图钉链表：一种用于资源预留问题的改进型索引链表描述及其对应的性能分析

1. 概论

在高性能计算网格环境中，资源是分布在各个不同地域和管理域中，由不同的组织拥有和操作，并且在策略和安全机制上各有不同。站点的自治性、资源的动态性和异构性需要一种有效机制来联合分配位于多个站点上的资源。资源预留可以保证人为在开始执行时获得资源，而在网格这种资源动态性很强的环境中，任务能够如期执行可以大大提高整个网格系统的资源利用率和系统性能。同时由于资源预留还可以实现网格服务的QoS保证，因而资源预留是网格资源管理中一种普遍采用的机制，也成为网格资源管理的研究热点。

目前网格资源预留机制从理论到实际应用的关键在于如何提高自身的处理性能，以便用尽可能小的代价，为用户提供更高的QoS保证的网格服务。资源预留要求接纳控制的数据结构必须能够快速高效的查询资源的实验情况。实验表明，资源预留中有60%以上的时间都用于对数据结构的处理，而这仅仅是提供最简单的预留服务，如果请求全部潜在的高级预留服务（如在扫描探测资源时间间隔时），与数据结构相关的处理达到了总的处理时间的90%[1]。可见为了最小化处理时间，对数据结构的改进是最能影响整个处理时间的，对数据结构的优化可以在较大程度上提高资源提前预留服务的处理速度。

本文首先介绍了传统算法中各方面表现优异的索引链表，通过分析该算法的缺陷和局限性，提出了一种改进的数据结构——图钉链表。在此基础上，本文对于图钉链表的性能表现进行了定量的研究，并通过实验进行了验证，对它的性能进行了详细的测试。最后为了验证各种数据结构的性能差异，本文对传统的时隙数组、索引数组、图钉链表和较晚出现的B+树进行了实验对比，进行进一步的性能测试，从内存需求，请求接纳时间两方面对上述数据结构进行评估，并给出结论。

1. 相关研究
2. 传统索引链表

文献[x]对前文中提到的存储结构进行了详细的性能分析（不含RRB+树），得出在处理速度上，索引链表优于除时隙数组外的所有存储结构。但时隙数组的性能表现与其表达精度反相关，若预留任务对时间精度要求较高时，其性能会随之下降。反之，若需要追求较高性能，则无法保证完全的预留精度。

* 1. 算法描述

本文的算法由文献[x]中整体评价最优的用于资源预留机制的索引链表改进而来，因此本文首先将会介绍和分析该算法的主要过程。为了描述的简洁我们将文献[x]中的结构称为索引链表。该索引链表的存储结构如图[x]所示。



1. 索引链表存储结构图

图1中，第一行代表索引列，每一个方格表示一个索引的位置，图中所示的索引间隔为10个时间单位。第二列为实际存储资源预留情况的资源节点，每个资源节点包含当前节点对应的时间以及该时间点后系统已经预留的资源数目。比如第二行第二个节点(9,bw)表示，在时间9之后，系统已经对bw个单位的资源进行了分配。

索引链表与其他存储结构最大的区别在于其定位操作使用索引进行快速定位。传统的索引链表性能大幅优于树形检索结构的重要因素就是将索引引入了节点定位操作。该操作用于确定请求发生的起始与终止时间点在链表中的节点位置。索引定位速度很快，往往一次定位可以使节点范围缩小索引数目大小的倍数，而传统的树形结构则需要索引数组大小的对数时间来完成这次定位。

在索引链表中，索引过程的具体的操作主要分为两步，第一步为使用散列式确认需要定位的节点对应的索引序列，第二步则是确定需要定位的节点在链表中的具体位置。在第一步中，索引链表首先通过一个散列公式来获取需要定位的节点在索引中的位置，一般通过节点时间÷索引间隔来进行散列。在获得对应索引位置后，若当前索引指向一个有效节点，则从该节点开始进行遍历，定位节点的具体位置。以图1的结构为例，如果需要查询时间为14的节点，通过散列公式可以得出该节点位于索引A1的索引范围内，该索引指向节点（11，bw），然后遍历找到节点具体位置为（13，bw）的后续节点。若当前索引未指向一个有效节点，则需要通过反向查询操作取得最近的有效节点位置。比如，若要查找时间为22的节点位置，通过散列公式确定查询节点在A2索引的索引范围内，但该索引指向头结点，为非有效节点，故需要向前反向查询至索引A1，再由A1开始进行遍历操作，最终定位。

* 1. 存在的问题

在以往的所有模型中

以往的模型存在哪些问题

1、反查操作消耗时间

2、未对性能做定量分析，参数设置使用经验值

3、无法面对多样的预留环境

由前文的分析可得，由于预留任务中时间点的分布是随机的，因此存在有些索引节点中并没有指向对应的预留数据节点。在查询中，如果遇到此类索引节点，往往需要沿着索引数组进行反向查询。如图[x]中索引节点A2，若新加入的节点位于索引节点A2的索引范围，而索引A2当前指向为无意义的头结点HEAD。为了确定新节点的正确位置则需要沿着索引节点进行反向查找，寻找最近的有效索引节点。对于A2索引节点，需要进行1次反查操作，而对于图中A6节点则需要两次。

本文使用简单的随机填充来模拟这个请求发生的过程。通过1000次随机重复填充得出，对于100个大小的索引，完全填充平均需要522次被接受的插入请求，在完成填充前平均需要大约419次反查操作，而当索引数组大小扩大至200时，这两个平均数据变为975和1185。其具体的前200插入请求需要进行的平均反查次数分布如图[x]所示。

1. 索引链表平均填充次数

通过进一步的统计，对于使用100大小索引的索引链表，其前十次插入平均需要进行总计197次反查操作，前50次插入请求平均需要进行329次反查操作。当索引扩展至200大小时，这两个数据变为397和678。由此可见，大量的反向查询操作严重影响了该系统的响应速度，而随着索引数目的增大影响会越来越大，极大的限制了索引链表的拓展性，使得该结构不太适于长时间的资源预留问题。

为了降低反向查询操作的不利影响，同时为了提高索引数组的可扩展性能，本文针对该模型的不足，引入了一种图钉机制，对以上两种问题进行了改进，形成了本文接下来所描述的用于资源预留问题的改进型索引链表——图钉链表，并针对各种不同的环境，对其性能进行了定量的分析，找出了一个确定最佳索引数目的可行方式。

1. 图钉链表

本文在此索引链表的研究基础上针对这些可以改进的方面，提出了一种改进型的索引链表，即图钉链表。

* 1. 设计思路

改进思路：

使用图钉链表可以避免反查操作

性能分析，确定影响性能的参数

~~引入二级索引，提高对各种情况的适应性~~

图钉链表通过使用图钉节点来填充空白的索引节点，进而达到避免反查，提高初期请求响应速度的目的。其具体存储结构如图[x]所示。



1. 图钉链表存储结构图

图[x]中所示，第一行为索引项，第二行为存储的链表结构。图中实线节点表示请求被接受后插入的有效节点，虚线节点则为图钉节点。在预留开始之前，所有的索引节点都有唯一对应的图钉节点，随着不断接受预留请求，索引对应的图钉节点将被有效节点不断替换。由于已经提前插入了临时节点，因此可以保证每个索引节点都能指向存储链表的正确位置，进而避免反向查询操作的发生，达到每次定位都为有效定位的效果。

* 1. 具体算法

定位

一个接纳控制程序的主要目的在于提供一个合适的数据结构来存储管理区域中的预留资源信息。当请求到达时，需要检查参数，并决定是否有足够的资源用来预留和分配。每一个预留请求都由一系列参数来描述，在本文中，我们采用文献[5]中预留请求的定义方法：

其中bw表示预留带宽，ts,te分别表示开始时间和结束时间。

图钉算法中每次接纳操作的主要伪代码如图[x]所示。



1. 图钉链表插入算法

其中主要涉及到的两个关键函数为接纳控制函数accept函数与节点插入函数InsertNode函数。其中accept函数是接纳控制算法的核心算法，一般为了研究的方便，在存储结构的研究中一般使用固定请求接纳控制算法。本实验中accept函数与InsertNode函数伪代码如图[x]与图[x]所示。



1. 图钉链表接纳判断算法



1. 图钉链表节点插入算法
   1. 性能分析

由于图钉链表的算法使用索引机制对存储结构中的节点进行定位，因此其性能表现与索引的大小有着直接的关系。索引的数量越大，其一次定位的准确度越高。但与此同时，索引的维护开销也会有所提高。

在图钉链表算法中，部分操作属于固定固定开销，而有些操作会随着试验参数的变化而变化。对于每一次接纳操作，图钉链表需要进行的计算如下：

接纳判断操作：在接纳判断阶段，图钉链表算法需要对每个请求进行起点定位，其需要一次取余计算定位节点，以及若干次遍历操作定位至具体节点。在完成定位后接纳判断操作需要对持续时间内的每一个节点进行遍历判断，分析其是否有足够的剩余资源完成接纳请求。该阶段的操作时间可以使用公式[x]表示。

(1)

其中Costaccept表示接纳操作需要消耗的时间，C­accept表示接纳操作中一些固定次数操作需要的时间，比如函数中变量的新建，图钉定位等固定次数的计算，该部分为常数。Nloc则表示在定位后需要遍历查找至具体节点的遍历次数。Ndur表示接纳判断中，由起始节点到结束节点的遍历次数。Rm1、Rm2表示两种遍历的单次遍历的开销。

插入节点：由于插入节点不需要进行遍历操作，因此可以认为每次插入节点其开销为定值。因此，该阶段的操作时间可以用公式[x]表示。

(2)

其中Costinsert表示插入操作需要消耗的时间，它的值为Cinsert，是一个常数。

接纳修改操作：在接纳请求并插入了对应的起止节点后，图钉链表算法需要修改对应的资源存储节点，其操作时间与请求接纳判断类似，需要进行定位和对应的处理操作。其消耗时间可以使用公式[x]表示。

(3)

公式[x]中，Costmodify表示接纳修改需要的总体时间，Cmodify表示修改操作中的固定操作时间，为常数。Nloc、Ndur分别表示定位与修改操作的遍历次数。Rm1表示定位操作的单次遍历消耗，Wm表示节点修改遍历中的单次修改消耗。

以上操作位单次请求插入，需要消耗的各种计算时间。而图钉链表除了进行请求接纳操作外，还要定期的对已经执行完毕的预留请求进行资源回收操作，以维护整体接纳系统的性能表现。在图钉链表算法中，其资源释放间隔为每个图钉长度，当旧图钉失效时，会释放图钉中所有已有的节点，并将图钉修改为新的可用图钉，用于之后的请求接纳操作。对于一个图钉的释放过程，其时间消耗可以使用公式[x]表示。

(4)

其中Costrelease表示图钉释放需要的总体时间消耗，Crelease表示修改图钉需要的时间，Nnode表示该图钉中需要释放的节点数目，Dnode表示释放单个节点需要的时间。

综合以上分析，我们可以得出，对于单个资源预留请求，图钉算法的时间消耗如公式[x]所示。

(5)

其中C为所有常数消耗的总和。Ntack为该节点所在图钉节点中所有节点的数目。

在所有请求平均分布的情况下，公式[x]中Nloc平均值为单个图钉的时间长度除以2倍的每个节点的时间间隔，Ndur等于单个请求的持续时间除以每个节点的时间间隔，Ntack平均值等于单个图钉的时间长度除以每个节点的时间间隔。因此，设单个图钉时间长度为T，每个请求之间的平均间隔为t，每个请求的平均持续时间为d，则公式[x]可以转换为公式[x]。

(6)

进一步的，公式[x]可以简化为公式[x]。

(7)

其中x、y、z分别表示定位消耗、节点读写消耗、节点释放消耗的时间消耗系数。且当T满足T=t√(z/x)时，单次请求的平均消耗取得最小值Costmin=C+2√(xz)。

1. 试验分析

前面的内容，对算法的过程和具体的性能表现进行的详细的分析，接下来本文将通过实验对前面的分析进行进一步的验证，并结合不同预留环境与其他模型进行进一步的对比分析。

* 1. 实验环境

本文所有试验均以C++语言环境为基础，使用eclipse+MinGW的编译环境进行代码编写。硬件环境为：Intel(R) Pentium(R) CPU G2020 @ 2.90GHz 双核CPU，4GB内存，操作系统为Windows 7 Service Pack 1 64位。每个试验具体的试验参数，会在试验中详细描述。

* 1. 性能验证

1、研究较短持续时间索引数与性能关系，求x

2、研究持续时间与间隔的性能关系，求y

首先，本文将通过实验验证4.3节对图钉链表的性能分析结果。由公式[x]分析可以得出，处理单个请求的平均响应时间与三个参数有关，即每个图钉间的间隔时间、每个请求间的间隔时间以及请求的平均持续时间。因此本文在性能验证方面，将分别研究：短持续时间时图钉间隔对性能的影响以及持续时间对于性能的影响。在此基础上求解本实验环境下的三个性能参数，即公式[x]中的x、y、z。

为了满足公式[x]的前置条件，即试验中各时间节点分布较为均匀，因此性能验证部分的试验请求参数均为取值范围内的平均分布。

在实验短持续时间时图钉间隔对系统请求响应时间的影响中，系统首先生成了10万个随机的预留请求，每个请求均满足持续时间为1个时间单位，开始时间为1至最大预留允许时间-1个时间单位。由于公式[x]中，预留结构的性能与最大预留允许时间无直接关联，而由于内存管理方式的问题，程序无法直接设置每个图钉的间隔情况，只能通过设置图钉个数等于最大预留允许时间除以图钉间隔来间接控制，因此必须要有足够大的预留允许时间来保证预留间隔，所以试验中设置最大预留允许时间为131072(2的17次方)个时间单位，同时为了保证每个请求的正常接收，所有请求的资源请求量均为1。

1. 短持续时间时图钉间隔对性能的影响

图[x]为请求短持续时间时图钉间隔对系统请求响应时间的影响的实验结果。图[x]中，纵坐标为响应全部10万次处理请求所花费的总体时间，单位为ms；横坐标为每个图钉之间的时间间隔，单位为1个时间单位。其中不同的示例参数t，表示每个预留请求间的间隔，单位为1个时间单位。

由公式[x]中可以得出，当T远大于t时，其参数z的影响变得极小。因此对图7中的数据进行截取，并求得其趋势函数，可得图[x]。

1. 短持续时间时图钉间隔对性能的影响的截取数据

将对应的t参数带入求解，可以得出对于t等于4、8、16、32四种情况，其x取值分别为0.4212、0.4216、0.3456、0.272。对于t为4和8时，可以认为此时z对其性能的影响已经足够小，因此x的值约为0.421\*10-4ms。

第二个实验，本文将研究持续时间对于不同设置下的图钉链表的性能影响。与前一个实验类似，本次试验系统会首先生成10万个随机分布的资源预留请求。其每个请求均满足开始时间为1至最大预留允许时间-持续时间个时间单位。最大预留允许时间依然为131072。每个请求的间隔时间为4个时间单位。则第二次试验的实验结果如图[x]所示。

1. 持续时间对于图钉链表的性能影响

图[x]中，纵坐标为响应全部10万次处理请求所花费的总体时间，单位为ms；横坐标为每个请求的持续时间，单位为1个时间单位。不同的示例参数T，表示每个图钉间的间隔，单位为1个时间单位。与前一个实验类似，这里求得的趋势函数分别为y = 0.408x + 115.76、y = 0.4067x + 55.637、y = 0.4084x + 24.28、y = 0.4058x + 10.469。将参数t=4代入公式[x]，可以求得公式[x]中的参数y=1.629\*10-4ms。

由于试验2中除参数d外，所有参数均与试验仪中t=4时情况相同，故将x、y参数代入公式[x]，并结合图[x]中t=4时的趋势函数，可以求出，公式[x]中C=23.329\*10-4ms。结合图[x]中，前半部分的数据，最终可以求出z=1.797\*10-4ms。而试验中的各种情况，也大致符合公式[x]的描述，因此可以认为公式[x]能够较为正确的反应本算法的性能，且本算法在T=t√(z/x)，即T=2.07t时取得最优性能。

* 1. 性能对比

1、与时索引链表间的初期性能对比关系

2、与时索引链表间的稳定性能对比关系

3、最大吞吐量研究

4、模拟实际环境的试验结果

本文所提出的图钉算法由传统的索引算法改进而来，因此在性能对比方面，首先需要与传统的索引链表性能进行比较。由于本文改进的主要出发点是提高初期请求的响应速度，因此与索引链表的性能对比又进一步分为初期性能对比与稳定性能对比。

在研究图钉链表与索引链表初期性能表现时，本文首先随机生成了20000个资源预留请求，每个请求的持续时间为4个时间单位，其开始的时间为当前时间到最大预留允许时间-4之间，每个请求间的间隔为4个时间单位。由4.3节的分析可以得出，最大预留时间与性能并无直接关联，因此为了体现每一次变化的精度，这里将最大预留允许时间设置为1048576。同时，为了使测量尽可能精确，本次试验将20000个请求分为20个一组，按顺序提交给1000个已经完成初始化操作的资源预留算法，计算器话费的总时间。最后结果如图[x]所示。

1. 图钉链表与索引链表的初期性能对比

图[x]中，横坐标为索引的数目，纵坐标为1000个资源预留结构完成所有20个请求的分组后消耗的总时间，单位为ms。由图中何以明显看出，图钉链表算法在索引变化时能够维持较低的处理时间，而索引链表则由于需要进行大量的反向查询操作，而随着索引数目的增加，响应时间逐步增长。

在研究图钉链表与索引链表稳定性能表现时，其他参数均与初期表现研究相同。但在稳定性能研究时，本文会生成10万次请求操作，并且针对每种索引设置，会首先使用这10万次请求中的若干次对预留接纳数据结构进行一次最大预留允许长度的填充。待完成第一次填充后，再对这10万次请求进行响应，并记录所需时间。最后的实验结果如图[x]所示。

1. 图钉链表与索引链表稳定性能表现

由图[x]可以发现，在稳定性能方面，由于图钉链表与索引链表采用类似的定位方案，其两者唯一的操作区别就存在于初期反向查询定位上，因此当预留请求进入稳定期，这两种预留结构在性能与参数的相关性上具有高度的一致性。

在完成与索引列表的初期性能和稳定性能对比之后，接下来本文将使用压力测试方案，对索引链表和图钉链表的性能进行测试。为了增强对比效果，本文额外引入了使用好多的时隙数组方案，以及最近由文献[x]提出的较新的用于资源预留的B+书方案进行对比。

在压力性能测试中，本文将会随机生成100万个请求，每个请求满足最大预留允许时间长度为432000秒（约合5天）。请求的持续时间为30秒至1800秒之间。为了保证每次请求都被接受，我们设定没个请求的资源申请数量为1，且资源预留系统的资源拥有量为UINT\_MAX（4294967295）。测试过程中，我们使用一个线程进行满负荷请求接纳操作，另一个线程每个200ms进行一次响应操作次数统计。得出前2000次记录如图[x]和图[x]所示。

1. 压力测试响应请求积累数
2. 压力测试响应请求增量图

图[x]和图[x]所示分别为要测试中响应的请求总数与每次检测时产生的增量。由图中可以看出，时隙数组其性能表现十分均匀，因为其性能仅于任务持续时间相关。而B+树、索引链表和图钉链表其性能表现随着存储结构的逐渐填充，而慢慢下降，直至稳定。

1. 总结
2. 参考文献

[1] 吴黎兵，于天水，何炎祥，李飞，“一种面向网格资源预留的索引链表研究”，武汉理工大学学报，第33卷6期p904-908，2011