

本科生毕业设计

|  |
| --- |
| 基于多GPU的动态图更新和处理机制研究 |

|  |  |
| --- | --- |
| 院 系 | 计算机科学与技术 |
| 专业班级 | CS1601 |
| 姓 名 | 刘本嵩 |
| 学 号 | U201614531 |
| 指导教师 | 张宇 |

2020年6月1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： 年 月 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、保密囗，在 年解密后适用本授权书

2、不保密囗 。

（请在以上相应方框内打“√”）

作者签名： 年 月 日

导师签名： 年 月 日

摘 要

理想的存储体系结构应该提供安全性、跨平台数据共享、高性能、以及可扩展性，然而当前广泛使用的三种存储结构（DAS，SAN和NAS）都不能兼顾所有这些需求。因此，工业界和学术界正在努力改变目前的存储技术，使存储设备从非智能的、外部管理发展成为智能的、自管理的设备，并且能够感知设备所服务的应用。基于对象的存储（Object-Based Storage）被认为是该问题的解决之道。

以SCSI OSD协议为基础，建立属性的两种传递机制。一种是调用专门的应用程序接口直接提供属性信息，存储设备根据属性直接做出存储策略选择。另一种是应用程序提供文件系统层的信息，设备内部根据文件的属性做出相关预测，并用于策略选择。比较而言，前者容易实现，而后者要求新的OSD文件系统，好处是不需要修改应用程序。同时还给出了三个属性页的定义作为OSD标准扩展提议。设计实现了符合T10 OSD标准协议的原型系统（iSCSI-OSD-RAID）。利用iSCSI协议作为OSD命令的传输层，加快了系统的实现过程。同时以一个正在进行的RAID控制器项目为基础，增加了新的对象存储管理模块，最终实现了OSD接口的磁盘阵列，并验证了属性控制的数据放置策略。实验测试结果表明….

**关键词**：对象存储；磁盘阵列；存储策略；数据放置

摘要撰写说明，阅后焚毁！！

中文摘要是对论文内容的高度概括，应用精练的语言概述论文的主要研究内容、目的意义、设计过程、实验手段及取得的成果等。摘要一般分为2~3段，第一段简要介绍背景知识，尽量简洁，切中要害，不要说些任何人都知道的无信息量的语句；后面可以用一到两段介绍毕业设计工作，主要体现自己的工作，摘要不要太长，但主要工作部分应该比第一段背景知识长，不要本末倒置。

摘要中不得出现“本文共有X章，第一章…，第二章…”之类的表述。摘要严禁出现“本文”，“我”，“我们”这样的第一人称主语，尽量采用动宾结构，比如设计了….实现了…利用了….技术…..进行了….实验…..实验结果表明……，如果需要设置主语可以用文中涉及的系统或研究机制等。

摘要不要超过一页，关键词与摘要在同一页，数量3～7个。中文关键词须用汉字，尽量不使用英文单词或其缩写，例如“DBMS”不能作为中文关键词，必须用对应的中文表述：“数据库管理系统”。关键词之间用逗号分隔，最后一个关键词后不用标点。关键词应该具体，不得用过于泛化的词做关键词，如“音乐、视频”，形容词不能作关键词。

Abstract

Failure resilience is of pivotal importance in practical network function virtualization (NFV) systems, but has been mostly absent in the existing ones. The absence is mainly due to the challenge of patching source code of the existing NF software for extracting important NF states, a necessary step toward flow migration and replication to provide failure tolerance.

This paper proposes NFActor, a novel NFV system that uses the actor programming model to provide transparent resilience, high scalability and low overhead in network flow processing. In NFActor, a set of efficient APIs are provided for constructing NFs, with inherent support for scalability and resilience. A per-flow management principle is advocated, different from the existing practice, which provides dedicated micro service chain services for individual flows, enabling decentralized flow migration and scalable flow replication. We implement NFActor and show that it achieves good scalability, prompt flow migration and failure recovery with large numbers of concurrent flows. We also show that NFActor can enable applications such as live NF update and correct MPTCP subflow processing, which cannot be efficiently achieved in previous systems.

**Keywords:** Network Function Virtualization (NFV), Failure resilience, Actor model, Flow migration, Service Chain

撰写说明，阅后删除！！！

英文摘要在中文摘要定稿后再开始撰写，否则会多次修改，不要过于依赖百度翻译等工具。

Abstract是论文的英文摘要，一般对照中文摘要翻译，要求另起一页。Keywords对照中文关键词翻译，与英文摘要在同一页，以“Keywords：”另行顶左开始。Keywords之间用“,”隔开，最后一个Keyword后不用加任何标点符号。

目 录

[摘 要 I](#_Toc23945441)

[Abstract III](#_Toc23945442)

[1 绪 论 1](#_Toc23945443)

[1.1 课题背景 1](#_Toc23945444)

[1.2 国内外研究现状 2](#_Toc23945445)

[1.3 研究目的和主要内容 3](#_Toc23945446)

[1.4 论文结构 4](#_Toc23945447)

[1.5 课题来源 4](#_Toc23945448)

[2 方案论证（或具体背景技术概述） 5](#_Toc23945449)

[2.1 系统需求分析 5](#_Toc23945450)

[2.2 系统可行性分析 6](#_Toc23945451)

[2.3 开发工具分析及选择 6](#_Toc23945452)

[2.4 关键技术分析 6](#_Toc23945453)

[2.5 基本方案制定 7](#_Toc23945454)

[2.6 本章小结 7](#_Toc23945455)

[3 XXX系统设计 8](#_Toc23945456)

[3.1 功能需求 8](#_Toc23945457)

[3.2 系统总体设计 9](#_Toc23945458)

[3.3 功能模块设计 13](#_Toc23945459)

[3.4 本章小结 16](#_Toc23945460)

[4 XXX系统实现 17](#_Toc23945461)

[4.1 过滤器实现 17](#_Toc23945462)

[4.2 属性管理模块实现 18](#_Toc23945463)

[4.3 数据迁移模块实现 19](#_Toc23945464)

[4.4 本章小结 22](#_Toc23945465)

[5 性能测试与分析 23](#_Toc23945466)

[5.1 测试环境 23](#_Toc23945467)

[5.2 功能测试 23](#_Toc23945468)

[5.3 系统界面 23](#_Toc23945469)

[5.4 性能测试 23](#_Toc23945470)

[5.5 本章小结 25](#_Toc23945471)

[6 总结与展望 26](#_Toc23945472)

[致 谢 27](#_Toc23945473)

[7 毕业设计模板基本框架 28](#_Toc23945474)

[7.1 封面 28](#_Toc23945475)

[7.2 原创性声明页 29](#_Toc23945476)

[7.3 摘要 29](#_Toc23945477)

[7.4 目录 29](#_Toc23945478)

[7.5 参考文献 30](#_Toc23945479)

[7.6 附录 30](#_Toc23945480)

[7.7 毕业设计任务书 30](#_Toc23945481)

[7.8 成绩评定页 31](#_Toc23945482)

[8 毕业设计撰写要求 32](#_Toc23945483)

[8.1 图的格式 32](#_Toc23945484)

[8.2 表的格式要求 36](#_Toc23945485)

[8.3 公式 38](#_Toc23945486)

[8.4 流程图 39](#_Toc23945487)

[8.5 常见格式问题 39](#_Toc23945488)

[参考文献 41](#_Toc23945489)

[附录：大学期间发表或提交的论文 44](#_Toc23945490)

撰写说明，阅后删除！！！

目录不会自动更新，排版变动后必须更新目录，更新目录方式：鼠标点击目录，按F9键或者右键更新域，选择更新整个目录即可。

目录仅包含一级标题和二级标题和标题样式，目录是全论文的纲要。中文摘要、Abstract、论文正文的各级标题（不包括第三级）、致谢、参考文献、附录等都应编入目录，标注其页码对照关系，但目录本身不出现在其中。中文摘要、Abstract、目录等使用希腊数字“I、II、…”编连续页码；论文正文、致谢、参考文献、附录等使用“1，2，3，… ”编连续页码。

# 绪 论

本章我们首先介绍了当前动态图处理系统面临的挑战和技术发展趋势，然后分析了动态图处理技术的产生及发展现状，介绍了国内外在GPU动态图处理领域的相关研究工作，并对本文的主要研究内容及工作意义作了具体说明。(除总结那一章以外，每章都加一段引言)

## 课题背景

绪论主要介绍本文的选题背景：说明本设计课题的来源、目的、意义、应解决的主要问题及应达到的技术要求；简述本课题在国内外发展概况及存在的问题，本设计的指导思想等。（请在开题报告中完成第一章绪论部分的内容）

### 研究背景和趋势

在大数据时代, 图处理系统面临着越来越大的数据量. 大规模图处理任务已经成为社交网络(例如Twitter和Facebook), 万维网, 推荐系统等涉及高维数据的系统的重要组成部分. 这些图往往是持续动态更新的, 例如Twitter平均每天有5亿个推文. 同时往往有对这些大规模数据进行实时分析的需求, 因此对大规模动态图的高效处理非常重要.

在最早期的图计算应用中, 往往使用CPU进行动态图的更新和处理. 后来, 随着数据量的增大和大数据时代的到来, CPU变得无法承受这庞大的数据量. 现在, 在GPU高性能节点上构建高效的大规模图数据的算法和系统已经日益成为研究热点, 得益于GPU的极多的计算核心数量, 它拥有比CPU强大得多的并行计算性能. 然而，使用GPU进行动态图处理需要在显存与主存之间进行大量数据传输, 带来了IO上的开销。对于动态图的更新和处理，仍然没有充分的利用计算机的所有计算资源。因此，如何有效在CPU-GPU混合系统上，同时利用CPU和GPU的可用资源，进行动态图更新和处理是一个急需解决的问题。

### 面临的问题和挑战

**同时充分利用CPU和GPU进行动态图更新.** 现有的大数据动态图更新系统中, 都是使用纯GPU来进行计算的. 尽管算法已经针对GPU的众核特点而进行了充分的优化, 但CPU到GPU的数据传输延迟仍然对性能造成重要影响. 如果能够将部分数据交由CPU和主存进行处理, 就可以降低任务延迟, 减少计算过程中IO等待时间的浪费, 并因此达到更高的吞吐量.

以GPMA+算法为例, 我们在测试中发现, GPMA+的GPU计算核心利用率可以达到70%左右, 但CPU却只有一个核心在满负荷工作.如果能够同时发挥CPU和GPU的计算能力, 就能够更充分的利用CPU靠近主存的优势, 进一步提升动态图更新任务的性能.

## 国内外研究现状

必须有有足够的文献阅读量，这部分替代原来的文献综述。

### 基于GPU的图处理系统

自从GPU图处理问题成为研究热点以来, 很多相关的研究已经陆续开展起来: Gunrock是一个利用GPU进行高性能图处理的库. 它对常用的图操作进行了抽象, 提供了简洁而强大的API, 并针对GPU计算中的IO, 负载均衡, 任务管理等问题进行了很多特定优化, 获得了比CuSha, MapGraph, Ligra更优异的性能表现.

在此基础上, Groute针对多GPU系统的通信优化问题提供了对应的高层抽象, 并增加了分布式工作表(Distributed Worklists), 流水线处理(Pipelined Operation)等一系列针对多GPU的优化, 大大降低了GPU间和CPU间的通信延迟带来的性能损失问题, 在多GPU系统上取得了比Gunrock更好的性能表现.

关于分布式图处理的分区问题, Gurbinder Gill讨论了包含1D, 2D和其他策略等多种分区方式, 揭示了不同分区策略对分布式系统中的通信模式的影响, 论证了CVC这种分区方式更有利于不同运算节点间的高效交流.

### GPU上大规模动态图的存储

// TODO:

2. Garaph:Efficient GPU-accelerated Graph Processing on a Single Machine with Balanced Replication , ATC 2017

~~3. CGraph: A Correlations-aware Approach for Efficient Concurrent Iterative Graph Processing, ATC 2018~~

~~10. Chronos: a graph engine for temporal graph analysis. EuroSys 2014~~

实际生产环境中出现的动态图往往是极度稀疏的, 因此图的存储等同于稀疏矩阵的存储, James King提出的DCSR数据结构在CSR的基础上, 增加了额外的Segments来缓存更新, 在充分利用GPU的巨大并行计算能力的同时, 还能兼容为传统CSR设计的图算法, 在GPU动态图更新领域获得了很大性能提升.

但是, DCSR是为插入设计的, 而不支持删除和快速搜索. 为此, 新加坡国立大学提出的GPMA和GPMA+数据结构, 与DCSR采用了不同的设计思路. 它采用类似于B树的无锁结构来管理树存储空间, 并使用细粒度的re-balance操作取代DCSR的Segments Merge操作, 在频繁的动态图更新中更充分的发挥了GPU的众核计算能力, 并且二进制数据结构一样兼容CSR格式.

### 大规模动态图的实时分析

除了动态图的存储问题, 对于大数据动态图中的实时分析需求, Keval Vora注意到动态图中不同Snapshots的相似性, 由此提出了Fetch Amortization和Processing Amortization的优化方案来减小IO和计算负担, 并在ASPIRE和GraphLab图处理系统中取得了很好的效果. 后来, GraPU在此基础上, 针对快速更新的动态图提出了在Buffer Node对Updates进行pre-computing, 以及通过划分Component与Subgraph来平衡不同计算单元workload的方法, 使之更加适合并行处理.

## 研究目的和主要内容

在本次课题研究的目的是, 在动态图处理任务中同时充分利用CPU和GPU的可用资源. 我们首先在CPU和GPU上分别实现GPMA+的数据结构, 然后设计并实现一个支持多设备的分布式GPMA+数据结构, 并设计和实现一个用于测试的分布式图处理算法. 然后设计在不同设备之间分配任务的组件, 并测试多种任务分配策略的实际性能表现.

## 论文结构

Recolic: 这里最后完成

本文的主要内容如下：

第一章我们首先介绍了当前存储系统面临的挑战和技术发展趋势，然后分析了对象存储技术的产生及发展现状，介绍了国内外在存储策略和智能存储领域的相关研究工作，并对本文的主要研究内容及工作意义作了具体说明。

第二章首先介绍几种存储技术、特别是存储区域网（SAN）在安全性、数据共享和设备管理等方面的存在问题。然后详细论述了基于对象的存储如何解决当前的问题。最后讨论了对象存储的实现途径，OSD标准和Lustre文件系统代表了两种发展方向。本文将重点以OSD为基础进行有关研究。

第三章以磁盘阵列作为研究对象，描述了属性控制的数据放置策略。我们建立了基于闭环fork-join排队网络（CFJQN）的RAID-5读写模型，对最优分条单元大小进行了定量分析，并归纳得到可以用来准确描述一个负载的若干属性（包括存储需求和行为）。

第五章比较详细的描述了一个原型系统的设计和实现。iSCSI-OSD-RAID成功地对传统的RAID控制器进行了扩展，增加了支持iSCSI和OSD协议的访问接口。除了软件方面的工作之外，还实现了基于Intel IOP321处理器的控制器硬件，本章对硬件结构和交叉开发环境也进行了介绍。最后还给出了部分测试结果。

第六章总结了所做的工作，并计划了下一步的工作以及展望了基于对象存储的前景。

# 相关背景技术介绍

第二章可以进行项目方案论证或进行具体背景知识的概述，说明设计原理并进行方案选择，阐明为什么要选择这个设计方案，包括各种方案的分析、比较，以及所采用方案的特点

本章名字可变，名字最后可结合自己的课题内容，如果毕业设计是一个大项目的一个模块，本章可以介绍整体项目结构以及相关背景知识，如果毕业设计是科学研讨型，第二章，第三章都可以介绍理论背景知识。

（除总结那一章以外，每章都加一段引言）

本章介绍CPU+GPU混合动态图处理使用到的相关技术, 这包含GPMA+数据结构和多设备的负载均衡方案.

## 系统需求分析

现有动态图处理系统往往是采用单一的多CPU或多GPU进行动态图处理. 然而, 使用GPU进行动态图处理的系统, 通常不能有效的利用系统中的CPU核心. 尽管算法已经针对GPU的众核特点而进行了充分的优化, 但CPU到GPU的数据传输延迟仍然对性能造成重要影响. 以GPMA+算法为例, 我们在测试中发现, GPMA+的GPU计算核心利用率可以达到70%左右, 但CPU却只有一个核心在满负荷工作.

如果能够将部分数据交由CPU和主存进行处理, 就可以降低任务延迟, 减少计算过程中IO等待时间的浪费, 并因此达到更高的吞吐量. 我们需要实现一个能够同时发挥CPU和GPU计算能力的动态图更新系统, 以便更充分的利用CPU靠近主存的优势, 进一步提升动态图更新任务的性能.

## 现有图存储数据结构

// TODO: 这附近几章都有抄参考文献, 把它的参考文献也一起copy过来.

由于现实中的大规模图数据都是稀疏的, 因此图数据的存储一般可以等同于稀疏矩阵的存储.

COO格式是最简单的稀疏矩阵格式。它用三个数组存储整个矩阵，数组中分别保存行索引，列索引以及矩阵中所有非零条目的值。为了有效执行SpMV操作，必须按行对COO格式内的条目进行排序。

CSR / CSC格式与COO相似，因为它们的数组除了值外，还完全存储三个数组中的两个，即列索引或行索引。行索引数组或列索引数组（分别对应CSR或CSC）被压缩, 仅存储与其他两个数组中的行/列位置相对应的偏移量。对于CSR，行偏移量数组中的第i和i + 1项分别存储行i的起始和结束偏移量。由于CSR可以压缩列索引的特点，因此已被证明是内存访问和SpMV性能最好的数据存储格式之一，并且已经被广泛使用[9]。CSR的压缩数组减少了内存带宽的占用，这是加速SpMV操作的重要因素。

ellpack（ELL）格式使用两个数组，每个数组的大小为m×k（其中m为行数，k为固定宽度），用于存储列索引和矩阵[10，11]的值。这些数组以列优先顺序存储，以便对不同行进行更高效的并行访问。这种格式适合每行具有固定元素数的矩阵, 而如果一行中存在过多的元素，ELL的内存空间开销将会非常昂贵。混合ellpack（HYB）格式通过结合使用ELL和COO提供了一种折衷方案。它将矩阵元素尽可能多地存储在ELL部分中，而元素数量大于ELL行宽度的元素将溢出到COO结构中。由于线程扭曲能够以有效的并行方式浏览连续的行，因此ELL和HYB在SIMD架构上已变得很流行[12]。

与此同时, 研究者还开发了许多其他的稀疏矩阵格式，包括对角线（DIA），锯齿形对角线存储（JDS），块对角线（BDIA），天际线存储（SKS），平铺COO（TCOO），块ELL（BELL），和sliced-ELL（SELL）[13]等. 它们为一些特定类型的矩阵提供了更好的性能。

## DCSR技术介绍

DCSR(动态CSR)是一种在CSR基础上改进得到的数据格式，它可在保持高效的SpMV操作性能的同时进行快速的动态更新操作。 DCSR的动态分配过程维护一个row offset数组，这个密集的数组像CSR一样保存了每个行的偏移量信息。列索引和矩阵元素存储在另外两个数组中，该数组在逻辑上分为多个segment，其方式与CSR行偏移量对列索引和矩阵元素进行分区的方式相同。每个segment都是一段连续内存，用于存储行中的元素。段可能包含空隙，用来允许高速的插入。

在向某一行中插入新的元素时, 先确定本行的最后一个segment, 如果空余空间足够, 则会执行插入. 否则就分配新的segment来存储新的插入. defragment过程被偶尔执行, 来减少segments的数量. 同时, 如果要运行CSR上的算法, defragment过程可以将DCSR数据结构转换成与CSR几乎完全相同的底层结构, 以便运行为传统CSR设计的图算法.

## GPMA+背景技术介绍

GPMA中的并发插入受PMA在CPU上的启发，它在GPU上同时并行处理一批插入。直观地，GPMA将插入分配给线程，并使用基于锁的方法并为每个线程同时执行PMA算法，以确保一致性。更具体地，预先识别所有插入的叶子节点上的段，然后每个线程从下到上检查插入的片段是否仍然满足其阈值。对于每个特定的段，以互斥的方式访问它。此外，在更新位于树上同一高度的所有段之后，所有线程将同步，以避免可能的冲突，因为较低级别的段完全包含在较高级别的段中。

给定一个已经存储在GPMA上的图， 还需要为GPMA设计对应的图算法。特别地，让现有算法能够访问GPMA格式的图数据至关重要。对于CSR数据格式，大多数算法都通过访问CSR中的有序数组来定位要访问的元素。因为存储在GPMA上的稀疏矩阵数据也是一个有序数组，该数组与CSR类似地具有交错的间隙。因此，我们能够用GPMA的操作有效地替换现有算法中对CSR的操作。

由于GPMA具有无序的内存访问, 锁开销, 线程冲突, 线程负载不平衡这4个问题, 更优化的GPMA+算法被提出. 这种算法具有以下3个特点:

（1）更新首先按其键排序，然后分派到GPU线程以根据排序顺序定位其对应的叶段。

（2）将属于同一叶段的更新分组以进行处理，GPMA +以自下而上的方式逐级处理更新。

（3）在树的每一级中，利用GPU原语来调用所有计算资源以进行段更新。

在以上算法中, 组件（1）解决了GPMA中未分批进行内存访问的问题，因为对更新线程进行了预先排序以实现有规律的遍历路径, 优化了缓存性能。组件（2）完全避免使用锁，从而解决了原子操作和线程冲突的问题。最后，组件（3）利用GPU原语在所有GPU线程之间实现工作负载平衡。值得注意的是, GPMA和GPMA+的算法有区别, 而底层数据结构并没有大的区别.

## 常用图分区技术介绍

在1D分区中，节点在主机之间分区。 如果某一个主机拥有节点n，则将连接到n的所有出边或入边分配给该主机。 在图分析领域，这种划分策略称为Outgoing或Incoming Edge-Cut [23，41，42，51]。 用矩阵理论的术语来说，这相当于为主机分配行（或列）。 一维分区通常用于科学计算应用中的代码中；代码可以通过向所有主机分配大致相等数量的节点来实现计算负载平衡。 在2D分区中，邻接矩阵沿两个维度都被分块，并且每个主机被分配了一些块。与一维分区不同的是，某一个节点的出边或入边都可以分布在不同的主机之间。在图分析领域, 这种划分策略被称为Vertex-Cut. 二维分区包含BVC, CVC, JVC等常用策略.

CheckerBoard Vertex-Cuts (BVC)将邻接矩阵划分成相同大小的块, 并且分配给每个主机, 被分配的主机就是这个块内所有节点的master. Cartesian Vertex-Cuts (CVC)将节点按照任意的一维分区方法分配给不同的主机, 然后被分配的主机就是此节点的master. 与BVC不同的是, CVC的分区可以不是等大小的. Jagged Vertex-Cuts (JVC)与CVC类似, 然而JVC不是将整个矩阵分为相等大小的块, 而是每一行可以被独立的进行分区, 这样可以得到更平衡的负载分配. 在Gurbinder Gill的论证和测试中发现, CVC的分区方式可以最小化不同主机之间的数据交换, 使得整个图系统的通信更有效率.

## 开发工具分析及选择

由于我们的GPU编程需要使用到NVIDIA的显卡, 因此需要使用CUDA来对GPU进行编程. 在CPU端, CUDA 10.2本身支持C++14标准, 能够满足CPU程序开发的需求.

同时, 我们使用了高性能的thrust和cub库来提供部分常用算法在单设备上的并行实现, 例如BlockScan, ExclusiveSum, RadixSort, RunLengthEncoding等.

## 本章小结

（第二章到倒数第二章每章都必须要有一个小结）本章提出了一种可以提高存储系统数据可靠性的技术，即存储设备安全预警及数据主动迁移技术，它包括磁盘数据自愈、磁盘数据主动迁移、阵列级数据自愈及系统级数据主动迁移四个层次。对磁盘数据自愈与数据主动迁移进行了详细描述。

# 动态图更新系统设计

前面论述了使用GPMA数据结构和GPMA+算法, 完成多设备动态图处理任务的必要性、重要性和可行性。GPMA+算法和数据结构拥有更强的并行优化潜力, 非常适合用作CPU-GPU混合动态图更新任务的底层算法之一. 本章详细描述了在动态图更新任务中, 关键数据结构和算法的设计，并就各功能设计时需要考虑的问题及解决方案进行了讨论。

## 功能需求

1. 首先在GPMA+算法的基础上, 完成一个高性能的算法在CPU上的实现. 以便支持接下来的CPU-GPU多设备GPMA.
2. 新实现的动态图更新系统能够高性能的完成大规模图数据的更新, 包括插入和删除等操作. 重点优化更新的过程, 用最小的开销完成并行优化.
3. 新实现的动态图更新系统能够充分利用系统内的多核CPU和GPU资源, 要将图的数据分区在不同的设备上, 并且在进行更新时充分利用每一个CPU核心和GPU核心的计算能力.
4. 除了动态图的更新操作外, 还应该在分区后的图上支持至少一种常用的图算法, 用于正确性测试. 程序用常规图算法的结果来检验正确性.
5. 一般的系统都必须要有需求分析，有功能需求，界面需求，接口需求，性能需求等等。具体参照软件工程，这是系统设计的重要依据，一定要将自己做什么说清楚，如果第二章写了需求这章就不需要了

**TODO 这一章的图都要改矢量图**

## 系统总体设计

### 单设备数据结构和图更新

在单个GPU或单个CPU上的数据结构采用已知单GPU上性能最好的GPMA+. 单设备上的GPMA+数据结构概览如图3.1所示.

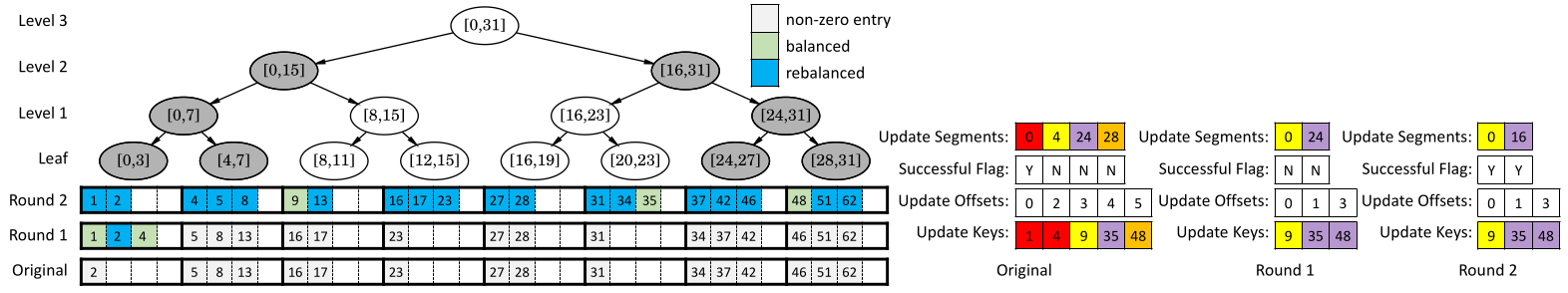


图3.1 GPMA+数据结构概览图

GPMA+数据结构是将PMA数据结构针对GPU进行优化而来的. PMA通过在一个有序的数组中预留空隙, 来允许高速的并行插入操作. PMA是一种自平衡的二叉树结构, 对于在任意一个高度上的节点(或者说是它所对应的一个segment), PMA记录了这一个segment的空槽使用率的上界和下界, 并在必要的时候进行re-balance操作. 正如一般的自平衡二叉树那样, 这种re-balance操作会由叶子向根进行递归, 并在必要时加倍或减半整体数组的空间.

在这种数据结构中, 一般的插入或删除, 很少需要递归到深层次的节点, 并且不同节点的re-balance操作可以被并行化, 可以取得较好的并行度.

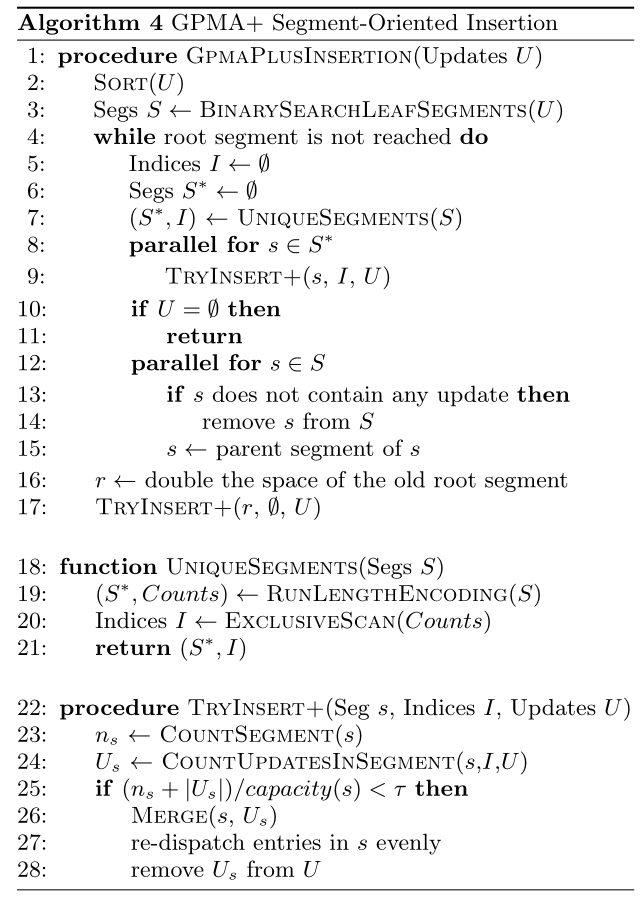


图3.2 GPMA+插入过程中, 算法的伪代码

对GPMA+数据结构的单设备插入过程如图3.2所示. Updates是一个由Key, Value对构成的数组. 在具体的实现中, 这里的key是由起始终止节点的ID构成的, 表示一条边的变量. 而value用1表示插入, 0表示删除.

GPMA+的插入过程, 首先在第2行将更新按key进行排序, 然后在第3行并行查找到节点所对应的段. 随后GPMA+在4-15行, 逐级更新segments, 完成更新操作. 其中, 第7行的UniqueInsertion将属于同一个segment的更新分类合并到S\*, 并在TryInsert过程中检查segment的密度是否满足上下界要求, 如果满足要求便在这一级完成这一批更新的插入或删除.

在UniqueSegments的实现中, 使用了适合GPU的RunLengthEncoding和ExclusiveScan. 对于核心数量相对较少的CPU, 使用常规的同步原语代替这一部分操作. 对于核心数量极高的GPU, 使用了NVIDIA CUB库来完成这两个操作. 这一步产生的IndexSet, 被传递到TryInsert函数中是为了在GPU中辅助进行同步.

最后, 如果TryInsert过程由于segment密度不符合要求而失败, GPMA+会逐级向更高的根节点进行递归. 如果已经递归到根节点, 则会对整体存储空间进行翻倍或减半, 并最终完成所有的插入操作.

GPMA+的插入与删除使用相似的逻辑. 在我们的实现中, 通过插入一个value为0的边来表示删除, 对具体实现逻辑的修改是极小的.

### CPU和GPU的多设备图更新

在对多设备GPMA进行初始化时, 我们为每一个设备都分配一个GPMA数据结构. 一般来说, 我们为每个CUDA设备(GPU)分配一个GPMA结构, 并为每个CPU核心分配一个GPMA结构. 如果系统中由N个CUDA设备, M个CPU核心, 则需要分配N个GPU上的实例, 和M-N个CPU上的实例. 随后, 系统对每一个GPMA实例进行初始化.

在进行插入时, 如图3.3所示, GPMA\_multidev会先将这一批插入按分配器的策略分配给每一个GPMA实例. 随后, 如果必要, 将CPU上的buffer拷贝至GPU, 然后并行的启动每一个实例的update\_gpma操作. 在每一个设备的update操作结束后, 整体的更新就被插入完成了.

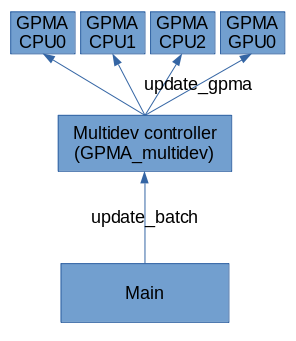


图3.3 多设备GPMA插入示意图

（软件整体结构图必须提供，有些系统还需要提供详细的动态运行分析图）

### 单设备的图算法

由于GPMA的数据结构与传统CSR具有相似性, 因此很容易将为CSR设计的算法迁移到GPMA上. 如图3.4所示, GPMA会像CSR一样记录Row Offsets数组, 并利用Row Offsets来访问对应偏移量上的元素. 随后, 在并行GPU算法中剔除掉NULL元素(这个过程容易并行, 因此在GPU上很快), 即可像CSR一样进行分析.

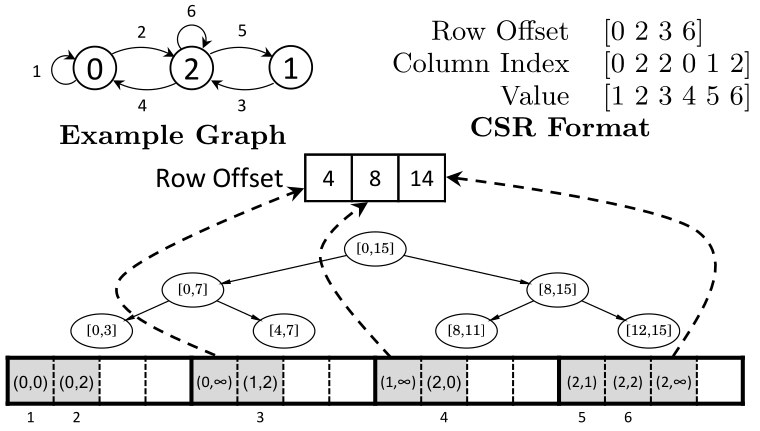


图3.4 GPMA与CSR结构示意图

GPU上进行的GPMA算法分为两个部分: 在循环中, 先通过gather操作, 将node queue的所有节点的邻居保存在edge queue中, 然后通过contract操作, 将edge queue中的所有下一级节点进行标注, 选出其中的新节点, 标注上level, 并放进node queue. 随后重复这个循环, 直到edge queue中没有任何新节点, BFS操作结束. 这整个过程如图3.5所示.

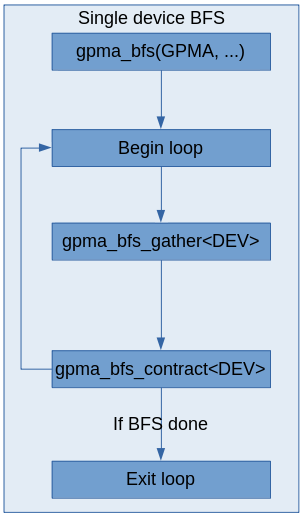


图3.5 单设备的BFS过程示意图

将这gather和contract操作分开的做法, 增加了算法的并行度, 使算法在GPU上获得了更好的性能. 对于CPU上的操作, 只需用OpenMP将contract和gather操作并行化即可.

### CPU和GPU的多设备图算法

为了将单个设备上的BFS算法改为支持多个设备, 我们设计了多设备上并行完成的BFS过程.

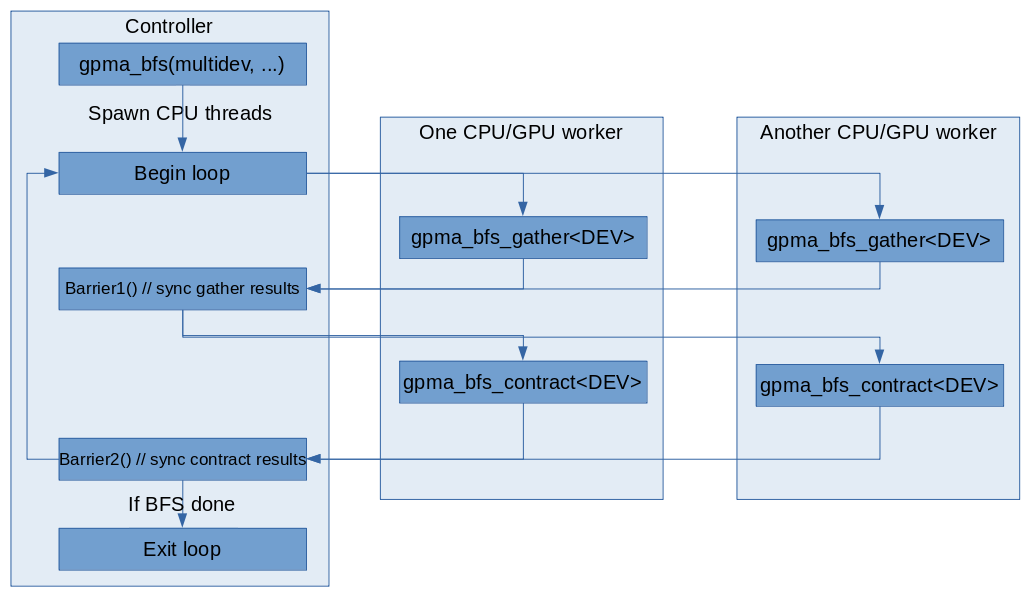


图3.6 多设备的BFS过程示意图

我们在多个设备上进行的BFS过程如图3.6所示. 我们首先为每一个GPMA实例产生一个CPU线程, 然后每一个CPU线程执行一个实例的gather 操作. 在gather操作完成后, 所有线程进行一次同步, 并对每个实例产生的edge queue进行合并.

由于我们的计算机往往是几十个CPU核心, 为了避免对大量的队列进行合并带来的性能开销, 我们在所有的CPU实例共用同一个node队列和edge队列, 它们用CPU上的原子操作进行元素的并行插入. 而不同GPU实例一般不会共享内存空间(除非使用cudaManagedMemory), 而机器上的GPU实例一般只有一到两个, 所以每一个GPU实例都有一个独立的node队列和edge队列参与合并.

在edge queue合并完成后, 每一个实例继续执行自己的contract操作. 操作完成后, 用相同的方法来合并node queue和results数组. 随后, 如果node队列为空, 所有线程就可以全部结束, 操作完成.

### 图分区策略

在本文的工作中, 将节点和更新分配到设备的分配策略对性能影响非常重要. 首先, GPMA数据结构中存储的是边. 我们令每一个设备拥有一些节点 , 将每一个边的起始节点所属的设备, 作为负责存储这一条边的设备. 同时, 随着边的数量的增加, 分配器在为新的节点分配设备的同时, 可以追踪目前各个设备的负载情况, 并且尽可能偏好较空闲的设备.

首先, 系统要确定一个系数, 用来表示单个GPU设备与CPU设备的算力的比值. 这个系数根据服务器上的硬件配置, 运行性能测试来进行确定. 然后, 分配器可以根据这个系数以及负载情况, 在运行时实时调整节点的分配情况.

在我们的GPMA数据结构, 插入算法, 和图算法的实现中, 为了保证对边进行删除时, 删除操作能被分配到含有这条边的节点上. 因此, 一个边被分配给一个设备后, 暂时不支持迁移操作.

## 本章小结

本章详细描述了CPU-GPU混合动态图更新系统中, 关键数据结构和插入,BFS, 图分区等关键算法的设计，对其主要功能模块进行了详细的说明。并就各功能模块设计时需要考虑的问题及解决方案进行了讨论。

# 动态图更新系统实现

我们将设计一个动态图更新系统. 该系统在传统的GPU图更新和图算法的基础上, 利用类似于分布式系统的设计思路, 让空闲CPU的算力也得到充分利用. 我们实现了同时支持多个CPU/GPU并行计算的动态图更新系统, 和配套的一个图算法.

## GPMA+算法的通用实现

我们需要将GPMA+的数据结构和算法, 实现为既能在CPU上工作, 又能在GPU上工作的模板类或模板方法. 以便为后面的多设备GPMA+提供基础实现.

### CPU/GPU的通用GPMA数据结构

*template <dev\_type\_t DEV>*

*class GPMA {*

*public:*

*NATIVE\_VEC\_KEY<DEV> keys; // 存储实例中所有的Key*

*NATIVE\_VEC\_VALUE<DEV> values; // 存储实例中所有的Value*

*SIZE\_TYPE segment\_length;*

*SIZE\_TYPE tree\_height;*

*// 段的密度上限和下限, 在超越限制时进行re-balance*

*static constexpr double density\_lower\_thres\_leaf = 0.08;*

*static constexpr double density\_lower\_thres\_root = 0.42;*

*static constexpr double density\_upper\_thres\_leaf = 0.92;*

*static constexpr double density\_upper\_thres\_root = 0.84;*

*thrust::host\_vector<SIZE\_TYPE> lower\_element;*

*thrust::host\_vector<SIZE\_TYPE> upper\_element;*

*// 为CSR格式兼容性而添加的额外成员*

*SIZE\_TYPE row\_num;*

*NATIVE\_VEC\_SIZE<DEV> row\_offset;*

*};*

数据结构中比较重要的字段是keys和values, 这里存储着图3.1所显示的最底层的数组. 在数据结构初始化时, 就会有一个4个元素的空树被插入到keys/values中. row\_offset成员也会在每一个更新的过程中被维护, 这个数组就相当于CSR的row offset数组, 在BFS算法运行过程中发挥重要作用. segment\_length和tree\_height记录了树的总高度, 和segment的长度, 它们会在update\_gpma过程中被维护. 同时, 结构中还包含了在re-balance中会用到的结构和阈值常量.

### GPMA+插入过程

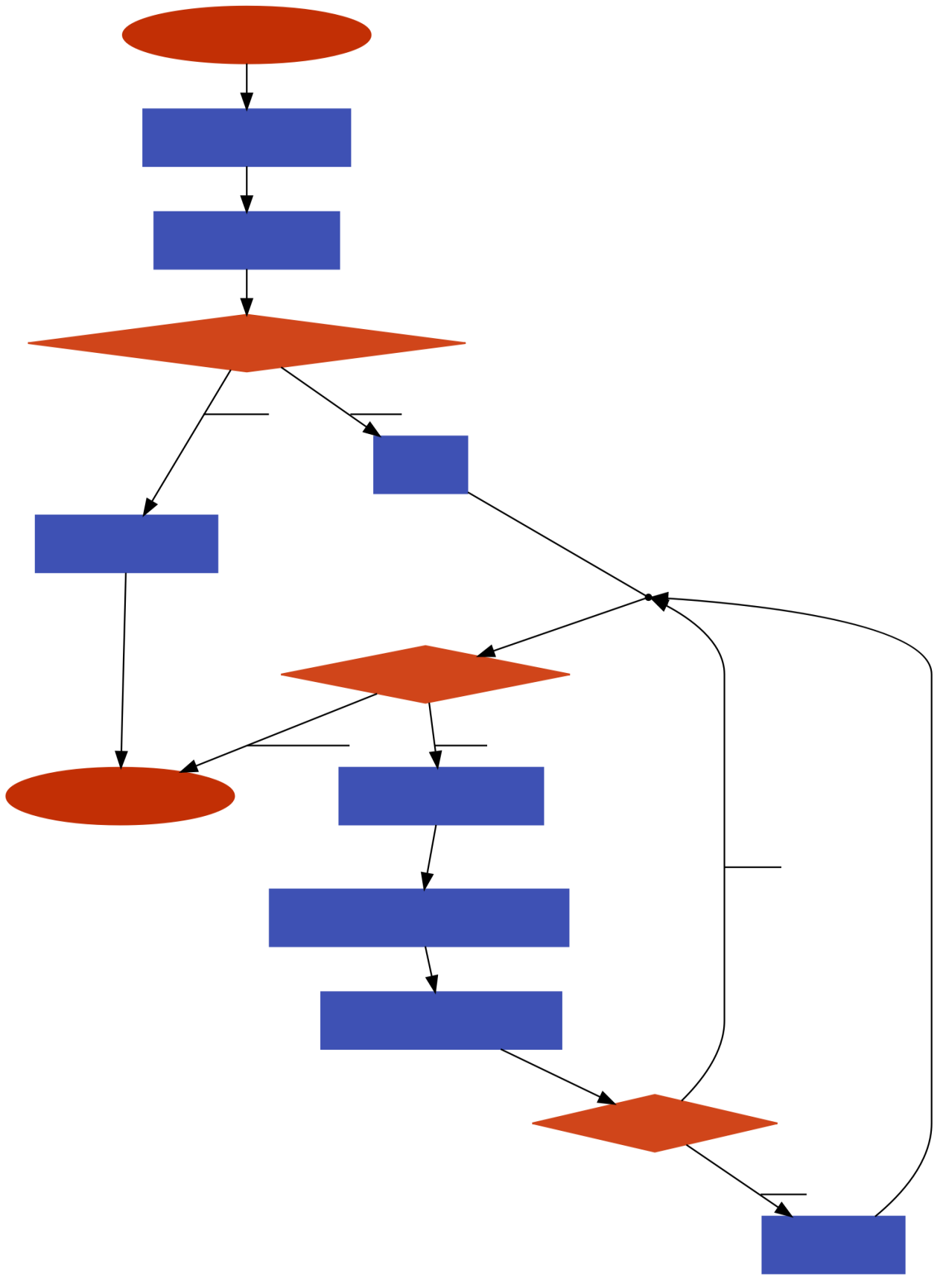


图4.1 GPMA+插入算法工作流程图

TODO 在windows上补充垃圾wps缺少的SVG字体.

在GPMA+算法的图更新过程中, 大致流程如图4.1所示. 首先用sort\_by\_key将输入的更新进行排序, 然后定位key所对应的叶子节点. 随后对远大于当前gpma容量的巨大更新进行特殊处理, 通过significant\_insert过程, 更高效的维护GPMA数据结构.

在一般情形下, 程序会进入逐层迭代的循环中, 在树的每一层尝试将对应于本segment的更新进行插入. 如果segment的密度超过限制, 则会将失败的更新留在updates数组中, 并在下一次循环中, 在更高一层再次尝试处理. 最终, 如果一直递归到根节点, 都仍有更新未能被合并, 就说明GPMA的当前容量不足以必须扩充整体GPMA的数据结构空间, 调用resize\_gpma完成新空间的分配.

在更新过程中, rebalance\_batch是最关键的部分. 它负责将要更新的数据与原segment进行合并. 在GPMA<GPU>的rebalance\_batch实现中, 如果更新的大小小于CUDA支持的block数量, 就直接将更新按固定的方式分配给每一个cuda block. 这有利于减少处理小更新时的调度开销. 如果更新数量较大, 就使用rebalancing\_kernel函数, 借助cub库的帮助, 用更灵活的方式来为cuda核心分配任务.

而在GPMA<CPU>的rebalance\_batch实现中, 使用类似的思路重新实现了GPU上数据量大于block数时的更新逻辑, 调用被OpenMP并行化的rebalancing\_impl\_cpu, 完成re-balance操作.

## 多设备GPMA和图分区的实现

### 设备管理数据结构

GPMA\_multidev代表一个分布在多个设备上的GPMA结构, dispatcher代表负责进行负载分配和均衡的结构. 其主要数据结构如下文所示.

*template <size\_t cpu\_instances, size\_t gpu\_instances>*

*struct GPMA\_multidev {*

*private:*

*static constexpr size\_t instances = cpu\_instances + gpu\_instances;*

*// 这两个数组存储所有设备上的GPMA实例指针.*

*std::array<GPMA<CPU> \*, cpu\_instances> ptrs\_cpu;*

*std::array<GPMA<GPU> \*, gpu\_instances> ptrs\_gpu;*

*public:*

*// 构造和析构上面的所有GPMA实例指针.*

*GPMA\_multidev(size\_t row\_num) {...}*

*~GPMA\_multidev() {...}*

*// 用来在不同设备间分配边的负载分配器.*

*gpma\_impl::dispatcher<cpu\_instances, gpu\_instances> dispatcher;*

*};*

*template <size\_t cpu\_instances, size\_t gpu\_instances>*

*struct dispatcher {*

*private:*

*static constexpr KEY\_TYPE hashSize = 1024;*

*// 这是一个由单个vector组成的哈希表, 用于缓存节点分配情况.*

*quick\_1way\_hash\_table<KEY\_TYPE, size\_t> mapKeyToSlot;*

*public:*

*dispatcher()*

*: mapKeyToSlot(hashSize, (size\_t)(-1)) {}*

*static constexpr size\_t gpu\_factor = 7;*

*[[gnu::always\_inline]] size\_t select\_device(const KEY\_TYPE &k) ;*

*};*

上述GPMA\_multidev结构通过它的两个模板参数进行实例化. 在此结构被构造时, 它为每一个设备分配一个单设备GPMA数据结构. 它还含有一个dispatcher结构, 这个对象负责在update\_batch时决定key所属的设备.

由于dispatcher和GPMA\_multidev时解耦的, 这里提供一种较简单的dispatcher实现. 在dispatcher接口中, select\_device函数用于确定一个key所对应的设备编号. 在我们的简单实现中, 有一个表示GPU相比于CPU的相对性能的系数gpu\_factor, 它通过实际测试来进行确定. mapKeyToSlot是一个当作哈希表使用的普通数组, 它由hashSize个槽, 每一个key会被映射到一个槽上, 然后槽内存储的dev\_id, 就表示对该槽对应的key进行负责的具体设备编号.

### 多设备图更新和图分区

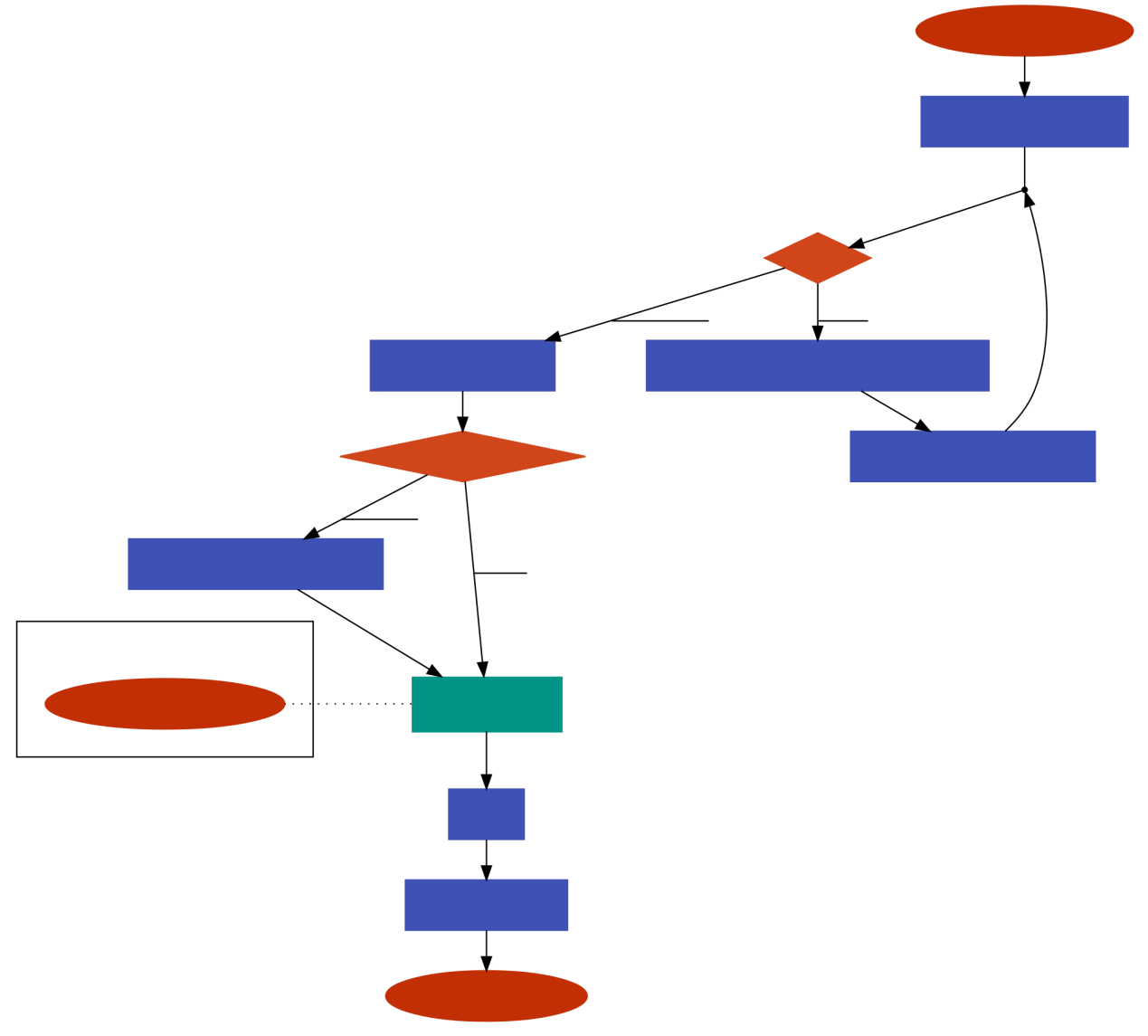


图4.2 多设备图更新流程图

在多设备图更新中, 使用一个update\_batch代替单设备的update\_gpma. 在update\_batch过程中, 首先将要更新的数据传递给dispatcher进行分配, 然后将分配后的更新并行的发送给每一个设备, 并行调用每一个设备的update\_gpma方法, 完成数据的更新. 这个过程如图4.2所示.

在dispatcher的几种可能实现中, 我们尝试了多种不同的分区方式. 但是, 注意到多设备BFS的实现中, 存在一个在多设备间同步node queue和edge queue的过程, 同时注意到对图的更新操作远大于BFS操作耗时. 因此, 当前应用中分区策略对通信模式的影响并不明显, 我们选择了对更新操作更有利的分区方式.

在现在的实现中, 所有的更新被分为固定数量的组, 每个设备将会负责一些任务组. 任务组到设备的映射会在更新的过程中, 按照gpu\_factor系数来进行建立. 而单个更新到任务组的映射, 只与组的数量这个常量有关. 这种做法在保证更新时性能的同时, 减小了任务分配过程中的开销.

## 多设备GPMA-BFS实现

### 多设备BFS相关数据结构

*template <size\_t cpus, size\_t gpus>*

*struct multidev\_bfs\_data {*

*private:*

*native\_vector<CPU, SIZE\_TYPE> cpu\_results, cpu\_bitmap, cpu\_nodeQ, cpu\_edgeQ;*

*native\_vector<GPU, SIZE\_TYPE> gpu\_results, gpu\_bitmap, gpu\_nodeQ, gpu\_edgeQ;*

*SIZE\_TYPE cpu\_nodeQ\_size = 0, cpu\_edgeQ\_size = 0;*

*SIZE\_TYPE \*gpu\_nodeQ\_size, \*gpu\_edgeQ\_size;*

*std::atomic<bool> select\_one\_thread;*

*Barrier barrier;*

*public:*

*multidev\_bfs\_data(size\_t node\_size, size\_t edge\_size);*

*void iteration\_barrier\_1(size\_t dev\_id);*

*bool iteration\_barrier\_2(size\_t dev\_id);*

*};*

BFS所用到的数据结构主要有, 表示每一个节点的层数的results数组, 用来表示节点是否被计算过的bitmap, 表示下一层节点的node queue, 表示本层节点的所有邻居的edge queue, 加上barrier等用于同步的辅助数据结构. 在results数组中, 每一个节点用0表示未访问过, 1表示初始节点所在的层, 依次类推. 两个barrier函数用于不同设备BFS过程的同步, 详细逻辑如4.3.2节所述.

### 多设备BFS算法实现

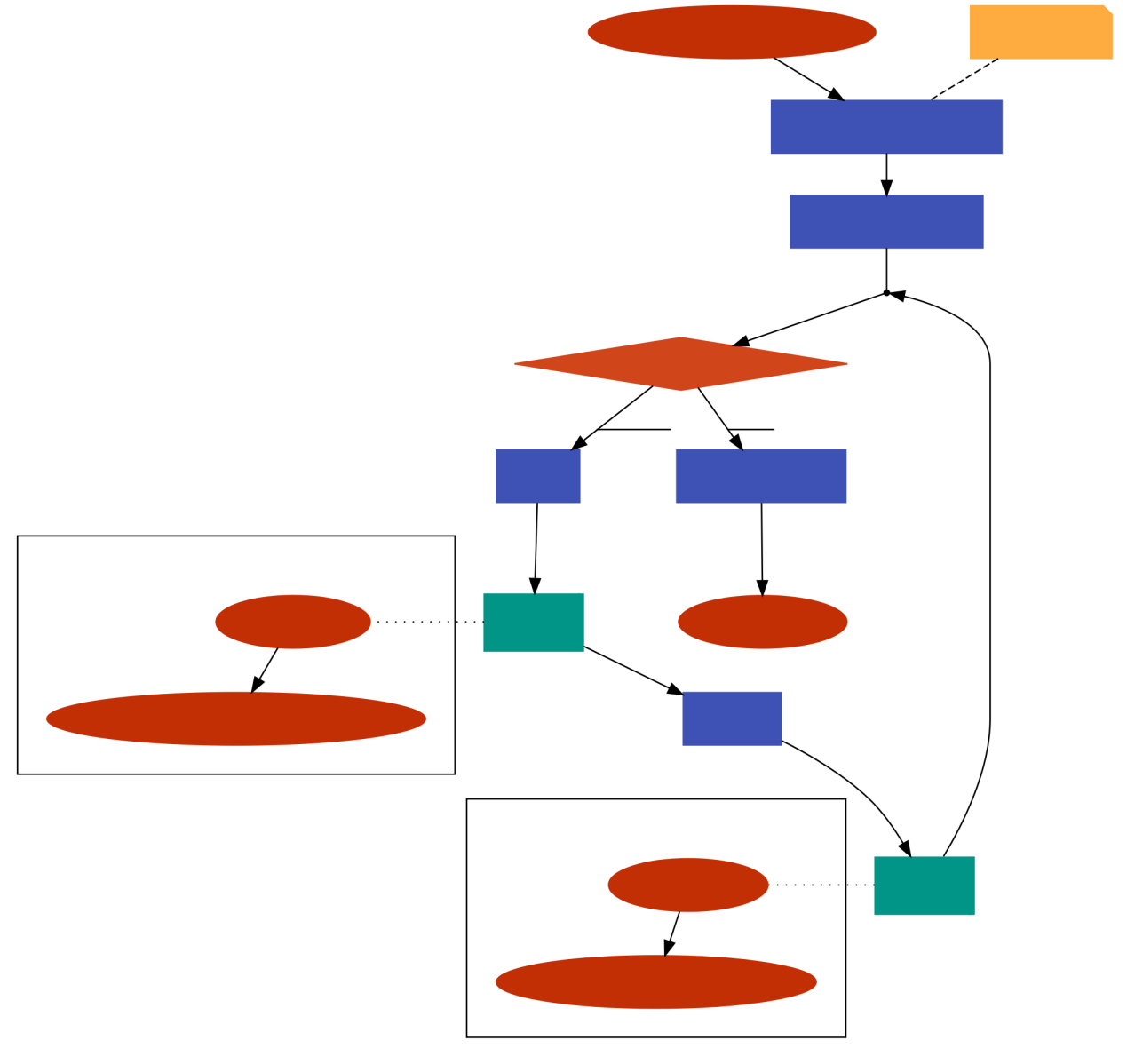


图4.3 多设备BFS流程图

多设备的BFS主要是在, 并行运行每一个单设备的BFS的基础上, 加入必要的同步完成的. 在单设备BFS过程中, 主要有gather和contract过程. 得益于原设计中良好的并行度, 我们只需将GPU共享内存上的算法, 改变为分布式共享内存上的算法即可.

在具体实现上, 如图4.3所示. 我们在每一个设备完成gather操作后, 调用multidev\_bfs\_data的iteration\_barrier\_1方法, 令调用线程等待其他线程操作完成. 随后, 在多设备间合并所需的数据结构, 并在完成后继续进行后续任务. 同样, 在每一个设备完成contract操作后, iteration\_barrier\_2方法再次进行同步, 并合并results和中间数据结构, 判断BFS过程是否完成.

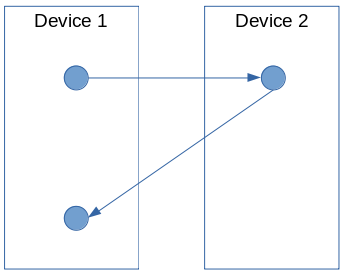


图4.4 多设备BFS问题样例示意图

值得注意的是, 如果BFS过程没有完成, 即使某一个设备的子图上已经没有任务了, 单个设备也不能提前结束自己的工作线程. 如图4.4所示的情形中, 如果Device1在完成第一次BFS后发现本设备上已经没有任务而退出. 那么当设备1上的节点再次被设备2引用时, 就会导致任务无人处理而出现错误.

## 本章小结

本章具体介绍了我们的多设备动态图更新系统的关键实现细节, 以及它如何同时利用CPU和GPU的计算资源, 进行图更新和图算法操作. 同时, 我们介绍了图更新和BFS过程中用到的具体串行与并行算法.

# 性能测试与分析

## 测试环境

我们首先对RAID软件进行了评估。测试平台配置为：CPU Intel Xeon 1-8GHz，512MB内存，LSI Logic 21320-R (1030) ULTRA320 SCSI HBA，连接4块Seagate Cheetah Ultra320磁盘（MODEL ST373307LC），一块Agilent 5221A FC HBA（2Gb），与主机（启动器）的Qlogic 2310F FC卡通过光纤连接。为评价磁盘阵列控制器最终性能，我们利用Iometer软件对阵列控制器进行了详细的性能评测，具体测试环境见**错误!未找到引用源。**：

表5‑1 光纤通道启动器与阵列控制器配置

|  |  |  |
| --- | --- | --- |
| 配 置 | 启动器 | 阵列控制器 |
| CPU | Xeon 2.0 | Intel 80321 |
| Memory | 512MB | 1GB |
| Fiber Channel HBA | Qlogic 23102G | Agilent DX2 |
| OS | Windows 2003 | ARM Linux |

## 功能测试

## 系统界面

不是必须

## 性能测试

其中阵列控制器挂接6个Maxtor STM3320820AS磁盘，利用6个磁盘分别构建RAID0、RAID10、RAID5三种不同的阵列级别，分条大小为128K字节，缓存大小为512MByte，LUN大小为500GByte，阵列控制器采取写回策略。



图5‑1 磁盘阵列控制器RAID5顺序读写IOPS性能

**错误!未找到引用源。**为6个磁盘构建RAID5时阵列控制器顺序读写IOPS(I/O Per Second)性能示意图，从图中可以看出，IOPS随着数据块大小变大而变小，当数据块大小为512Byte时，阵列控制器处理读请求的性能可以达到每秒15000个，写性能由于包含冗余校验信息计算，IOPS略低，约为12000，此指标高于同类美国阵列nStor 4502F，其写IOPS最大为7000左右。



图5‑2 磁盘阵列控制器RAID5读写性能

为6个磁盘构建RAID5时阵列控制器数据传输率示意图，从图中也可以看出，随着主机端I/O请求数据块大小逐渐增大，阵列控制器数据传输率逐渐增加，当数据块大小达到128Kbyte时，系统性能最佳，读性能可以达到185MByte/s，写性能约为167MByte/s。**（有坐标轴的图必须给出单位，坐标轴意义，所有测试图应该有详细分析）**

**（标准的流程图一定要有开始和结束当然有些程序没有结束，请注意流程图的画法，Yes/No分支应该清晰的标注在菱形分支附近，而不是很远，此图中少了一个N）**

**(论文中出现代码时请遵循以上代码格式，灰色底纹，斜体字，注意论文中不能出现过多的代码，只能出现一些关键数据结构。系统实现过程尽可能的用程序流程图表示。)**

（所有表必须插入题注，注意在表格的上方，编号形式为X.X，2级编号即可，不要3级编号，表格格式参照如下，原则上表不能跨页）

表3‑1 原型系统实现的几种类型的Region配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Type 0 | Type 1 | Type 5 | Type 7 |
| RAID级别 | 0 | 10 | 5 | N/A |
| 磁盘个数 | 8 | 8 | 8 | N/A |
| 分条单元大小 | 32KB | 16KB | 32KB | N/A |
| 数据块大小 | 512KB | 16KB | 64KB | 1KB |
| 数据块个数 | 8K | 8K | 8K | 8K |

表格由表号、表名、表头、表身等组成。表号按章编，如第2章的表为表2.1、表2.2、…，第3章的表为表3.1、表3.2、…等。表名是表格的名称，扼要概括表的内容，字数不宜太多。**表号、表名放在表的正上方**，相对于表体居中排版。表号及表名后不加标点。表头包括栏头、行头，与表身一起构成表格的主体。表中的竖称为栏，横格称为行。表身的内容，一般包括：数据、文字、公式和表图等。表内的数据对应位要对齐。少数表有表注，表注写在表下面且字号应比表号、表名的字小一号。

所有表格均应在正文中予以引用。引用某表格时，一般写为“…见表x.y”或“表x.y是…”。表格应尽量靠近正文的叙述，一般应先见文，后见表，表不跨节。表格允许转页。表格转页部分可以不写表号和表名，但要重复书写表头，并在表头右上角写“（续）”字标注。

从总体性能评测结果看，本文最终设计的FC-SATA磁盘阵列控制器基本发挥了Intel IOP处理器的最佳性能，同比性能优于同类进口磁盘阵列，基本达到了设计要求，如需要更高的数据传输率，必须同时提升主机通道以及IOP本身的性能，才能有实质性的提高。

## 本章小结

本章给出了控制器完整的软硬件设计方案，提出了主机板与通道板分离的架构，可以灵活构建全系列阵列控制器产品线；提出了磁盘阵列软件分层异步体系结构，可以有效提高I/O请求处理的并行度，减少I/O处理同步等待时间；最后提出了一种针对阵列层次性结构的细粒度测试方法，可单独测试评价各模块性能，为系统性能精确评测，以及故障诊断提供决策依据，最后给出了阵列控制器性能综合评测。

# 总结与展望

基于对象的存储是为了克服当前基于块的存储存在的诸多难题，在存储接口和结构层次的重要发展。基于对象的存储设备可以具有更多智能性，可以根据应用负载选择优化的存储策略。实现属性控制的存储策略，关键需要解决两个方面的问题：第一，属性表达什么信息，怎样影响存储策略？第二，属性怎样从用户应用传递给存储设备？围绕这两个问题，作了如下几点研究和工作：

1. 提出了对象属性控制存储策略的模型。OSD协议作为SCSI的扩展集，可以在TCP/IP网络和iSCSI协议之上来传输，基于对象的存储设备接收OSD命令并进行处理。
2. 结合对应用负载特征的分析，提出根据对象属性描述的负载特征动态选择RAID级别和分条单元大小的数据放置策略。I/O请求较小的对象放在RAID-10中，同时分条单元尺寸较小；反之则放在RAID-5中。
3. 建立了属性的传递机制，一种是应用程序接口静态传递；另一种是动态机制，即根据属性当中包含的文件系统信息预测文件的分类，并根据分类做出进一步的存储策略选择。
4. 论文总结必须用这样的条目的形式给出，总结是对全文的总结，不是对毕业设计的心得，心得部分不要写。

本文为更好地认识基于对象的存储以及OSD的实现进行了有益的探索。该领域还有很多研究工作有待完成，包括：

* 以OSD为节点构建大规模（PB级）存储系统
* 属性控制的caching/prefetching策略
* 实现Device-aware（表达设备能力，按需分配资源）
* 设备的学习能力，以文件系统和历史访问信息等为提示进行预测

随着存储需求的爆炸性增长，下一代互联网络必须有新的存储结构来应对构建和管理更大规模存储系统的挑战。基于对象的存储是最有希望的突破点之一。相信存储技术的发展一定能够满足人们不断增长的存储需求。

致 谢

论文完成之际，首先要感谢我的导师郭德纲教授。他站在学科发展的前沿，从论文的选题，研究工作逐步深入，到论文的撰写，都给我以细致的指导和建设性的意见，使我得以圆满而顺利地完成……。XXX严谨的治学态度、诲人不倦的师德和一丝不苟的工作作风将会给我留下不可磨灭的记忆……。

在…… 四年里，得到了许多老师和同学的大力帮助和支持，在此表示深深的谢意。感谢XXX等老师对我的关心和帮助。在课题研究和项目开发过程中，我与很多同学一起度过了一段段难忘的时光，值得怀念。我们大家共同创造的良好的学术氛围，将给我以永远而美好的回忆。

……

最后，我要深深地感谢我的父母，他们给予我无尽的关怀和无止境的爱，….

致谢属于论文的辅文部分。使用第一人称，采用散文体，对指导教师以及协助完成设计的有关人员表示谢意，并可简述自己通过本次毕业设计的体会，注意只写是查重最容易出问题的地方，请千万不要看别人写的，照搬。

# 毕业设计模板基本框架

2015年3月华中科技大学教务处发布了的《关于进一步加强本科生毕业设计（论文）规范化的通知》，该文对毕业设计开题报告，译文翻译，毕业设计论文的撰写提出了具体的规范化要求，该规范全部采用文字描述，实际操作有些困难，为方便计算机科学与技术学院全体学生撰写毕业设计论文，特开发此计算机科学与技术学院毕业设计标准模板供大家使用，本模板是对学校教务处标准的实例化，整体基本遵循学校标准，但也适当进行美化，如与学校教务处标准有冲突，请以此模板为主。

## 封面

本模板中封面中所有需要用户填写部分全部采用插入文档部件中的文档属性完成，如图7‑1中的姓名显示为作者，在开题报告中、成绩评定页中均包括学生姓名，采用文档属性的方式可以做到一次修改多处同时变化，避免数据重复输入和不一致性，注意不要删除文档属性框另外输入，日期部分不需要修改，会自动更新为正确时间。



图7‑1 封面模板中的文档属性

## 原创性声明页

最终定稿的毕业设计论文需要提交正式胶印版一份，双面黑白打印，这个版本需要将所有签字完成，其他未定稿的版本均不需要签字。学位论文原创声明页需要作者本人签名两次，提交导师签字之前请学生本人将自己的签名完成，再找导师签字，否则非常不礼貌。

## 摘要

中文摘要是对论文内容的高度概括，应用精练的语言概述论文的主要研究内容、目的意义、设计过程、实验手段及取得的成果等。摘要一般分为2~3段，第一段简要介绍背景知识，尽量简洁，切中要害，不要说些任何人都知道的无信息量的语句；后面可以用一到两段介绍毕业设计工作，主要体现自己的工作，摘要不要太长，但主要工作部分应该比第一段背景知识长，不要本末倒置。

摘要中不得出现“本文共有X章，第一章…，第二章…”之类的表述。摘要严禁出现“本文”，“我”，“我们”这样的第一人称主语，尽量采用动宾结构，比如设计了….实现了…利用了….技术…..进行了….实验…..实验结果表明……，如果需要设置主语可以用文中涉及的系统或研究机制等。

摘要不要超过一页，关键词与摘要在同一页，数量3～7个。中文关键词须用汉字，尽量不使用英文单词或其缩写，例如“DBMS”不能作为中文关键词，必须用对应的中文表述：“数据库管理系统”。关键词之间用逗号分隔，最后一个关键词后不用标点。关键词应该具体，不得用过于泛化的词做关键词，如“音乐、视频”，形容词不能作关键词。

## 目录

目录不会自动更新，排版变动后必须更新目录，更新目录方式：鼠标点击目录，按F9键或者右键更新域，选择更新整个目录即可。

目录仅包含一级标题和二级标题和标题样式，目录是全论文的纲要。中文摘要、Abstract、论文正文的各级标题（不包括第三级）、致谢、参考文献、附录等都应编入目录，标注其页码对照关系，但目录本身不出现在其中。中文摘要、Abstract、目录等使用希腊数字“I、II、…”编连续页码；论文正文、致谢、参考文献、附录等使用“1，2，3，… ”编连续页码。

## 参考文献

参考文献必须25篇论文以上，尽量避免网页链接引用。英文10篇以上。

// 讲个故事, 只要把参考文献的参考文献给他复制一遍, 250篇都很轻松

## 附录

附录部分非必须内容，如果有一些需要特别交代的内容可以在这里列出，如本人参加的项目，提交的论文等，如没有特别闪光的地方建议删除本章。

## 毕业设计任务书

### 任务书封面

毕业设计任务书封面内容均可根据本模板封面数据自动生成，无需没写，但如果题目超过两行，请自行输入题目到两行中，避免出现图7‑2中尴尬的情况，该封面中所有时间都已经固定好，不需要修改。



图7‑2 任务书封面标题过长

### 任务书内容

课题内容和要求请按毕业设计互选系统中的内容酌情修改完善后填充，内容尽可能丰满完善一点，尽可能的细化，不要干瘪的一句话，参考文献部分应列出最终论文中最重要的5~6篇参考文献，尽可能的将表格充满，注意列表格式应与原格式一致。如有同组设计者请填写姓名，没有填无。

### 任务书签字流程

任务书封面部分需要指导老师所在单位教学负责人签字审查，并经教学院长批准，请各班集中收集任务书封面单页后，到教务科集中签字盖章审核，为减少学生和相关负责人工作量，原则上不针对单个学生进行该项业务。

计算科学理论研究所： 石柯 并行分布式计算研究所： 陆枫

信息安全研究所： 付才 数字媒体研究所： 李丹

数据存储研究所： 谭志虎 数据工程研究所： 潘鹏

## 成绩评定页

成绩评定页需要导师和答辩组就论文和答辩情况进行综合评定，参照硕士论文答辩流程，请学生**事先草拟导师意见**，提交导师参考手写到评定页，评分并签字。**答辩时请携带答辩评审意见空白页**（含姓名，班级，打印论文时单独取出，不要装订）请**答辩评议组当场给出评审意见**并签字打分。

# 毕业设计撰写要求

## 图的格式



图8‑1 图注必须在图的正下方

正文中所有插图要求图面整洁，布局合理，线条粗细均匀，大小适中。插图必须是矢量图，不能是从他处直接拷贝位图，所有曲线、图表、线路图、流程图、程序框图、示意图必须按国家规定标准或工程要求采用visio等工具绘图后复制到word。插图在论文中采用的是正文样式，所以会自动缩进，请调整标尺取消缩进，让插图严格居中，图表中的文字大小必须小于正文字号。

**TODO: 把图变成矢量图**

### 图的题注

所有插图均应有图号和图名（office中称题注，图注），图注出现在插图的正下方，小四黑体居中，在本模板中可以直接使用样式库中的题注样式即可规范格式，图号和图名后面不加标点符号，也不得加参考文献引用。

图号按章编编号，如第2章的图为图2-1、图2-2、…，第3章的图为图3-1、图3-2、…等，第一个编号是章号，第二个编号是本章的图的序号，常见错误是出现二级编号如：图2-1-1；第二个图名是插图的名称，扼要概括图的内容，字数不宜太多。

图的题注可在引用菜单栏通过插入题注的方式插入**（最简单的方法是复制其他图的图注然后更新域即可）**，引用时采用交叉引用引用，注意题注，交叉引用是word很重要的概念，是office排版的高级技巧，撰写毕业设计论文时应研究学习一下此功能，如果插入题注没有对应的图题注，只有figure等，可以自己创建，具体流程如图 8‑2，图 8‑3所示，需要注意的是，由于office不是为中文排版设计，图编号前面会增加一个空格，建议手工删除。

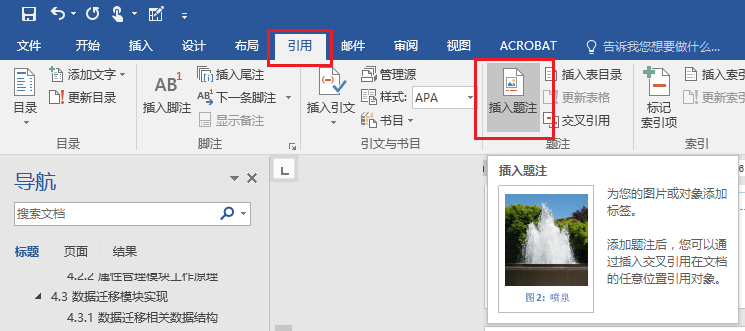


图 8‑2 插入图的题注

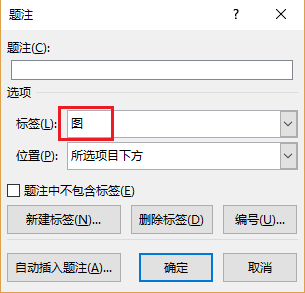


图 8‑3 插入题注对话框

### 图的引用

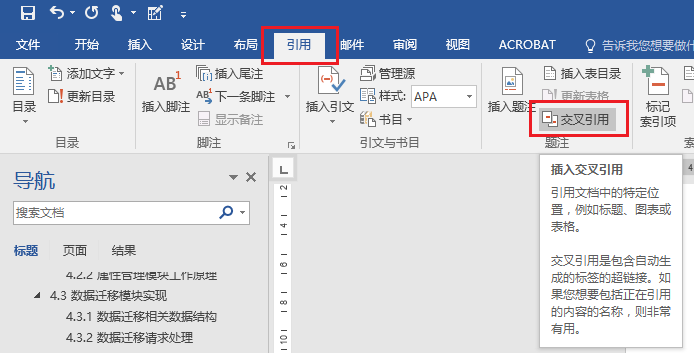


图8‑4 交叉引用

所有插图均应在正文中予以引用。引用某插图时，一般写为“…如图2-1”或“图3-2是…”。正文中的插图均须安排在文中第一次引用到该图的正文下面，一般要求先见文，后见插图，且图一般不跨页绘制。

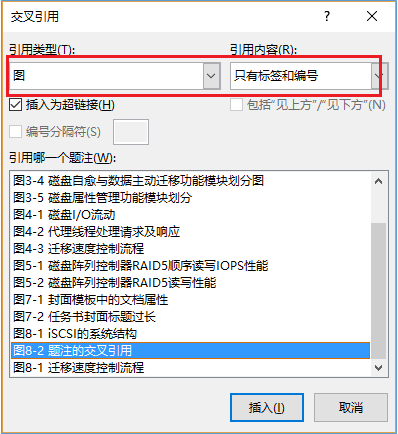


图8‑5 插入交叉引用对话框

手工输入图引用的方式非常不方便，如果图编号发生更改要全文修改编号，Office中提供交叉引用的功能进行题注的自动引用，方便编号自动更新，如图8‑4所示，点击交叉引用后得到图8‑5所示对话框，选择合适的图编号即可，注意引用内容选择只有标签和编号。

### 错误的插图格式

图8‑6所示图的排版中插入时未取消左侧的格式缩进，右侧超出了页边距，专业术语称之为“出血”现象，请一定要仔细核对插图以及图注的缩进问题，插图和图注应该严格居中，插图不得超出页边距。另外文章红色字体过大，一般也是有问题，建议所有图标中的文字字体均小于正文字体。

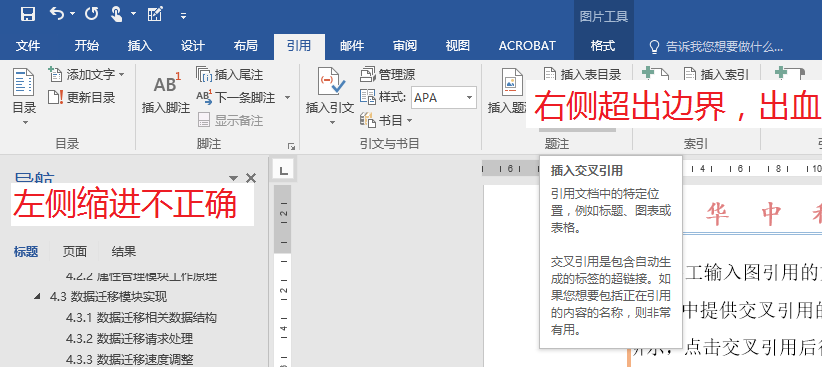


图8‑6 缩进错误，出血错误实例

不得出现下面这样大段的空行或留白现象，每章最后一节除外。

## 表的格式要求

表的格式要求基本和图一致，注意图表中的文字均需要小于正文字体大小，与图注位置相反，表的题注必须在表的正上方。

表8‑1 表头必须在表的正上方

|  | Type 0 | Type 1 | Type 5 | Type 7 |
| --- | --- | --- | --- | --- |
| RAID级别 | 0 | 10 | 5 | N/A |
| 磁盘个数 | 8 | 8 | 8 | N/A |
| 数据块大小 | 512KB | 16KB | 64KB | 1KB |
| 数据块个数 | 8K | 8K | 8K | 8K |

所有表均应有表号和表名（office中称题注，表注），表注出现在表的正上方，小四黑体居中，在本模板中可以直接使用样式库中的题注样式即可规范格式，表号和表名后面不加标点符号，也不得加参考文献引用。

表格由表号、表名、表头、表身等组成。表号按章编，如第2章的表为表2-1、表2-2、…，第3章的表为表3-1、表3-2、…等。表名是表格的名称，扼要概括表的内容，字数不宜太多。

表头包括栏头、行头，与表身一起构成表格的主体。表中的竖称为栏，横格称为行。表身的内容，一般包括：数据、文字、公式和表图等。表内的数据对应位要对齐。少数表有表注，表注写在表下面且字号应比表号、表名的字小一号。

所有表格均应在正文中予以引用，具体引用方式和图的引用方式类似，如图8‑7所示。引用某表格时，一般写为“…见表1-2”或“表3-1是…”。表格应尽量靠近正文的叙述，一般应先见文，后见表，表不跨节。表格允许转页。表格转页部分可以不写表号和表名，但要重复书写表头，具体设置如图8‑8所示。

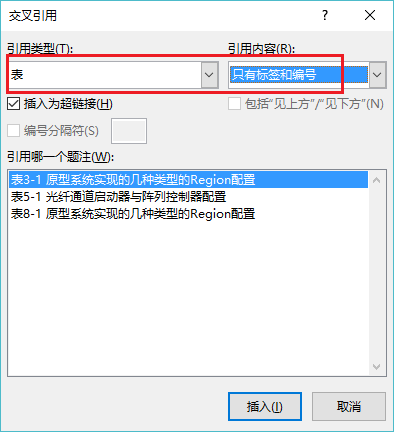


图8‑7 插入表注对话框

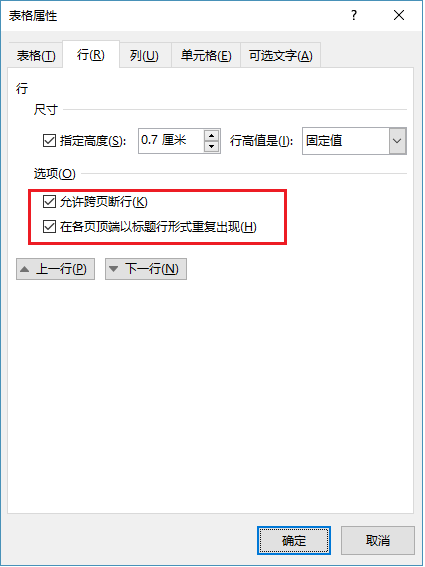


图8‑8 表格跨行设置

## 公式

常见公式排版格式如下：

(8‑1)

公式必须进行标注，公式中的变量必须用斜体，标注要居右，公式居中，要做到这一点可以在标号前面加空格进行调整。

公式一般另行居中写，公式末不加标点。若公式前有文字，如例、解、证、假定等，文字顶格写，公式仍居中写。一行写不下时，公式允许转行。公式转行需处理得当，做到既意义正确，又使版面美观匀称。

公式要有编号，公式编号按章编，如第2章的公式为(2-1)、(2-2)、…，第3章的公式为(3-1)、(3-2)、…等。公式编号写在公式右侧行末顶边线，并加圆括号。模板中公式编号已经按照图题注的方式给出，可以直接复制然后更新域自动排序。。

公式一般应在正文中予以引用，引用时以公式编号指示公式。正文中常有公式中表示量的符号说明，采用“式中”二字作为标志。一般可写成接排形式，如“式中，A指……；B指……”。

## 流程图



图8‑9 迁移速度控制流程

（标准的流程图一定要有开始和结束当然有些程序没有结束，请注意流程图的画法，Yes/No分支应该清晰的标注在菱形分支附近，而不是很远。

## 常见格式问题

|  |
| --- |
| 未定义标签 |
| 各章之间未分页 |
| 标题格式不正确 |
| 章节标题不合适 |
| 缩进不规范 |
| 行距不统一 |
|  |
|  |
| 不必要的留白和空行 |
| 图表中字体明显大于正文 |
| 图表比例失调，甚至有出血现象 |
| 表注不规范 |
| 表格排版欠美观 |

参考文献

1. Palkar S, Lan C, Han S, et al. E2: a framework for NFV applications[C]. Symposium on Operating Systems Principles. ACM, 2015:121-136.
2. Bremler-Barr A, Harchol Y, Hay D. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions[C]. Conference on ACM SIGCOMM 2016 Conference. ACM, 2016:511-524.
3. Sekar V, Egi N, Ratnasamy S, et al. Design and implementation of a consolidated middlebox architecture[C]. Usenix Conference on Networked Systems Design and Implementation. 2011:24-24.
4. Anderson J W, Braud R, Kapoor R, et al. xOMB: extensible open middleboxes with commodity servers[C]. Eighth ACM/IEEE Symposium on Architectures for NETWORKING and Communications Systems. ACM, 2012:49-60.
5. A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual Middleboxes as First-class Entities[R]. Technical report, UW-Madison 2012.
6. Zhang W, Liu G, Zhang W, et al. OpenNetVM: A Platform for High Performance Network Service Chains[C]. The Workshop on Hot Topics in Middleboxes and Network Function Virtualization. ACM, 2016:26-31.
7. Gember-Jacobson A, Viswanathan R, Prakash C, et al. OpenNF: enabling innovation in network function control[C]. ACM Conference on SIGCOMM. ACM, 2015:163-174.
8. Rajagopalan S, Dan W, Jamjoom H, et al. Split/merge: system support for elastic execution in virtual middleboxes[C]. Usenix Conference on Networked Systems Design and Implementation. 2013:227-240.
9. J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella. Paving the Way for NFV: Simplifying Middlebox Modifications Using StateAlyzr[C]. In Proc. of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI’16), 2016.
10. Ballani H, Costa P, Gkantsidis C, et al. Enabling End-Host Network Functions[J]. ACM SIGCOMM Computer Communication Review, 2015, 45(4):493-507.
11. Paxson V. Bro: a system for detecting network intruders in real-time[C]. Conference on Usenix Security Symposium. USENIX Association, 1998:3-3.
12. Yoan N. Iptables[M]. Miss Press, 2013.
13. Sherry J, Gao P X, Basu S, et al. Rollback-Recovery for Middleboxes[C]. ACM Conference on Special Interest Group on Data Communication. ACM, 2015:227-240.
14. Chrome G. HTTP persistent connection[J]. 2015.
15. Surhone L M, Tennoe M T, Henssonow S F, et al. Ffmpeg[M]. 2010.
16. Housley R, Hoffman P. Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP[J]. Ietf Rfc Sri Network Information, 1999, 11(3):82--89.
17. Maheshwari A, Sharma A, Ramamritham K, et al. TranSquid :Transcoding and Caching Proxy for Heterogenous E-Commerce Environments[C]. International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/e-Business Systems. IEEE Computer Society, 2002:50.
18. J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella. Paving the Way for NFV: Simplifying Middlebox Modifications Using StateAlyzr[C]. In Proc. of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI’16), 2016.
19. NFV Paper [EB/OL].https://portal.etsi.org/NFV/NFV White Paper2.pdf,2017-05-05.
20. Hwang J, Ramakrishnan K K, Wood T. NetVM: high performance and flexible networking using virtualization on commodity platforms[C]. Usenix Conference on Networked Systems Design and Implementation. USENIX Association, 2014:445-458.
21. S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy. SoftNIC: A Software NIC to Augment Hardware[R]. Technical report, EECS Department, University of California, Berkeley, 2015..
22. Martins J, Ahmed M, Raiciu C, et al. ClickOS and the art of network function virtualization[C]. Usenix Conference on Networked Systems Design and Implementation. USENIX Association, 2014:459-473.
23. Panda A, Han S, Jang K, et al. NetBricks: taking the V out of NFV[C]. Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2016:203-216.
24. A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual Middleboxes as First-class Entities[R]. Technical report, UW-Madison 2012.
25. Palkar S, Lan C, Han S, et al. E2: a framework for NFV applications[C]. Symposium on Operating Systems Principles. ACM, 2015:121-136.
26. Rajagopalan S, Dan W, Jamjoom H. Pico replication:a high availability framework for middleboxes[C]. Symposium on Cloud Computing. 2013:1-15.
27. Sherry J, Gao P X, Basu S, et al. Rollback-Recovery for Middleboxes[C]. ACM Conference on Special Interest Group on Data Communication. ACM, 2015:227-240.
28. Qazi Z A, Tu C C, Chiang L, et al. SIMPLE-fying middlebox policy enforcement using SDN[J]. Computer Communication Review, 2013, 43(4):27-38.
29. Zhang W, Hwang J, Rajagopalan S, et al. Flurries: Countless Fine-Grained NFs for Flexible Per-Flow Customization[C]. International on Conference on Emerging NETWORKING Experiments and Technologies. ACM, 2016:3-17.
30. Bremler-Barr A, Harchol Y, Hay D. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions[C]. Conference on ACM SIGCOMM 2016 Conference. ACM, 2016:511-524

附录：大学期间发表或提交的论文

1. A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual Middleboxes as First-class Entities[R]. Technical report, UW-Madison 2012.
2. Palkar S, Lan C, Han S, et al. E2: a framework for NFV applications[C]. Symposium on Operating Systems Principles. ACM, 2015:121-136.

这老哥真有意思, 这两篇论文的作者名字, 没有一个重合的. 这老哥改名了吧

此页以后的内容为装订材料！先仔细阅读！

撰写论文时此页以后的内容全部删掉！

装订毕业论文时应将任务书原件装订到最后



**本科生毕业设计任务书**

|  |  |
| --- | --- |
| 题 目 | 宇宙超级无敌XXX系统的设计与实现 |
|  |  |

（任务起止日期：2020年1月1日～2020年6月1日）

|  |  |
| --- | --- |
| 院 系 | 计算机科学与技术 |
| 专业班级 | 卓越1601 |
| 姓 名 | 小岳岳 |
| 学 号 | U201315102 |
| 指导教师 | 郭德纲 |

教研室（系、所）负责人 2020年1月6日 审查

院（系）负责人 2020年1月9日 批准

|  |
| --- |
| 课题内容 |
| 实现一个简单的分布式网络系统，实现文件的分布式网络化存储。，实现文件的分布式网络化存储。，实现文件的分布式网络化存储。，实现文件的分布式网络化存储。，实现文件的分布式网络化存储。，实现文件的分布式网络化存储。，实现文件的分布式网络化存储。，实现文件的分布式网络化存储。 |
| 课题任务要求 |
| 研究学习文件系统的相关原理及实现研究分布式系统概念及应用研究网络文件系统的原理、架构及实现。研究学习文件系统的相关原理及实现研究分布式系统概念及应用研究网络文件系统的原理、架构及实现。研究学习文件系统的相关原理及实现研究分布式系统概念及应用研究网络文件系统的原理、架构及实现。 |
| 主要参考文献 |
| 1. 董晓明，谢长生．基于对象的进化存储系统研究．计算机科学，2005，32(11): 223~226 2. 庞丽萍编．操作系统原理（第二版）．武汉：华中理工大学出版社，1994.9．225~270 3. （美）Nils J. Nilsson著；郑扣根等译．人工智能（Artificial Intelligence）．北京：机械工业出版社，2000.9．177~194 4. （美）Tom Mitchell著；曾华军等译．机器学习（Machine Learning）．北京：机械工业出版社，2003.1．38~56 5. 谢长生，董晓明，万继光，谭志虎，刘瑞芳．磁盘阵列控制器的设计与原型实现．小型微型计算机系统，2006, 27(1): 173~176 6. （美）Nils J. Nilsson著；郑扣根等译．人工智能（Artificial Intelligence）．北京：机械工业出版社，2000.9．177~194 |
| 同组设计者 |
| 无 |

**成 绩 评 定**

**指导教师评定意见**

装订论文时应将成绩评定页放最后，导师评语放前面！

一、对毕业设计（论文）的学术评语（应具体、确切、实事求是）

|  |
| --- |
|  |

二、对毕业设计评分

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 调研论证  (10分) | 外文翻译  (5分) | 设计(论文)撰写质量  (10分) | 学习态度  (10分) | 基本理论和基本技能  (50分) | 创 新  (15分) | 合 计  (100分) |
| 得分 | **8** | **4** | **8** | **8** | **40** | **12** | **80** |

指导教师签字：**郭德纲** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 年 月 日

**答辩小组评定意见**

答辩小组意见和综合评定页放最后，双面！

一、评语（根据学生答辩情况及其设计（论文）质量综合评价）

|  |
| --- |
|  |

二、评分

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 答 辩 情 况 | | 论 文 质 量 | | 合 计  (100分) |
| 答辩情况  (15分) | 回答问题情况  (25分) | 规范要求与文字表达  (20分) | 学术水平  (40分) |
| 得分 |  |  |  |  |  |

答辩小组长签字：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 年 月 日

**毕业答辩及成绩评定说明**

1. 毕业答辩
2. 答辩前，答辩小组应详细审阅每个学生的毕业设计（论文），为答辩做好准备。
3. 严肃认真组织答辩，开好答辩会。
4. 指导教师应参加所指导学生的答辩会，但评定其成绩时宜回避。
5. 答辩中要做好记录以供成绩评定时参考。
6. 成绩评定
7. 答辩前每个学生都要将自己的毕业设计（论文）在指定时间内交给指导教师，由指导教师审阅，写出评语并预评分。
8. 答辩工作结束后，答辩小组应举行专门会议进行讨论，在参考指导教师预评结果的基础上，结合学生毕业设计（论文）质量和学生答辩情况，综合评定每个学生的成绩。
9. 院（系）对专业答辩小组提出的优秀和不及格的毕业设计（论文），要组织院（系）级答辩，最终确定成绩，并向学生公布。
10. 各专业学生的最后成绩应符合正态分布规律。
11. 请用蓝、黑钢笔手写或五号宋体字编辑，签名须手写，A4纸双面打印。

**毕业设计（论文）成绩评定**

|  |
| --- |
| 班号：**卓越1601** 学生姓名：**小岳岳**  综合成绩：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_分（折合等级\_\_\_\_\_\_\_\_\_\_\_\_\_\_）  答辩小组长（签名）：\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 年 月 日 |