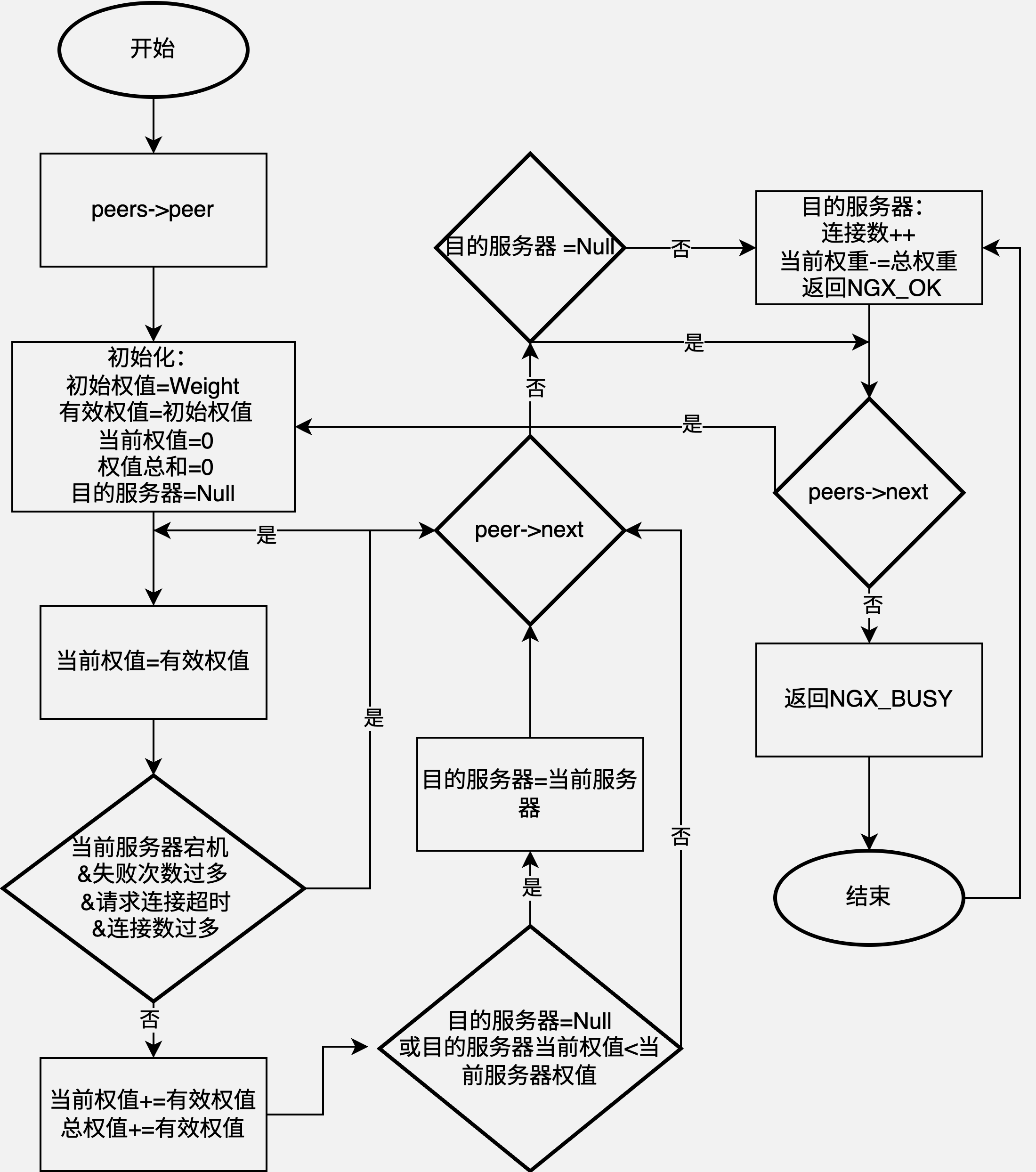


图 5.2 Nginx 内置平滑轮询算法流程图

为了证明平滑性，只要证明不要一直都是连续选择那一个节点即可。假设

总权重为S，假如某个节点i 连续选择了t（𝑡 < 𝑥𝑖）次，只要存在下一次选择的不是节点 i，即可证明是平滑的。

假设 𝑡 = 𝑥𝑖 − 1，此时第 i 个结点的当前权重为 𝑤𝑖 = 𝑡𝑥𝑖 − 𝑡𝑆 = (𝑥𝑖 − 1)

𝑥𝑖 − (𝑥𝑖 − 1)𝑆。即证明下一轮的第 1 步执行完的值 𝑤𝑖 + 𝑥𝑖 不是最大。

𝑤𝑖 +𝑥𝑖 => (𝑥𝑖 −1)𝑥𝑖 −(𝑥𝑖 −1)𝑆+𝑥𝑖 => 𝑥2−𝑥𝑖𝑆+𝑆 => (𝑥𝑖 −1)(𝑥𝑖 −𝑆)+𝑥𝑖 (5.1)

𝑖

因为𝑥𝑖 恒小于S，所以𝑥𝑖 − 𝑆 <= −1。所以上面：(𝑥𝑖 − 1)(𝑥𝑖 − 𝑆) + 𝑥𝑖 <= (𝑥𝑖 − 1) − 1 + 𝑥𝑖 = −𝑥𝑖 + 1 + 𝑥𝑖 = 1。所以第 t 轮后，再执行完第 1 步的值

𝑤𝑖 + 𝑥𝑖 <= 1。如果这 t 轮刚好是最开始的 t 轮，则必定存在另一个结点 j 的值为 𝑥𝑗𝑡，所以有 𝑤𝑖 + 𝑥𝑖 <= 1 < 1𝑡 < 𝑥𝑗𝑡。所以下一轮肯定不会选中 i。

平滑加权轮询算法改善了加权轮询算法调度的缺陷，即调度序列分散的不均匀，避免了实例负载突然加重的可能，但是仍然不能动态感知每个实例的负载。若由于实例权重配置不合理，或者一些其他原因加重系统负载的情况，平滑加权轮询都无法实现每个实例的负载均衡，这时就需要有状态的调度算法来完成。由此本文在该算法基础上进行了相关调整，首先采集后台各个节点指标数据，包括 CPU、内存、网络带宽以及磁盘 I/O 指标数据，并对采集到的数据惊醒相关处理，通过这些数据来确定不同节点的初始权重，反应不同节点的初始性能差异。并持续周期性的收集综合负载指标，使用训练好的模型预测剩余性能，并根据该性能调整节点权重。同时为了防止初期无法得到有效的服务器负载均衡后的有效数据，将会使用默认的平滑加权算法来得到初期数据，经过一定周期，将收集到的负载均衡数据整理并使用预测模型进行预测，并根据预测的数据进行初始权重的调整和优化。

## 改进的负载均衡算法的提出

本次改进后的算法主要分为四个模块：服务器负载信息收集模块，服务器信息处理模块，Nginx 动态加权轮询模块，周期预测模块。四个模块协同完成改进的动态加权轮询算法的功能。其中负载均衡负载信息收集模块一直运转在各个集群节点中，负载周期性的获取各个节点的负载数据并存入 Memcached 中，包含访问次数，CPU、内存、网络带宽以及磁盘 IO 等利用率。服务器信息处理模块在 Nginx 负载均衡的服务器上，从 Memcached 中获取各个节点的负载数据，并根据获取到的负载数据得到各个节点的综合负载情况, 根据设定的阈值 M 和 N

相比较，判断是否负载正常。如果节点负载过高或者过低，负载均衡服务器会调整该节点的权重，并将调整后的权重保存在 Memcached 服务中。另外，在周期预测性模块中，会周期性的以网站访问量以及综合负载指标来进行预测，调整阈值 M 和 N。Nginx 动态加权轮询模块从 Memcached 中获取各个节点的最新权重信息，按照改进后的算法选取某一节点完成任务请求。系统总体结构如下图[5.3](#_bookmark33)所示。

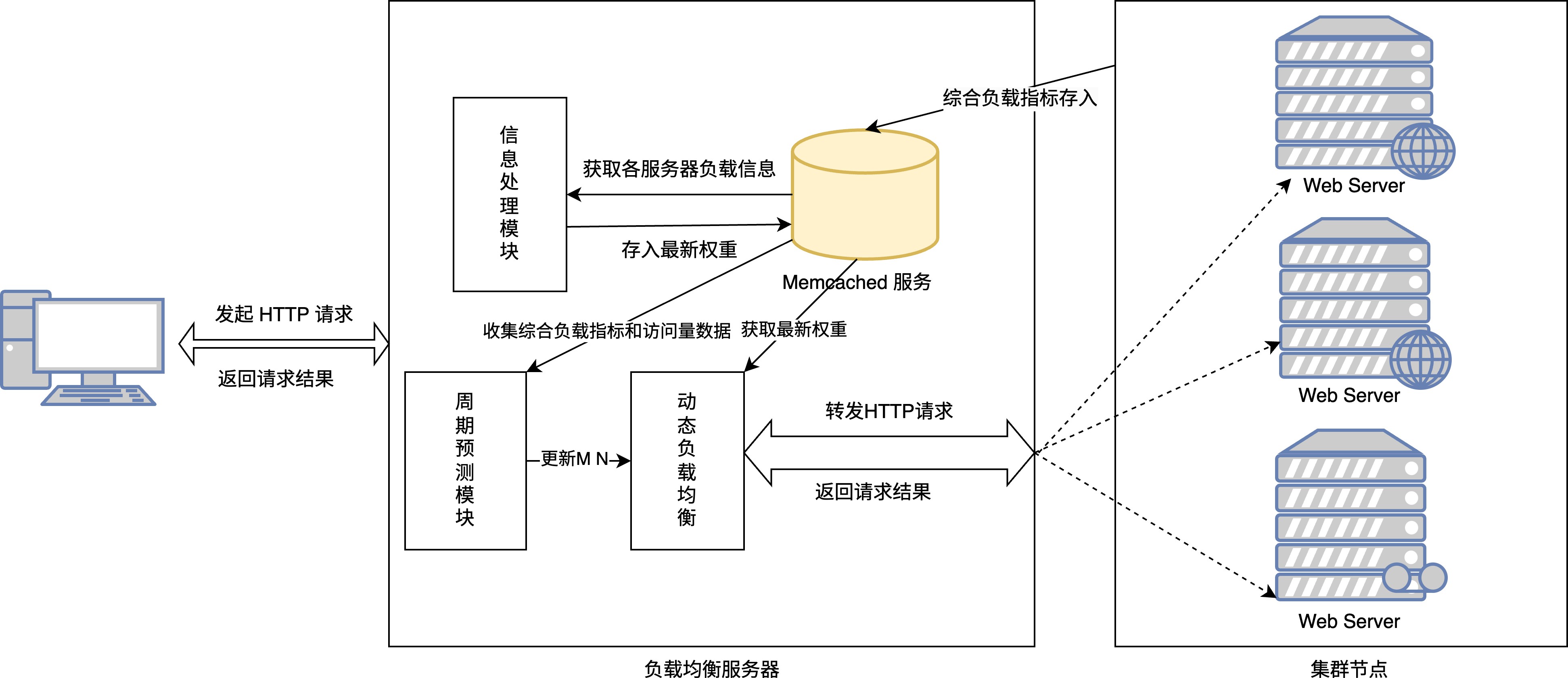


图 5.3 总体结构图

当client 用户发起 HTTP 请求任务到Nginx 服务器，由 Nginx 服务器根据当前使用的负载均衡策略选取集群中某一个节点，把 client 发来的 HTTP 请求任务转发给该节点，该节点收到任务后进行处理，把数据发给 Nginx 服务器，Nginx将收到的数据发回给客户端，客户端收到响应数据包，对数据进行处理并显示在网页上。改进后的平滑动态加权轮询算法流程如下图[5.4](#_bookmark34)所示。

改进后的算法分为两个阶段：没有预测指标阶段和已有预测负载指标阶段。最开始没有合适的数据通过周期性预测处理模块对当前服务的相关访问量和服务器负载情况进行分析，则可以通过常规的平滑加权轮询算法完成初期任务。在第一个阶段中通过服务器负载信息收集模块对后端每一个节点此时的 CPU、IO、内存、网络带宽的利用率进行采集。再将每个节点相关的负载数据交给服务器信息处理模块和周期性预测模块做相关数学分析，并通过公式（4.4）计算出各个节点出各个节点的总体负载情况，根据确定的阈值确定该节点是不是均衡，如果不均衡则更新该节点的权重数据。等却动的阈值确定该节点是不是均衡，如果不均衡则更新某一节点的权重。当权重数据分析完毕后，由 Nginx 平滑加权轮

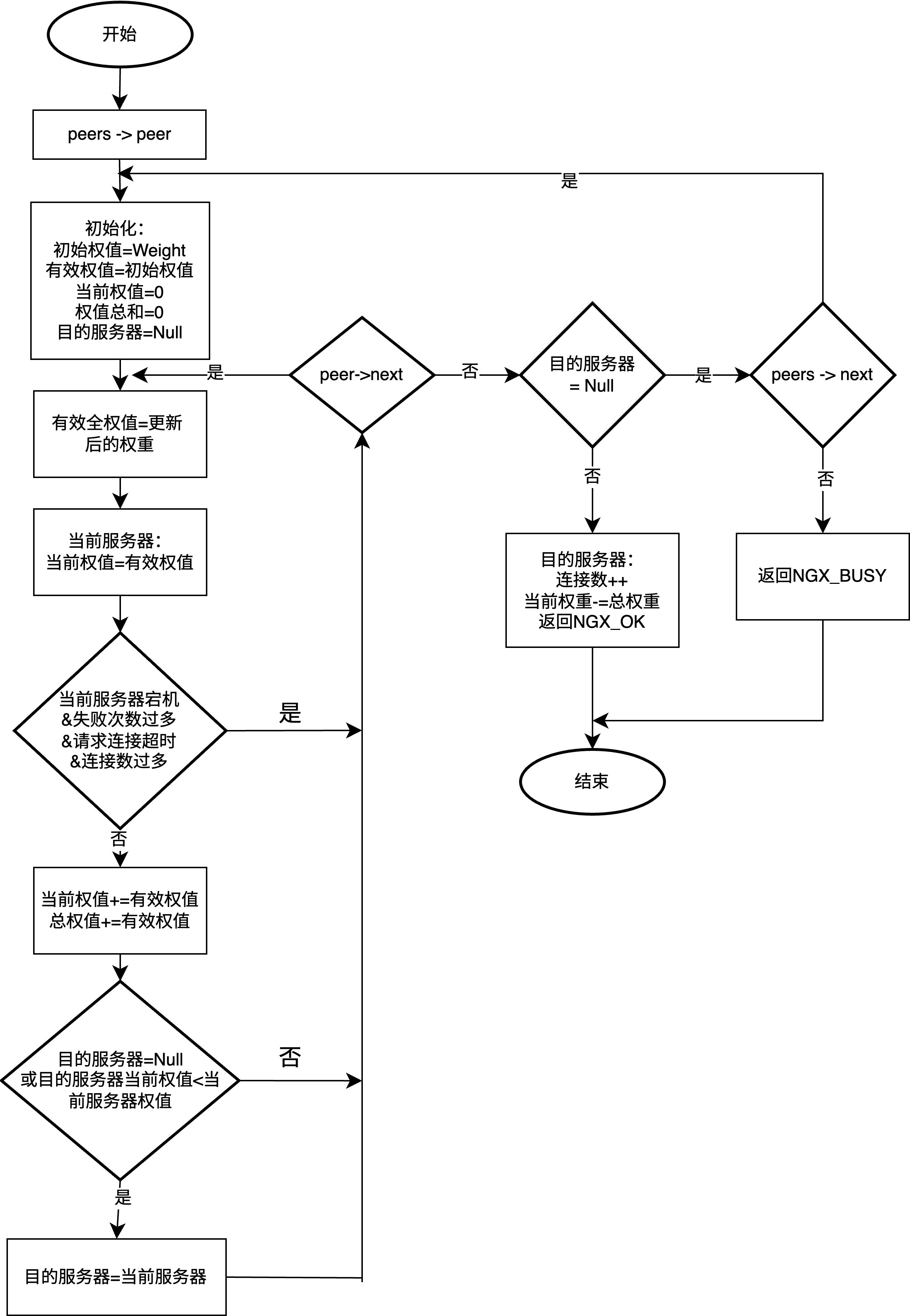


图 5.4 改进后的平滑加权轮询算法流程图

询模块获取到最新的节点权重信息。最后进行服务器配置以及权重初始化，该项初始化完成后，由 nginx\_http\_upstream\_get\_dynlb\_select\_peer 函数选择一台所有上游可用服务器中 current\_weight 最大的服务器对请求进行处理，如果所有节点都不符合条件，则选择 Nginx 自带的轮询算法。当请求都处理完毕后调用 ngx\_http\_upstream\_free\_dynlb\_select\_peer 释放系统所占用的资源，提高算法的效率。第二个阶段则与第一个阶段类似，不过阈值和初始权重的调整会有所不同。

## 负载均衡算法的优化

通过第三章、第四章预测访问量的算法和剩余性能深度学习卷积神经网络的研究，有效的证明了算法的有效性和深度学习网络的优秀性能。通过对访问量的预测能够有效的在根据不同时间内分配合适的任务量，优化集群内的节点效率，有效降低集群的资金输入。通过时间卷积网络则可以很好的提取到不同时间段蕴藏的信息，同时得到了节点未来的负载情况，使得负载均衡器有效的分配合适的任务给合适的集群节点。但是对于初期没有负载数据和访问量数据的问题，也需要值得重视，本节既是为了研究初期负载算法的优化。

第四章中通过主成分分析法分析了在已有数据情况下的公式（1.1）的权重参数，但是在没有数据之前仍然需要通过一定的分析得出一个较为合理的权重和综合负载情况的阈值。通过对不同论文的研究[[47](#_bookmark90)]，本文选择了层次分析法来进行初期的权重参数的计算。

首先，需要构建出层次分析模型。将综合负载指标作为目标层，CPU、内存、网络带宽以及磁盘 IO 利用率指标作为准则层，层次分析模型结构如下图[5.5](#_bookmark36)所示。

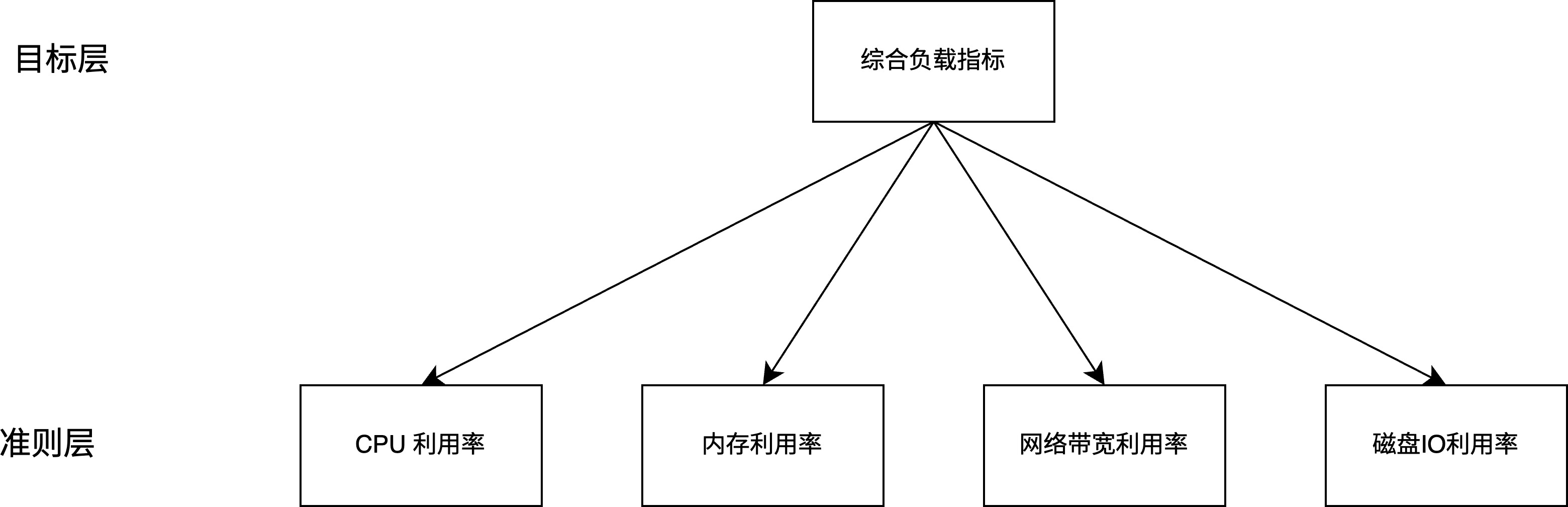


图 5.5 综合负载层次分析模型

采用“1-9”标度法，结合服务器测试情况构建判断矩阵。根据论文得知，

CPU、内存利用率较高，而网络带宽与磁盘 IO 利用率相对较低[[25](#_bookmark68)]。服务器对资源消耗标度表如下。

表 5.2: 服务器对资源消耗情况

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Q | CPU | Mem | Net | IO |
| CPU | 1 | 1 | 3 | 3 |
| Mem | 1 | 1 | 3 | 3 |
| Net | 1  3 | 1  3 | 1 | 1 |
| IO | 1  3 | 1  3 | 1 | 1 |

于是，得到了判断矩阵：

𝖥 3⎤

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 3 |
| 1 | 1 | 3 |

⎢ 3⎥

𝑄 = ⎢ ⎥

(5.2)

⎢1/3 1/3 1 1⎥

⎢ ⎥

⎣1/3 1/3 1 1⎦

按照层次分析法的要求，得到了判断矩阵，通过合积法计算其近似特征向量。首先将矩阵（5.2）按照公式（5.3），再按列相加再做归一化，归一化的元素按行相加得到的结果处理后如下表所示。

𝑞𝑖𝑗

𝑞𝑖𝑗 =

𝑛

𝑖=1 𝑖𝑗

∑ 𝑞

(𝑖, 𝑗 = 1, 2, … , 𝑛) (5.3)

表 5.3: 对比表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Q | CPU | Mem | Net | IO | ∑ |
| CPU | 0.375 | 0.375 | 0.375 | 0.375 | 1.5 |
| Mem | 0.375 | 0.375 | 0.375 | 0.375 | 1.5 |
| Net | 0.125 | 0.125 | 0.125 | 0.125 | 0.5 |
| IO | 0.125 | 0.125 | 0.125 | 0.125 | 0.5 |

对上表自左向右最后一列元素进行归一化处理之后，即可得到特征向量

𝑀′ = (𝑀1, 𝑀2, 𝑀3, … , 𝑀𝑛)T 的近似特征向量 𝑀′。

𝑀′ = (0.375, 0.375, 0.125, 0.125)

于是初始阶段的权重已经确定，则初期阶段综合负载决策函数为：

𝑋 = 0.375 × (𝐶𝑃𝑈) + 0.375 × (𝑀𝑒𝑚) + 0.125 × (𝑁𝑒𝑡) + 0.125 × (𝐼𝑂) (5.4)

当节点的资源使用率达到了系统的瓶颈时，节点处理任务请求的效率就会明显下降，那么负载均衡器分配给节点的权重值就会减少。通过表（2.2）可知，当节点的资源利用率评价为坏的时候，则可以认为此时节点负载过高。在初期阶段，此时总体剩余利用率可以通过公式（5.4）计算得到，在使用了周期性预测模块阶段则通过（4.4）计算得到，将该值设为M，M 是判断节点负载是否平衡的条件，当 𝑇 ≥ 𝑀 时，判断节点为高负载状态，则可以取阈值 𝑁 = 𝐿 = 1 − 𝐻/2。

由于信息采集周期 T 对与负载信息收集模块和周期性预测模块有着很大的影响，对于负载信息采集模块来说，周期太短，可能导致信息收集太过于频繁而耗费过多的系统资源，周期太长，则会导致负载均衡服务器节点权重参数更新不及时，权重滞后。周期不能简单的通过定性进行分析得到，需要进行一定的实验对比，本文推荐使用 siege 来进行测试。

Siege 是一款开源的压力测试工具，设计用于评估 WEB 应用在压力下的承受能力。可以根据配置对一个 WEB 站点进行多用户的并发访问，记录每个用户所有请求过程的相应时间，并在一定数量的并发访问下重复进行。对于不同业务的要求，可以选择不同的并发量，选择记录在指定文本文件内的链接访问不同类型的任务来模拟真实的负载情况。下面是推荐的并发量和任务类型。

表 5.4: 企业与任务类型

|  |  |
| --- | --- |
| 企业类型 并发量 | 任务 |
| 小型企业 500-1000  中型企业 1000-5000  大型企业 10000-100000  超大型企业 100000- | 文本图片音频  视频 |

siege 的测试命令如下。

1

siege -c 并发量 -r 重复次数 -f url.txt

Listing 5.1 siege 的测试命令

其中 url.txt 则是指定的任务类型，具体类型如下。

1

<http://162.168.66.166/xx.jpg> <http://162.168.66.166/xx.html> <http://162.168.66.166/06.mpv> <http://162.168.66.166/06.mp4>

2

3

4

Listing 5.2 url.txt 请求任务类型

通过 siege 的测试可以得到不同周期 𝑇 下 Nginx 的负载程度，通过对 Nginx负载程度的定量分析，可以得到当前业务下最适合的周期 𝑇。同时在周期预测阶段则可以选择每小时，每天，每月，每年作为整个周期进行拟合和预测。

在改进的动态负载均衡策略中，有两个重要部分，其一是使用合适的周期来收集各个服务器节点的剩余性能，并保存在 Memcached 服务中。Memcached是一个分布式的告诉缓存系统，对于频繁获取负载信息的操作很有利，保证了收集数据的实时性，控制了资源的消耗。其二，如果负载均衡器判断了集群某些节点负载不均衡后，如何调整各个节点的权重。

在本算法中，使用阻塞控制的思想来对权重进行调整。拥塞控制是作用于网络的，它是防止过多的数据注入到网络中，避免出现网络负载过大的情况；常用的方法就是：（1 ）慢开始、拥塞避免（2 ）快重传、快恢复。本改进算法使用了，快重传，和快恢复算法。常见的改进的动态负载均衡方法使用的是 TCP Tahoe 算法，但是现在已经遭到抛弃。所以本算法使用的是 TCP Reno 算法。

快重传要求接收方在收到一个失序的报文段后就立即发出重复确认，而不要等到自己发送数据时捎带确认。快重传算法规定，发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段，而不必继续等待设置的重传计时器时间到期。与快重传算法配套的还有快恢复算法，当发送方连续收到三个重复确认时，就执行“乘法减小”算法，把ssthresh 门限减半（为了预防网络发生拥塞）。但是接下去并不执行慢开始算法，考虑到如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方现在认为网络可能没有出现拥塞。所以此时不执行慢开始算法，而是将 cwnd 设置为 ssthresh 减半后的值，然后执行拥塞避免算法，使 cwnd 缓慢增大。下图是具体的拥塞控制窗口和传输轮次的关

系表。下图[5.6](#_bookmark37)是改进的算法中，TCP Reno 的变化。

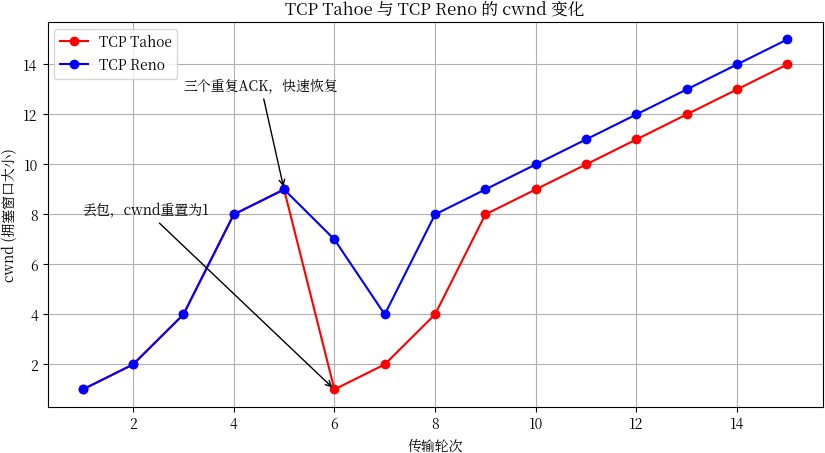


图 5.6 TCP Reno 的 cwnd 变化

TCP Reno 在 Tahoe 的基础上增加了快速恢复机制即快重传机制。在 Reno中，当收到三个重复的 ACK 时（表示一个单一的数据包丢失），它不会像 Tahoe那样直接进入慢启动阶段，而是进入快速恢复阶段。在这个阶段，它将阈值设为当前 cwnd 的一半，并将 cwnd 设置为新阈值加 3 个段的大小，然后进行拥塞避免算法。如果又出现超时，则会和 Tahoe 一样，将 cwnd 设为 1，重新开始慢启动过程。Reno 的这种机制可以在某些丢包情况下避免 cwnd 的大幅度减少，使得网络能够更快地恢复到有效的传输状态。从而使的负载均衡节点更有效的调整节点的权重

根据这种思想，本文想去了服务器节点的综合负载指标这一指标作为判断负载时的条件，并设置一个阈值 𝑁 作为改进的动态轮询算法的权重调整门限，值

𝑀 判断节点是否负载均衡的条件。但是，初期阶段根据服务器信息处理模块收集到的信息来进行判断具有滞后性。所以在预测阶段能使用深度学习算法来预测节点综合负载指标来作为判断标准，获取所有节点下一时间段的综合负载情况，若判断存在服务器节点下一时间会变成高负载状态，则将下一时刻判断为负载不均衡的节点进行权重调整，调整方式为 𝑊𝑗 = 1000 × 𝑈𝑗/ ∑ 𝑈𝑖，∑ 𝑈𝑖 表示下一时刻所有负载不均衡节点综合负载指标的和，𝑈𝑗 为第 𝑗 个且被判断为下一时刻负载不均衡的节点。并将节点的权重信息存入 Memcached 服务，对于低负

载节点权重，将其调整为 𝑊𝐶𝑗 = 𝑊𝑗 − 2𝑛，𝑛 为权重调整次数，直至权重变为初始权重。

持续观察高负载节点节点负载状况，在该服务器节点的综合指标 𝑋 < 𝑁 当前权重值小于初始权重时，使该节点的权重为 𝑊𝐶𝑗 = 𝑊𝑗 + 2𝑛，𝑛 是调整次数，若此过程中当前权重增长值等于初始权重大小，则调制权重增长。单个服务器节点的综合负载指标 𝑋 ≥ 𝑁 且当前权重小于初始权重时，停止次节点的指数增长，使其呈现线性增长。𝑊𝐶𝑗 = 𝑊𝑗 + 2𝑘，𝑘 为停止指数增长此刻权重调整的次数。若在此过程中权重带到初始权重值，则停止权重增长，或者该节点再次被判断为下一时刻处于高负载状态，则再将权重进行调整。在节点的权重信息调整过程中，持续更新其权重值，并录入至 Memcached 中。其高负载节点权重调整流程图如图[5.7](#_bookmark38)所示。

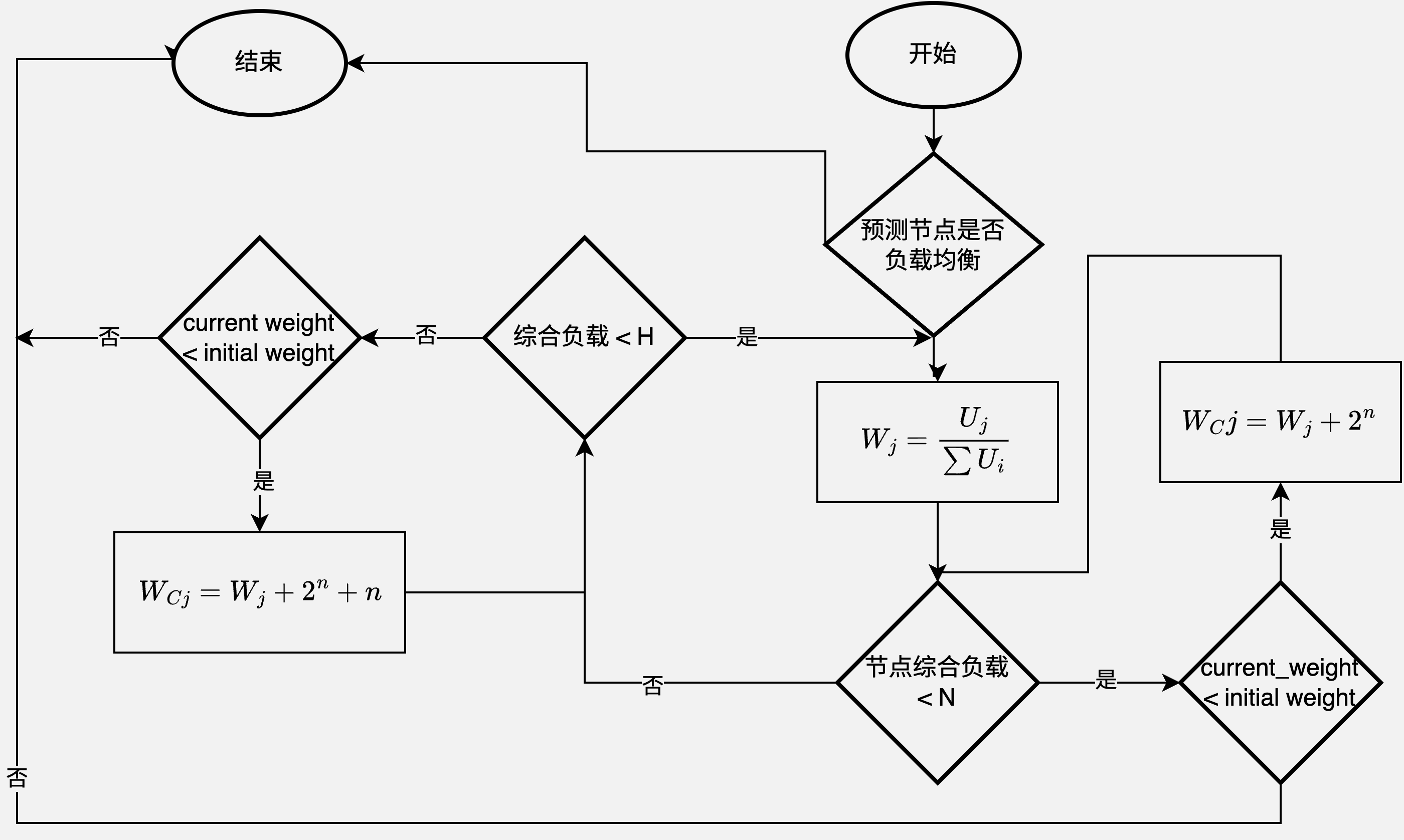


图 5.7 基于拥塞控制思想的权重调整流程

## 本章总结

在本章中，通过研究常规的动态负载均衡方案的不足，提出了与预测模型相结合的动态负载均衡方案。其中包含两个阶段，初期阶段和预测阶段，很好的解决了改进算法在没有数据时的负载均衡方案，提出了收集周期 𝑇 和初期阶段的初始权重，和调整权重的阈值𝑁，同时借鉴了 TCP Reno 方案优化了负载均衡器调整权重时的阻塞情况。

# 总结与展望

## 本文工作总结

现在互联网发展愈来愈迅速，服务器集权架构越来越复杂，用户对各种服务的要求也越来越高，不仅各种中小型企业甚至大型企业也很重视对于复杂均衡的研究，以达到更好的性价比。本文在背景下，对现在常用的负载均衡策略和基本的时间序列预测模型进行了较为充分的理解和研究，针对了具体的访问量预测方面和 Nginx 内置的平滑加权轮询算法进行了与神经网络的深度结合，很好的处理了节点调整权重的问题。通过对于时间序列分析和预测以及对传统的负载均衡算法策略的深入研究，提出了基于时间序列预测模型的动态加权轮询算法；对如何进行合理分配的负载分配和权重的调整进行了深入研究，对加权轮询算法进行的改进，从而使得较好的利用好服务器集群的剩余性能，从而达到降低花销，提高服务器整体负载性能。本文完成的工作内容有以下几个方面。

1. 研究常规负载均衡算法涉及的相关理论和技术，分析了负载均衡的技术意义和优势，了解了集群的概念和分类，通过对 Nginx 源码的了解探究了Nginx具体的工作模式和进程模型，确认了整体的研究方向。
2. 根据Kaggle 比赛“Web Traffic Time Series Forcasting“所提供的网络访问量数据集，运用 ARIMA 模型和 prophet 模型对网络访问量进行预测，并针对不同情况的访问量对比两个模型的性能。最终确定了prophet 在预测网络访问量的性能优势和准确性。
3. 针对 Nginx 内部自带的加权轮询算法不能实时监控各个服务器节点的负载信息的问题，对峙服务器集群不能合理地对请求任务进行分配，制定了在各个服务器节点周期性进行收集并存入 Memcached，尽量减少对于节点资源的消耗。同时创新的设置了两个不同的阶段，解决了改进算法初期没有数据，初期集群不稳定的难题。

本文的创新点如下：

对于传统负载均衡没有考虑到的可能访问量的问题提出了使用prophet 模型来进行访问量预测，并证明了prophet 模型具有较高的准确度和性能。针对负载均衡算法中动态调整权重的问题，对加权轮询算法进行了一定的改进，引入拥塞控制思想，并分为两个不同的阶段，解决了初期没有数据的问题，根据节点负

载预测结果对服务器节点进行调整，提高了集群系统运行期间权重分配的合理性，减少了负载调整的滞后性。

## 工作展望

本文对 Nginx 在负载均衡问题上做了许多研究和分析，与自带的负载均衡策略相比，本文提出的优化后的预测模型动态加权轮询算法一定程度上解决了传统算法的弊病，但是仍旧存在很多的不足。

1. 由于实验条件和笔者技术条件的限制，此次研究没有合适的具体实验数据来对改进的算法进行实验，但是证明了 prophet 模型在访问量预测的有效性，若改进的算法在复杂的服务器集群中，有可能会产生一系列其他的问题。
2. 本文中用户的访问请求全部交给了负载均衡服务器，通过它使用的负载均衡算法进行合理的任务分发。但是如果当负责负载均衡的服务器宕机或者资源利用达到上线，那么会造成整个集群无法正常工作甚至瘫痪，为了解决该问题可以采用Nginx 和KeepAlive 双热机主从备份的方法，保证服务器集群的正常运行。

# 参考文献

1. 国家图书馆研究院. 中国互联网络信息中心发布第 52 次《中国互联网络发展状况统

计报告》[J]. 国家图书馆学刊, 2023, 32(5):13

1. 秦娥. 基于 Nginx 高并发服务器的负载均衡算法研究与改进[D]. [硕士学位论文]. 武汉理工大学, 2020
2. 李杰. 基于 LVS 集群的动态负载均衡算法研究[D]. [硕士学位论文]. 南昌航空大学, 2018
3. Wen Z, Li G, Yang G. Research and realization of nginx-based dynamic feedback load balancing algorithm[C]. 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference : [Volume 1 of 4]. Chongqing, 2018: 2541-2546
4. Li W, Liang J, Ma X, et al. A dynamic load balancing strategy based on haproxy and tcp long connection multiplexing technology[C]. Proceedings of the Fifth Euro-China Conference on Intelligent Data Analysis and Applications 5. Springer, 2019: 36-43
5. Chi X, Liu B, Niu Q, et al. Web load balance and cache optimization design based nginx under high-concurrency environment[C]. 2012 Third International Conference on Digital Manufacturing & Automation. IEEE, 2012: 1029-1032
6. 张淇. 服务器集群负载均衡算法在商务系统中的研究与应用[D]. [硕士学位论文]. 华北理工大学, 2020
7. 王永辉. 基于 Nginx 高性能 Web 服务器性能优化与负载均衡的改进与实现[D]. [博士学位论文]. 成都: 电子科技大学, 2015
8. 王东. 动态反馈负载均衡策略的研究[D]. [博士学位论文]. 哈尔滨: 哈尔滨工程大学, 2018
9. 谭畅, 谭歆, 胡磊, 等. 云中心基于 Nginx 的动态权重负载均衡算法.[J]. Journal of Chongqing University of Posts & Telecommunications (Natural Science Edition), 2021, 33 (6)
10. Kanellopoulos D, Sharma V K. Dynamic load balancing techniques in the iot: A review[J].

Symmetry, 2022, 14(12):2554

1. 刘金秀, 陈怡华, 谷长乐. 基于 Nginx 的高可用 Web 系统的架构研究与设计[J]. 收藏, 2019, 11
2. 刘卓, 张向利. 基于 Nginx 的负载均衡集群设计与实现[J]. 桂林电子科技大学学报, 2017, 37(6):490-493
3. Zongyu X, Xingxuan W. A modified round-robin load-balancing algorithm for cluster-based web servers[C]. The 33th Chinese Control Conference 第 33 届中国控制会议论文集. 南京: 复旦大学, 2014: 3580-3584
4. 中华人民共和国国家质量监督检验检疫总局. 信息安全技术信息系统灾难恢复规范

[Z]. 2007

1. 吴宝花. 基于 Nginx 的服务器集群负载均衡策略研究与优化[D]. [硕士学位论文]. 南昌大学, 2020
2. Mahato D P, Singh R S, Tripathi A K, et al. On scheduling transactions in a grid processing system considering load through ant colony optimization[J]. Applied Soft Computing, 2017, 61:875-891
3. 黄伟华, 马中, 戴新发, 等. 一种特征加权模糊聚类的负载均衡算法[J]. 西安电子科技大学学报, 2017, 44(2):127-132
4. 常智. 高性能 HTTP 反向代理研究与实现[D]. [硕士学位论文]. 哈尔滨工程大学, 2013
5. Pak I, Qiao B, Shen M, et al. An efficient load balancing approach for n-hierarchical web server cluster[J]. Wuhan University Journal of Natural Sciences, 2015, 20(6):537-542
6. 凌质亿, 刘哲星, 曹蕾. 高并发环境下Apache 与Nginx 的 I/O 性能比较[J]. 计算机系统应用, 2013(6):204-208
7. 张炜森, 陈涛, 李康. Nginx 高并发负载均衡原理与策略比较研究[J]. 工业控制计算机, 2018, 31(1):85-86
8. 崔娟, 王伟民, 冯继虎, 等. 基于Nginx 反向代理解决公网上服务跨域问题的研究[J]. 现代信息科技, 2023, 7(8):79-82,87
9. 马原龙. Nginx 负载均衡技术研究[D]. [硕士学位论文]. 重庆邮电大学, 2016
10. 吴陈. 基于 Nginx 的服务器集群负载均衡策略的研究与改进[D]. [硕士学位论文]. 华南理工大学, 2020
11. 邓仲举. 高可靠性集群部署的设计与实现[D]. [硕士学位论文]. 华中科技大学, 2012
12. Sufiev H, Haddad Y. A dynamic load balancing architecture for sdn[C]. 2016 IEEE Inter- national Conference on the Science of Electrical Engineering (ICSEE). IEEE, 2016: 1-3
13. 周常志. 基于改进加权最小连接数的微服务负载均衡算法研究[D]. [硕士学位论文]. 长江大学, 2023
14. 邱亚飞. 哈希算法的实现与验证[D]. [硕士学位论文]. 广东工业大学, 2021
15. 张艳肖, 李守智, 张江江, 等. 基于 Fair 函数神经网络的厚度传感器输出特性分析[J]. 电子设计工程, 2023, 31(24):100-103,108
16. 贾钟研. 基于时间序列的网络流量分析与预测[D]. [硕士学位论文]. 黑龙江大学, 2020
17. 赵鹏, 李璐. 基于 ARIMA 模型的城市轨道交通进站量预测研究[J]. 重庆交通大学学报

(自然科学版), 2020, 39(01):40

1. 李志超, 刘升. 基于 ARIMA 模型、灰色模型和回归模型的预测比较[J]. 统计与决策, 2019, 35(23):38-41
2. Taylor S J, Letham B. Forecasting at scale[J]. The American Statistician, 2018, 72(1):37-45
3. 李威, 鲁铁定, 贺小星, 等. Prophet 模型在 GNSS 坐标时间序列中的插值分析[J]. 大地测量与地球动力学, 2021, 41(4):362-367,377
4. 葛娜, 孙连英, 石晓达, 等. Prophet-LSTM 组合模型的销售量预测研究[J]. 计算机科学, 2019, 46(S1):446-451
5. 李新. 基于Web 服务器集群的动态负载均衡算法改进及实现研究[D]. [硕士学位论文].

湖南大学, 2012

1. KPFRS L. On lines and planes of closest fit to systems of points in space[C]. Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (SIGMOD). 1901: 19
2. 李可佳, 胡学先, 陈越, 等. 基于主成分分析和函数机制的差分隐私线性回归算法[J]. 计算机科学, 2023, 50(8):342-351
3. Lea C, Vidal R, Reiter A, et al. Temporal convolutional networks: A unified approach to ac- tion segmentation[C]. Computer Vision–ECCV 2016 Workshops: Amsterdam, The Nether- lands, October 8-10 and 15-16, 2016, Proceedings, Part III 14. Springer, 2016: 47-54
4. 吕国豪, 罗四维, 黄雅平, 等. 基于卷积神经网络的正则化方法[J]. 计算机研究与发展, 2014, 51(9):1891-1900
5. He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]. Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778
6. Bai S, Kolter J Z, Koltun V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling[J]. arXiv preprint arXiv:1803.01271, 2018
7. 赵洋, 王瀚墨, 康丽, 等. 基于时间卷积网络的短期电力负荷预测[J]. 电工技术学报, 2022, 37(5):1242-1251
8. Hewage P, Behera A, Trovati M, et al. Temporal convolutional neural (tcn) network for an effective weather forecasting using time-series data from the local weather station[J]. Soft Computing, 2020, 24:16453-16482
9. Kingma D P, Ba J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014
10. 吴文辉. 编程计算层次分析法[J]. 电脑编程技巧与维护, 2013(14):20-21

# 附录 A 附录 A 相关模型算法实现主要代码

1 定义一个1-D卷积层的截断模块，主要用于去除sequence右边的padding



2 class Crop1d(nn.Module):

3 def init (self, crop\_size):

4 super(Crop1d, self). init ()

5 self.crop\_size = crop\_size 输入需要被裁剪的序列长度

6

7 def forward(self, sequence):

8 return sequence[:, :, :-self.crop\_size].contiguous() 裁剪输入，返回截断后的序列

9

10 定义用于时间序列预测的1-D卷积神经网络

11 class TemporalBlock(nn.Module):

12 def init (self, n\_inputs,

n\_outputs,

kernel\_size,

stride,

dilation, padding, dropout=0.2):

13 super(TemporalBlock, self). init ()

14

15 定义两个卷积层，对权重进行规范化

16 self.conv1 = weight\_norm(nn.Conv1d(n\_inputs, n\_outputs,

kernel\_size,

17 stride=stride, padding=

padding, dilation=dilation)) 第一层卷积层

18 self.crop1 = Crop1d(padding) 第一层卷积层后面的截取模块

19 self.relu1 = nn.ReLU() 第一层卷积的激活函数

20 self.dropout1 = nn.Dropout(dropout) 第一层卷积的dropout层

21

22 self.conv2 = weight\_norm(nn.Conv1d(n\_outputs, n\_outputs, kernel\_size,

23 stride=stride, padding=

padding, dilation=dilation)) 第二层卷积层

24 self.crop2 = Crop1d(padding) 第二层卷积层后面的截取模块

25 self.relu2 = nn.ReLU() 第二层卷积的激活函数

26 self.dropout2 = nn.Dropout(dropout) 第二层卷积的dropout层

27

28 将两个卷积层、截取模块、激活函数和dropout层按照顺序放入一个新的模型中，并作为TimeBlock的网络模型

29 self.net = nn.Sequential(self.conv1, self.crop1, self.relu1,

self.dropout1,

30 self.conv2, self.crop2, self.relu2, self.dropout2)

31

32 如果输入和输出的特征通道数不同，定义1x1的卷积层进行通道数的转换

33 self.downsample = nn.Conv1d(n\_inputs, n\_outputs, 1) if

n\_inputs != n\_outputs else None

34 self.relu = nn.ReLU()

35 self.init\_weights()  初始化模型参数

36

37  对模型的参数进行初始化

38 def init\_weights(self):

39 self.conv1.weight.data.normal\_(0, 0.01)

40 self.conv2.weight.data.normal\_(0, 0.01)

41 if self.downsample is not None:

42 self.downsample.weight.data.normal\_(0, 0.01)

43

44  前向传播计算

45 def forward(self, x):

46 out = self.net(x)  对输入的x进行卷积操作，并通过relu和dropout层进行处理

47 res\_part = x if self.downsample is None else self.downsample(

48 x)  如果输出的通道数和输入的通道数不一致，则需要定义一个1x1的卷积层进行通道数转换，这里将res\_part指定为输入x或转换后的输出结果。

49 res = out + res\_part  使用相加的方式进行残差连接

50 return self.relu(res)  通过激活函数处理后返回

51

52  定义用于时间序列预测的卷积神经网络模型

53 class TemporalConvNet(nn.Module):

54  num\_inputs是一个列表，其长度代表TemporalBlock的层数，值表示输入序列的通道数（特征维度）。

55  out\_channels是一个列表，其长度代表TemporalBlock的层数，值表示每个 TemporalBlock中输出的通道数（特征维度）。

56  kernel\_size是卷积核的大小，默认为2。

57 def init (self, in\_channels, dropout=0.2):

out\_channels,

kernel\_size=2,

58 super(TemporalConvNet, self). init ()

59 layers = []

60 num\_levels = len(out\_channels)

61  使用for循环，定义多个TemporalBlock模块，并按序添加到网络结构中

62 for i in range(num\_levels):

63 dilation\_size = 2 \*\* i  逐层dilation以2的指数增长

64 in\_channel = in\_channels[i]

65 out\_channel = out\_channels[i]

66 layers += [TemporalBlock(in\_channel, out\_channel, kernel\_size, stride=1, dilation=dilation\_size,

67 padding=(kernel\_size - 1) \*

dilation\_size, dropout=dropout)]

68 self.network = nn.Sequential(\*layers)

69

70  模型前向计算，按顺序执行各模块的前向计算

71 def forward(self, x):

72 return self.network(x)

# 致谢

在毕业之际，以毕业论文的方式结束大学四年的美好时光。在撰写毕业论文的过程中，深刻体会到法学知识就像建房子，扎实的知识基础才会有牢固的地基，构建好论文的框架和清析的逻辑思维才能写好一篇有质量的论文。此次的毕业论文也让我知道，独立的思考善于发现问题是作为一名计算机人应具备的美好品质。

本次毕业论文能顺利完成首先最感谢的是我的指导老师张格老师，从我们论文的选题、开题报告、一稿、二稿、定稿和最终稿，张格老师都以最负责、最认真的态度给我们进行指导，对学生的毕业论文进行严格把关，针对我们的论文进行专业的指导，针对我们存在的问题，给予有建设性的建议，给学生进行论文修改带来了很多便利。对张格老师表示诚挚的感谢。其次也非常感谢对我予以帮助的同学，感谢你们在我写作的过程中，遇到困惑时给予我帮助，对于我的毕业论文完成也起到关键作用。

感谢科技的发展，如果没有日新月异的大模型，就不能做出很多相关的实验。感谢网友的网友的指导，没有你们就没有论文的方向，感谢论文的参考文献的作者们，你们提供给我的理论支持和实验数据，是我论文的骨干。感谢父母背后对我的资金的支持和精神上的支持，没有你就没有安稳地写作环境。

大学四年即将画上圆满的句号，在此也感谢各位任课老师大学四年的教导。对于论文老师指出存在的问题，我会虚心听取，积极的改进。在此再次向各位帮助过我的老师和同学们表示诚挚的感谢。

请君试问东流水，别意与之谁短长。山水有相逢，来日皆可期。

苏峻锋

2024 年 4 月 18 日