代　　号	10701	学　　号	0911120728
分 类 号	TP391.41	密　　级	公开

题 (中、英 文) 目 云计算平台的任务资源调度调度的设计与实现

The Desgined and Implentments of Scheduler In Cloud Computing Plantform

作 者 姓 名	方祯	**指导教师姓名、职务**	马建峰 教授
学 科 门 类	工学	**学科、专业**	计算机系统结构

提交论文日期 二〇一四年十二月

西安电子科技大学
学位论文独创性（或创新性）声明

秉承学校严谨的学风与优良的科学道德，本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别中以标和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得西安电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切的法律责任。

本人签名：_____　　日期_____

西安电子科技大学
关于论文使用授权的说明

本人完全了解西安电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属西安电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律署名单位为西安电子科技大学。

(保密论文在解密后遵守此规定)

本学位论文属于保密，在____ 年解密后适用本授权书。

本人签名：_____　　导师签名：_____

日　　期 _____　　日　　期 _____

摘 要

图像是多媒体信息时代的主要数字信息资源。如何从海量的图像数据中迅速而准确地搜寻到我们所需的信息成为研究热点。...。本文的主要研究内容包括以下几部分：

(1) 针对人类视觉系统更加关注视觉信息丰富的图像区域，提出基于视觉信息量的图像显著性检测算法。首先根据图像像素间的相关性，度量图像内容的视觉冗余程度；接着，根据像素的分布特性，计算图像内容的信息熵；然后，从信息熵中去除图像的视觉冗余，获得图像内容的视觉信息量；最后，采用视觉信息量来度量图像显著性，从而建立显著性检测模型。实验结果表明提出的基于视觉信息熵的图像显著性检测算法能准确检测出任何潜在图像特征下的显著性内容，且所得显著图与主观视觉关注区域高度吻合。

(2) 针对人类视觉系统对具有规则结构的图像区域高度敏感，提出基于结构自相似性的恰可识别失真阈值估计算法。视觉系统非常善于提取图像的结构信息，并通过结构比对及模式匹配来理解图像内容，因此视觉系统对具有自相似结构区域分辨能力强。根据相邻像素间的相似性，首先度量图像内容的结构自相似程度；然后，根据结构自相似性提出新的空域掩膜方程；最后，结合现有的亮度敏感度方程和所提空域掩膜方程，建立恰可识别失真阈值估计模型。实验结果表明，所提算法能准确估计纹理区域的恰可识别失真阈值，而现有其它算法无法准确估计该区域。

（3）...

（4）...

（5）...

上述研究成果从主观视觉感知的角度对图像处理进行分析与研究，具有一定的前瞻性和挑战性。本文在理论分析上取得一些突破，在技术实现上具有一些创新，为基于主观视觉感知的客观图像处理开辟了新的思路，具有重要的理论意义及实用价值。

关键词： **人类视觉系统 视觉关注 恰克识别失真阈值 结构自相似 视觉敏感度 不确定信息 图像质量评价**

ABSTRACT

In the multimedia era, image information plays a very important role in our daily life...The main contributions of this thesis can be summarized as follows:

(1) According to that the HVS pays more attention to these regions with abundant visual information, we introduce a visual information based saliency detection model. Firstly, the visual redundancy of an image is measured based on the correlations among pixels. And according to the distribution of pixels, the entropy of the image is computed. Then, by removing the visual redundancy from the entropy, the quantity of visual information of the image is acquired. Finally, the visual information is used to estimate the saliency of the image. Experimental results show that the proposed saliency detection model can accurately estimate the salient object under any potential image feature, and the saliency map from the proposed model is highly coincided with the subjective visual attention.

(2) According to that the HVS is highly sensitive to regular regions, we introduce a structural self-similarity based JND threshold estimation model. The HVS is highly adapted to extract structural information for image perception and understanding, and the HVS is highly sensitive to these regions with self-similar structures. According to the similarities among nearby pixels, we firstly measure the structural self-similarity of an image. And then, a novel spatial masking function is introduced based on structural self-similarity. Finally, combining the existing luminance adaptation function and the proposed spatial masking function, a new JND threshold estimation model is built. Experimental results demonstrate that the proposed JND model can accurately estimate the JND threshold of the textural region, where the existing JND models always failed.

(3) ...

(4) ...

(5) ...

The results above are image processing researches from the perspective of subjective visual perception, which are forward looking and full of challenges. This thesis has some breakthrough in theory and some innovation in technology. This work opens up a new way for visual perception based image processing, which has extremely important theoretical significance and application value.

Keywords: **The Human Visual System, Visual Attention, Just Noticeable Distortion, Structural Self-Similarity, Visual Sensitivity, Uncertainty, Image Quality Assessment**

目 录

第一章 引言

1.1 研究的背景及意义

1.2 国内外研究现状

1.3 本论文的研究内容

1.4 本论文的组织结构

第二章 云计算资源管理与调度相关技术

对于企业和公司，为了完成各种对外的服务以及公司内部业务逻辑，需要大量的硬件资源, 而硬件的资源的代价往往比较昂贵，所以如何充分挖掘硬件资源潜力从而增加硬件的利用率

2.1　OpenStack 云计算平台

OpenStack 是一个美国国家航空航天局和 Rackspace 合作研发的，以 Apache 许可证授权，并且是一个自由软件和开放源代码项目。

OpenStack 是一个云平台管理的项目，它不是一个软件。这个项目由几个主要的组件组合起来完成一些具体的工作。

OpenStack 是一个旨在为公共及私有云的建设与管理提供软件的开源项目。它的社区拥有超过 130 家企业及 1350 位开发者，这些机构与个人都将 OpenStack 作为基础设施即服务（简称 IaaS）资源的通用前端。OpenStack 项目的首要任务是简化云的部署过程并为其带来良好的可扩展性。OpenStack 的核心组件有以下 9 个:

- 计算 (Compute):Nova，Nova 是 OpenStack 云中的计算组织控制器。支持 OpenStack 云中实例（instances）生命周期的所有活动都由 Nova 处理。其中的调度器 nova-schedule 作为一个守护进程运行，通过恰当的调度算法从可用资源池获得一个计算服务。nova-scheduler 会根据诸如负载、内存、可用域的物理距离、CPU 构架等作出调度决定。
- 对象存储 (Object):Swift，其最初是由 Rackspace 公司开发的高可用分布式对象存储服务，并于 2010 年贡献给 OpenStack 开源社区作为其最初的核心子项目之一，为其 Nova 子项目提供虚机镜像存储服务。Swift 支持多租户模式、容器和对象读写操作，适合解决互联网的应用场景下非结构化数据存储问题。
- 镜像 (Image):Glance, 用来管理在 OpenStack 集群中的镜像，但不负责实际的存储。它为从简单文件系统到对象存储系统 (如 OpenStack-Swift 项目) 的多种存储技术提供了一个抽象。除了实际的磁盘镜像之外，它还保存描述镜像的元数据和状态信息。
- 身份 (Identity):Keystone(OpenStack Identity Service) 是 OpenStack 框架中，负

责身份验证、服务规则和服务令牌的功能，它实现了 OpenStack 的 Identity API。

- 网络地址管理 (Network):Neutron 是 OpenStack 核心项目之一，提供云计算环境下的虚拟网络功能.
- 块存储 (Block):Cinder,Cinder 用来提供块存储 (Block Storage)，类似于 Amazon 的 EBS 块存储服务，OpenStack 中的实例是不能持久化的，需要挂载 volume，在 volume 中实现持久化。Cinder 就是提供对 volume 实际需要的存储块单元的实现管理功能。
- UI 管理界面 (Dashboard):Horizon,Horizon 套件提供 IT 人员一个图形化的网页接口，让 IT 人员可以综观云端服务目前的规模与状态，并且，能够统一存取、部署与管理所有云端服务所使用到的资源。
- 测量 (Metering):Ceilometer, 主要负责监控数据的采集，采集的项目包括虚拟机的性能数据，n eutron-l3-router 使用的网络带宽，glance，cinder，swift 等租户使用信息，甚至是通过 snmp 采集物理机的信息，以及采集支持 opendaylight 的网络设备信息。
- 编配 (Orchestration):Heat 类似于 AWS 的 CloudFormation, heat 实现了一种自动化的通过简单定义和配置就能实现的云部署方式。可以在 heat 模板中定义连串相关任务，然后交由 heat，由 heat 按照一定的顺序执行 heat 模板中定义的一连串任务。利用 heat 还可以连接到 neutron 来帮助编排负载均衡和其他网络功能。

2.2 资源管理抽象模型

资源管理调度模型如图所示

图中说明了一个资源管理与系统的抽象模型. 从概念上讲资源管理与调度系统的主要目的是根据用户任务的请求, 从集群中分配资源. 目前资源分配的主要资源主要包括内存,CPU, 网络和 IO 资源等. 而资源调度的模型主要涉及 3 个要素资源组织模型, 调度算法和任务组织方式.

1. 资源组织模型主要是整理和规整集群中所有的资源, 以方便后续的资源分配过程. 通常的做法是将资源组织成为多层级队列的形式, 例如 Corona 的”All-resource->group->pool” 的三级队列模式组织, 另外, 平级队列或是单队列也是常见的资源组织模型, 其本质是多层级队列的特殊形式.
2. 资源调度算法的负责将集群中的资源按照一定的策略分配的任务. 常见的调度策略有 FIFO 调度算法, 公平调度算法, 能力调度算法以及延迟调度算法等,

具体策略可以参加本章 2.4 资源调度算法.

3. 任务组织方式主要是将多用户提交的任务通过一定的方式组织起来, 来方便调度算法进行分配. 常见的组织方式是将任务以队列的形式组织起来, 例如 Hadoop1.0 中将任务按照多队列组织, 队列之间采用平级关系, 而在 hadoop2.0 中的任务队列则增加了层级队列的树形结构, 用以提供更加灵活的方式来管理任务队列.

2.3　资源管理与调度系统范型

目前有多种多种的资源管理和调度系统实现, 根据实际的宏观运行机制进行分类, 可以分为以下几种资源管理与调度系统范型: 集中式调度, 两级调度器和状态共享调度器.

2.3.1 集中式调度器

集中式调度器在整个系统中只有一个唯一的全局调度器, 其特点是，资源的调度和作业的管理功能全部放到一个进程中完成，在集群上运行的所有框架或者是计算任务的资源请求均由集中式调度器来满足, 因此, 整个调度系统在集群上缺乏并发性, 并且所有的调度逻辑均由集中式调度器独自完成. 开源界典型的代表是 Hadoop JobTracker 的实现。

这类调度器又分为 2 种类型, 一种是单路径调度器, 另一种是多路径调度器.

1. 单路径调度器 (Single Path Scheduler): 这种调度器是指不管任务的类型均采用全局统一的调度器进行资源管理, 这种调度算法基本是采用融合考虑各种因素来实现的调度器, 在此基础之上结合任务的优先级, 集群资源的状态统计, 决定调度的任务资源分配和调度顺序.
2. 多路径调度器 (Multi Path scheduler): 这种调度器在单路经调度器的基础之上做了该进, 首先支持多种调度策略, 其大致思路是将调度算法模块化, 根据任务的类型进行调度, 比如批处理的任务采用批处理的调度算法, 计算量大的任务采用计算量大的调度算法, 在具体的实现的时候各个算法独自实现, 而在调度器的逻辑中根据任务类型进行选择, 其选择类型类似于程序语言的 Switch-Case 分支路径, 这种调度器可以根据调度算法的类型采用多线程的方式进行实现, 较单路经调度器的调度灵活性和并发性有了一定的提高

但是集中式调度器的设计方式的缺点也是比较明显的.

1. 由于将所有的调度逻辑均写入中央调度器中, 所以实现的逻辑比较负载, 可扩展较差. 这点在单路径调度器上尤其明显, 多路径调度器虽然在此基础之上进

行了改进, 但是在针对某个类型任务的算法进行替换的过程中, 中央调度器此时不得不整体停止, 对其他的调度任务产生影响.

2. 集群的规模受限, 中央调度器的并发性不足. 由于是对整个系统的全局资源进行调度, 在小规模集群上可以完成其调度工作而当集群规模扩大时, 任务和工作负载加大, 当调度与资源分配决策执行时间较长的情况下, 这个时候往往调度器会首先达到工作饱和, 此时, 后续任务会花费大量的时间在等待被调度, 从而导致集中式调度器成为整个系统的运行瓶颈, 严重影响系统的运行性能.

2.3.2 两级调度器

为了解决中央式调度器的不足，双层调度器是一种很容易想到的解决之道(实际上是分而治之策略或者是策略下放机制)。双层调度器仍保留一个经简化的中央式调度器，但调度策略下放到各个应用程序调度器完成。这种调度器的典型代表是 Apache Mesos 和 Hadoop YARN。Omega 论文重点介绍了 Mesos，Mesos 是 twitter 开源的资源管理系统，它的详细设计架构我已在多篇博文中进行了介绍，在此简要介绍一下：Mesos 资源管理部分由两部分组成：分别是 Mesos Master 和 Mesos Slave，其中，Mesos Slave 是每个节点上的代理，负责向 Master 汇报信息和接收并执行来自 Master 的命令，而 Master 则是一个轻量级中央化的资源管理器，负责管理和分配整个集群中的资源。如果一个应用程序想通过 Mesos 资源管理系统申请和使用资源，需编写两个组件：框架调度器和框架执行器，其中，框架调度器负责从 Mesos Master 上获取资源、将资源分配给自己内部的各个应用程序，并控制应用程序的执行过程；而框架执行器运行在 Mesos Slave 中，负责运行该框架中的任务。当前很多框架可以接入 Mesos 中，包括 Hadoop、MPI、Spark 等。

双层调度器的特点是，各个框架调度器并不知道整个集群资源使用情况，只是被动的接收资源；Mesos Master 仅将可用的资源推送给各个框架，而框架自己选择使用还是拒绝这些资源；一旦框架（比如 Hadoop JobTracker）接收到新资源后，再进一步将资源分配给其内部的各个应用程序（各个 MapReduce 作业），进而实现双层调度。双层调度器的缺点是：1）各个框架无法知道整个集群的实时资源使用情况。很多框架不需要知道整个集群的实时资源使用情况就可以运行的很顺畅，但是对于其他一些应用，为之提供实时资源使用情况可以为之提供潜在的优化空间，比如，当集群非常繁忙时，一个服务失败了，是选择换一个节点重新运行它呢，还是继续在这个节点上运行？通常而言，换一个节点可能会更有利，但是，如果此时集群非常繁忙，所有节点只剩下小于 5GB 的内存，而这个服务需要 10GB 内存，那么换一个节点可能意味着长时间等待资源释放，而这个等待时

间是无法确定的。2）采用悲观锁，并发粒度小。在数据库领域，悲观锁与乐观锁争论一直不休，悲观锁通常采用锁机制控制并发，这会大大降低性能，而乐观锁则采用多版本并发控制 (MVCC ,Multi-Version Concurrency Control)，典型代表是 MySQL innoDB，这种机制通过多版本方式控制并发，可大大提升性能。在 Mesos 中，在任意一个时刻，Mesos 资源调度器只会将所有资源推送给任意一个框架，等到该框架返回资源使用情况后，才能够将资源推动给其他框架，因此，Mesos 资源调度器中实际上有一个全局锁，这大大限制了系统并发性。

2.3.3 状态共享调度器

为了克服双层调度器的以上两个缺点，Google 开发了下一代资源管理系统 Omega，Omega 是一种基于共享状态的调度器，该调度器将双层调度器中的集中式资源调度模块简化成了一些持久化的共享数据（状态）和针对这些数据的验证代码，而这里的“共享数据”实际上就是整个集群的实时资源使用信息。一旦引入共享数据后，共享数据的并发访问方式就成为该系统设计的核心，而 Omega 则采用了传统数据库中基于多版本的并发访问控制方式（也称为“乐观锁”, MVCC, Multi-Version Concurrency Control），这大大提升了 Omega 的并发性。由于 Omega 不再有集中式的调度模块，因此，不能像 Mesos 或者 YARN 那样，在一个统一模块中完成以下功能：对整个集群中的所有资源分组，限制每类应用程序的资源使用量，限制每个用户的资源使用量等，这些全部由各个应用程序调度器自我管理和控制，根据论文所述，Omega 只是将优先级这一限制放到了共享数据的验证代码中，即当同时由多个应用程序申请同一份资源时，优先级最高的那个应用程序将获得该资源，其他资源限制全部下放到各个子调度器。引入多版本并发控制后，限制该机制性能的一个因素是资源访问冲突的次数，冲突次数越多，系统性能下降的越快，而 google 通过实际负载测试证明，这种方式的冲突次数是完全可以接受的。Omega 论文中谈到，Omega 是从 Google 现有系统上演化而来的。既然这篇论文只介绍了 Omega 的调度器架构，我们可推测它的整体架构类似于 Mesos，这样，如果你了解 Mesos，那么可知道，我们可以通过仅修改 Mesos 的 Master 将之改造成一个 Omega。

2.4　资源调度算法

对于企业和公司，为了完成各种对外的服务以及公司内部业务逻辑，需要大量

2.4.1 FIFO 调度算法

对于企业和公司，为了完成各种对外的服务以及公司内部业务逻辑，需要大量

2.4.2 公平调度策略

对于企业和公司，为了完成各种对外的服务以及公司内部业务逻辑，需要大量

2.4.3 能力调度器

对于企业和公司，为了完成各种对外的服务以及公司内部业务逻辑，需要大量

2.4.4 延迟调度策略

对于企业和公司，为了完成各种对外的服务以及公司内部业务逻辑，需要大量

2.5 本章小结

对于企业和公司，为了完成各种对外的服务以及公司内部业务逻辑，需要大量

第三章 CBDRF 算法的相关技术

3.1　图的联通性算法

3.1.1 强联通分量

在有向图 G 中，如果两个顶点可以相互通达，则称两个顶点强连通 (strongly connected)。如果有向图 G 的每两个顶点都强连通，称 G 是一个强连通图。非强连通图有向图的极大强连通子图，称为强连通分量 (strongly connected components)。

下图中，子图 1，2，3 为一个强连通分量，因为顶点 1，2，3 两两可达。4，5，6 也分别是两个强连通分量。

直接根据定义，用双向遍历取交集的方法求强连通分量，时间复杂度为 O(N*N+M)。更好的方法有算法有，Tarjan 算法，Kosaraju 算法。其中 Tarjan 算法和 Kosaraju 算法均是是基于对图深度优先搜索的算法, 与 Kosaraju 算法不同，Tarjan 算法只进行一遍深度优先搜索索，较 Kosaraju 算法进行两遍深度优先搜索有 30% 的性能提升。

Tarjan 算法以一个有向图 G 作为输入，每个强连通分量为搜索树中的一棵子树首先定义结点 u 的深度优先搜索标号 DFN(u)，表示节点 u 是被访问的次序编号。此外，每个结点 u 还有一个值 Low(u)，表示从 u 出发经有向边可到达的所有结点中最小的次序号。显然 Low(u) 总是不大于 DFN(u)，且当从 v 出发经有向边不能到达其他结点时，这两个值相等，此时，以 u 为根的搜索子树上的所有节点属于同一个强连通分量。其中 Low(u) 在深度优先搜索的过程中求得，通过上述可以发现以 u 为根的搜索子树上的所有节点属于同一个强连通分量当且仅当 DFN(u)=Low(u) 时。而 Low(u) 的计算由以下公式给出:

$$Low(u) = \min \begin{cases} DFN(u) & \\ Low(v) & \text{(u,v) 为树枝边，u 为 v 的父节点} \\ DFN(v) & \text{(u,v) 为指向栈中节点的后向边 (非横叉边)} \end{cases}$$

由此可得，Tarjan 算法的基本流程如下:

1. 任选图 G 中的未被访问的结点 u 开始进行深度优先搜索遍历，如果深度优先搜索结束后仍有未访问的结点，则再从中任选一点再次进行。
2. 搜索过程中标记已访问的节点，如果是已访问的结点不再访问。

3. 搜索时，把当前搜索树中未处理的节点加入一个堆栈，回溯时可以判断栈顶到栈中的节点是否为一个强连通分量。

由此可以得到 Tarjan 算法在搜索时的主要的伪代码

Algorithm 1 Tarjan Algorithm

```
tarjan(u)
1  Index ← time + 1
2  DFN[u] ← time
3  Low[u] ← time
4  Stack.push(u)
5  for each(u, v) in E[u]
6      if visted(v)
7          tarjan(v)
8          Low[u] ← min(Low[u], Low[v])
9      elseif inStack(v)
10         Low[u] ← min(Low[u], DFN[v])
11 if DFN[u] == Low[u]
12     while u! = v
13         v ← Stack.pop()
14         print(v)
```

3.1.2 拓扑排序

在图论中，如果一个有向图无法从某个顶点出发经过若干条边回到该点，则这个图是一个有向无环图（DAG 图）。因为有向图中一个点经过两种路线到达另一个点未必形成环，因此有向无环图未必能转化成树，但任何有向树均为有向无环图。如右图，不为有向树，但为有向无环图。

对一个有向无环图 (Directed Acyclic Graph 简称 DAG)G 进行拓扑排序，是将 G 中所有顶点排成一个线性序列，使得图中任意一对顶点 u 和 v，若边 (u,v)□E(G)，则 u 在线性序列中出现在 v 之前。通常，这样的线性序列称为满足拓扑次序 (Topological Order) 的序列，简称拓扑序列。简单的说，由某个集合上的一个偏序得到该集合上的一个全序，这个操作称之为拓扑排序。

一个较大的工程往往被划分成许多子工程，我们把这些子工程称作活动

(activity)。在整个工程中，有些子工程 (活动) 必须在其它有关子工程完成之后才能开始，也就是说，一个子工程的开始是以它的所有前序子工程的结束为先决条件的，但有些子工程没有先决条件，可以安排在任何时间开始。为了形象地反映出整个工程中各个子工程 (活动) 之间的先后关系，可用一个有向图来表示，图中的顶点代表活动 (子工程)，图中的有向边代表活动的先后关系，即有向边的起点的活动是终点活动的前序活动，只有当起点活动完成之后，其终点活动才能进行。通常，我们把这种顶点表示活动、边表示活动间先后关系的有向图称做顶点活动网 (Activity On Vertex network)，简称 AOV 网。

在 AOV 网中，若不存在回路，则所有活动可排列成一个线性序列，使得每个活动的所有前驱活动都排在该活动的前面，我们把此序列叫做拓扑序列 (Topological order)，由 AOV 网构造拓扑序列的过程叫做拓扑排序 (Topological sort)。AOV 网的拓扑序列不是唯一的，满足上述定义的任一线性序列都称作它的拓扑序列。

由 AOV 网构造拓扑序列的拓扑排序算法主要是循环执行以下两步，直到不存在入度为 0 的顶点为止。

Algorithm 2 Topological Sort

TopologicalSort(G)

```
1  Queue.clear()
2  for each vertex u in G
3      if u.indeg == 0
4          Q.push(u)
5          isVisited[u] ← True
6  while Queue.size() > 0
7      u = Queue.front()
8      Queue.pop()
9      for each (u, v) in Edge[v]
10         v.indeg ← v.indeg − 1
11         if v.indeg ==0 and isVisited[v] == False
12             Q.push(v)
13             isVisited[v] ← True
```

1. 选择一个入度为 0 的顶点并输出之；

2. 从网中删除此顶点及所有出边。
3. 循环结束后，若输出的顶点数小于网中的顶点数，则输出“有回路”信息，否则输出的顶点序列就是一种拓扑序列。

3.1.3 并查集

在计算机科学中，并查集是一种树型的数据结构，其保持着用于处理一些不相交集合（Disjoint Sets）的合并及查询问题。有一个联合 -查找算法（union-find algorithm）定义了两个操作用于此数据结构：

- Find：确定元素属于哪一个子集。它可以被用来确定两个元素是否属于同一子集。
- Union：将两个子集合并成同一个集合。

因为它支持这两种操作，一个不相交集也常被称为联合 -查找数据结构（union-find data structure）或合并 -查找集合（merge-find set)。其他的重要方法，MakeSet，用于建立单元素集合。有了这些方法，许多经典的划分问题可以被解决。

为了更加精确的定义这些方法，需要定义如何表示集合。一种常用的策略是为每个集合选定一个固定的元素，称为代表，以表示整个集合。接着。Find(x) 返回 x 所属集合的代表，而 Union 使用两个集合的代表作为参数。

并查集实现方式有多种，包括并查集链表和并查集森林，其中并查集森林是并查集的高效实现。并查集森林是一种将每一个集合以树表示的数据结构，其中每一个节点保存着到它的父节点的引用。这个数据结构最早由 Bernard A. Galler 和 Michael J. Fischer 于 1964 年提出，但是经过了数年才完成了精确的分析。

在并查集森林中，每个集合的代表即是集合的根节点。“查找”根据其父节点的引用向根行进直到到底树根。“联合”将两棵树合并到一起，这通过将一棵树的根连接到另一棵树的根。实现这样操作的一种方法是：

这是并查集森林的最基础的表示方法，这个方法不会比链表法好，这是因为创建的树可能会严重不平衡；然而，可以用两种办法优化。

第一种优化方式是按秩合并，即总是将更小的树连接至更大的树上。因为影响运行时间的是树的深度，更小的树添加到更深的树的根上将不会增加秩除非它们的秩相同。在这个算法中，术语“秩”替代了“深度”，因为同时应用了路径压缩时（见下文）秩将不会与高度相同。单元素的树的秩定义为 0，当两棵秩同为 r 的树联合时，它们的秩 r+1。只使用这个方法将使最坏的运行时间提高至每个 MakeSet、Union 或 Find 操作 $O(\log n)$。采用按秩合并的 Union 伪代码如下所示：

Algorithm 3 Union-Set Union

```
Union(u, v)
1  rootu ← Find(u)
2  rootv ← Find(v)
3  if rootu ! = rootv
4      if rootu.rank < rootv.rank
5          rootu.parent ← rootv.parent
6          rootv.rank ← rootv.rank + rootu.rank
7      else
8          rootv.parent ← rootu.parent
9          rootu.rank ← rootv.rank + rootu.rank
10 return u.parent
```

另外一种优化的方式是采用路径压缩，这是一种在执行“查找”时扁平化树结构的方法。关键在于在路径上的每个节点都可以直接连接到根上；他们都有同样的表示方法。为了达到这样的效果，Find 递归地经过树，改变每一个节点的引用到根节点。得到的树将更加扁平，为以后直接或者间接引用节点的操作加速。采用路径压缩的 Find 伪代码如下所示：

Algorithm 4 Union-Set Find

```
Find(u)
1  if u.parent ! = root
2      u.parent ← Find(u.parent)
3  return u.parent
```

这两种技术可以互补，可以应用到另一个上，每个操作的平均时间仅为 O($\alpha(n)$)，$\alpha(n)$ 是 n = f(x) = A(x,x) 的反函数，并且 A 是急速增加的阿克曼函数。因为 α(n) 是其的反函数，α(n) 对于可观的巨大 n 还是小于 5。因此，平均运行时间是一个极小的常数。

3.2 公平性的度量

3.2.1 max-min 公平

我们经常面临给一组用户划分稀有资源的问题, 他们都享有等价的权利来获取资源, 但是其中一些用户实际上只需要比其他用户少的资源. 那么我们如何来分配资源呢? 一种在实际中广泛使用的分享技术称作“最大最小公平分享”. 直观上, 公平分享分配给每个用户想要的可以满足的最小需求, 然后将没有使用的资源均匀的分配给需要‘大资源’的用户。

最大最小公平分配算法的形式化定义如下：资源按照需求递增的顺序进行分配不存在用户得到的资源超过自己的需求未得到满足的用户等价的分享资源与之对应的可执行定义：

考虑用户集合 1, …, n 分别有资源需求 x1, x2, …, xn. 不失一般性, 令资源需求满足 x1 <= x2 <= …<= xn. 令服务器具有能力 C. 那么, 我们初始把 C/n 资源给需求最小的用户. 这可能会超过用户 1 的需求, 继续处理. 该过程结束时，每个用户得到的没有比自己要求更多，而且，如果其需求得不到满足，得到的资源也不会比其他用户得到的最多的资源还少. 我们之所以称之为最大最小公平分配是因为我们最大化了资源得不到满足的用户最小分配的资源.

示例 1

有一四个用户的集合, 资源需求分别是 2,2.6,4,5, 其资源总能力为 10, 为其计算最大最小公平分配

解决方法: 我们通过几轮的计算来计算最大最小公平分配. 第一轮, 我们暂时将资源划分成 4 个大小为 2.5 的. 由于这超过了用户 1 的需求, 这使得剩了 0.5 个均匀的分配给剩下的 3 个人资源, 给予他们每个 2.66. 这又超过了用户 2 的需求, 所以我们拥有额外的 0.066…来分配给剩下的两个用户, 给予每个用户 2.5+0.66…+0.033…=2.7. 因此公平分配是: 用户 1 得到 2, 用户 2 得到 2.6, 用户 3 和用户 4 每个都得到 2.7.

到目前为止, 我们假设所有的用户拥有相同的权利来获取资源. 有时候我们需要给予一些用户更大的配额. 特别的, 我们可能会给不同的用户 1, …, n 关联权重 w1, w2, …, wn, 这反映了他们间的资源配额.

我们通过定义带权的最大最小公平分配来扩展最大最小公平分配的概念以使其包含这样的权重:

资源按照需求递增的顺序进行分配, 通过权重来标准化? 不存在用户得到的资源超过自己的需求未得到满足的用户按照权重分享资源下面的示例描述了如何实

现?

示例 2

有一四个用户的集合, 资源需求分别是 4,2,10,4, 权重分别是 2.5,4,0.5,1, 资源总能力是 16, 为其计算最大最小公平分配.

解决方法: 第一步是标准化权重, 将最小的权重设置为 1. 这样权重集合更新为 5,8,1,2. 这样我们就假装需要的资源不是 4 份而是 5+8+1+2=16 份. 因此将资源划分成 16 份. 在资源分配的每一轮, 我们按照权重的比例来划分资源, 因此, 在第一轮, 我们计算 C/n 为 16/16=1. 在这一轮, 用户分别获得 5,8,1,2 单元的资源, 用户 1 得到了 5 个资源, 但是只需要 4, 所以多了 1 个资源, 同样的, 用户 2 多了 6 个资源. 用户 3 和用户 4 拖欠了, 因为他们的配额低于需求. 现在我们有 7 个单元的资源可以分配给用户 3 和用户 4. 他们的权重分别是 1 和 2, 最小的权重是 1, 因此不需要对权重进行标准化. 给予用户 3 额外的 7 × 1/3 单元资源和用户 4 额外的 7 × 2/3 单元. 这会导致用户 4 的配额达到了 2 + 7 × 2/3 = 6.666, 超过了需求. 所以我们将额外的 2.666 单元给用户 3, 最终获得 1 + 7/3 + 2.666 = 6 单元. 最终的分配是,4,2,6,4, 这就是带权的最大最小公平分配.

3.2.2 简氏公平

3.3　本章小结

第四章 CBDRF 调度系统的设计与实现

4.1 CBDRF 调度系统的提出

4.1.1 通信任务的关联性

4.1.2 系统的负载均衡

4.1.3 DRF 分配策略

4.2 CBDRF 调度系统的设计

4.2.1 任务调度解析

4.2.2 调度迭代控制

4.2.3 调度迭代评估

4.3 CBDRF 调度的实现

4.3.1 CBDRF 调度的流程

4.3.2 关联任务的解析

4.3.3 基于 DRF 算法的分配

4.3.4 关联任务的合并

第五章 实验与结果分析

5.1　实验环境

5.2　实验结果和分析

第六章 总结与展望

插图索引

表格索引

致 谢

在此论文完成之际，回首过往五年硕博连读学习时光，有欣慰也有艰辛，有成功也有失败，太多的感激、太多的谢意想要表达。在此，谨向所有在我生活中留下色彩的人表示由衷的感谢！

首先，我要衷心感谢我的指导老师...

...

感谢齐飞副教授提供 xdthesis 模版，它的存在让我的论文写作轻松自在了许多，让我的论文格式规整漂亮了许多。

...

攻读博士学位期间的研究成果

期刊论文

1. **Jinjian Wu**, Fei Qi, and Guangming Shi.“Self-Similarity Based Structural Regularity for Just Noticeable Difference Estimation,”Journal of Visual Communication and Image Representation, 23(6): 845-852, August, 2012.(SCI)
2. ...

会议论文

1. **Jinjian Wu**, Fei Qi, and Guangming Shi.“Image Quality Assessment Based on Improved Structural Similarity,”PCM 2012, Singapore.(EI)
2. ...

已授权专利

- 齐飞，**吴金建**，石光明，刘焱。基于人类视觉系统的图像感兴趣区域自动提取方法。授权专利号：ZL 2009 1 0022191.0
- ...

参加研究的科研项目

- **国家 863 计划项目**, 无线传感器网络协同获取融合与表达, 2008.1.-2009.12. (2007AA01Z307);
- ...