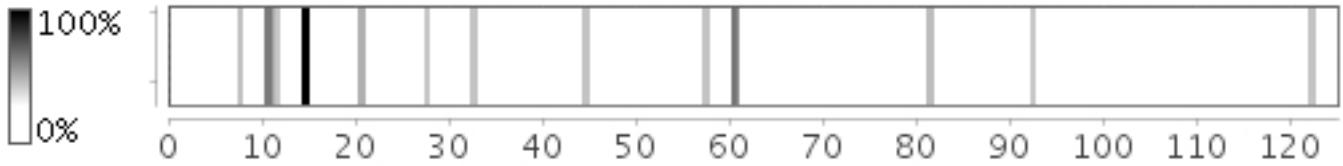
PaperPass专业版检测报告
简明打印版

比对结果（相似度）：
总　体：5 %　*（总体相似度是指本地库、互联网的综合比对结果）*
本地库：4 %　*（本地库相似度是指论文与学术期刊、学位论文、会议论文数据库的比对结果）*
期刊库：1 %　*（期刊库相似度是指论文与学术期刊库的比对结果）*
学位库：3 %　*（学位库相似度是指论文与学位论文库的比对结果）*
会议库： 1 %　*（会议库相似度是指论文与会议论文库的比对结果）*
互联网：1 %　*（互联网相似度是指论文与互联网资源的比对结果）*

编　号：58B695769BC76N4ER
版　本：专业版
标　题：基于Node-red与Redis的实时流数据处理模型的设计与
作　者：王江波
长　度：5331 字符(不计空格)
句子数：125句
时　间：2017-3-1 17:33:42
比对库：学术期刊、学位论文（硕博库）、会议论文、互联网资源
查真伪：http://www.paperpass.com/check

句子相似度分布图：



本地库相似资源列表（学术期刊、学位论文、会议论文）：
1. 相似度：1 % 篇名:《非关系型数据库数据恢复技术研究》
来源：学位论文 杭州电子科技大学 2014 作者: 徐小威

互联网相似资源列表：
1. 相似度：1 % 标题：《数据结构课设之跳跃链表_百度文库》
http://wenku.baidu.com/view/2ff29c869e314332396893f1.html

全文简明报告：

第四章 基于Redis有序集合的去重统计方法的研究

上一章详细阐述了基于 Node- red与 Redis的流式数据处理模型的设计， 在整个流式数据处理模型中， Redis作为数据交换和数据计算的中心， Redis的有序集合 zset被用来进行统计计算， 最重要的工作就是去重统计。 本章将从“ 跳表 ”Skip List的基本原理到zset的源码分析，详细阐述有序集合的去重统计的原理。

4.1 Skip List基本原理

Skip List是由William Pugh提出的一种基于并联链表的、随机化的数据结构。 {42％： Skip List的查找效率与二叉查找树的查找效率差不多，可以实现平均复杂度为 的插入、删除和查找操作。} 一般而言，跳表是在有序的链表的基础上增加的额外的链接，这种链接方式的增加， 不是随便增加的而是以随机化的方式增加的，同时对随机函数也有要求，必须是以对数函数的方式产生， {64％：所以在链表中的查找可以迅速地跳过部分链表，跳表也因此得名。} {41％：众所周知，对于有序链表的查找操作，其时间复杂度为，尽管真正插入与删除节点的操作的复杂度只有，} 但是，这些操作都需要首先查找到节点的位置，换句话说，是查找拉低了有序链表的整体性能。 而Skip List采用“ 空间换时间 ”的设计思想，除了原始链表外还保存一些“ 跳跃 ”的链表，达到加速查找的效果。 {100％：可以很好解决有序链表查找特定值的困难。}

接下来研究一下Skip List实现的原理，首先来认识一下Skip List。 因为，“ 跳表 ”是在有序链表的基础上做改进的，所以我们从认识有序链表开始研究Skip List。 图4-1 展示的就是一个有序链表的数据结构图（这里H表示链表头部，T表示链表尾部，不是有效节点）：

图4-1 有序链表结构图

现在假设要在该有序链表中查找value为7的节点，只能一步一步地从头到尾按照1-]2-]3...的顺序找下去，很明显查找效率是。 {47％：如果是数组的话，可以利用二分查找，时间复杂度可以提高到。} 但是链表不支持随机访问，所以不能应用二分查找。 但是可以考虑把中间位置的节点保存下来，重新构成新的顺序链表，经过重构的链表如图4-2 所示：

图4-2 重构后的有序链表

毫无疑问，这是一种典型的以空间换时间的设计思想。 原始的顺序链表，经过重构后变成了三个顺序链表，从下到上将这三个链表编号为0、1、2， 不难发现，2号链表就是原始链表，1号链表就是原始链表的四等分节点构成的， {41％：0号链表是原始链表的二等分节点构成的。} 现在，再来查找value为7的节点则只需要如下三个步骤：

（1）初始的的搜索范围是(H，T)，在0号链表中与4进行比较，7]4，将搜索范围更新为(4，T)。

（2）在1号链表中与6进行比较，7]6，继续更新搜索范围(6，T)。

（3）在2号链表中与7进行比较，结果7=7，查找成功。

{41％：很明显，在Skip List中保存了二分查找的信息，以此来提高查找效率。} 当然在具体的实现上，如果要开辟额外的空间来保存新链表的话，会造成空间的极大浪费。 由于是链表，可以利用链的性质，改进存储结构，改进后的Skip List的存储结构图4-3 所示。

图4-3 改进后的Skip List存储结构

前面所讨论的Skip List结构是一种比较理想的结构，实际的Skip List算法是一种随机算法，它非常依赖于所生产的随机函数。 当然对随机函数的要求也比较严格，不能简单的按照 的形式来生成随机数，而是必须要按照满足概率 的几何分布来构造随机函数。 可以设计出如下随机函数randLevel():

现在考虑 的情况，可能的返回值有0、1、2、3四种情况，他们各自出现的概率是： 、、、。 也就是说，如果有16个元素的话，第零层预计有16个元素，第一层预计有8个元素， 第二层约有4个元素，第三层约有2个元素，从下向上每层元素数量大约会减少一半。 因此，Skip List适合自顶向下进行查找，理想情况下，每下降一层搜索的范围就会缩小一半，可以达到二分查找的效率，时间复杂度为。 {41%：最坏的情况是当前节点从head移动到链表的尾部，时间复杂度为。}

4.2 redis有序集合zset的源码分析

Redis的有序集合zset的底层数据结构就是通过Skip List来实现的，而没有采用hash和hashtable来实现，虽然hash可以实现快速的查找，但是无法保证有序。 在了解了Skip List的基本原理后，接下来通过分析redis的源码，详细阐述zset的实现。 Redis中的zset所使用的Skip List与William Pugh提出的基本一致，只是做了部分改进，主要有三个方面的改进。

（1）Redis中的Skip List可以有重复的分值score，这是为了支持有序集合中可能有多个元素具有相同的分值score这样的需求。

（2）在节点进行比较的时候，不仅仅比较他们的score，同时还要比较他们所关联的元素的value。

（3）在Skip List中每个节点还有一个前向指针，这就相当于在双向链表中的prev指针，通过这个指针，可以从表尾向表头进行遍历。 正因为有了这个改进，zset就支持一些逆向操作命令，比如zrevrange、zremrangebyscore等。

redis的源码中，zset的Skip List的节点定义在redis.h头文件中，其具体定义如下：

有了节点的定义，那么就该是Skip List的定义了，Skip List同样也是定义在redis.h头文件中的。 和定义链表的结构一样，需要头节点、尾节点，他们都是指向zskiplistNode 的指针，同时还需要定义节点的数量，目前跳表的最大层数。 下面就是zset的跳表定义：

{41%：其实，Redis的有序集合zset主要支持的编码方式有两种，一种是ZIPLIST方式，另一种是SKIPLIST方式。} 其中ZIPLIST方式可以表示较小的有序集合，而SKIPLIST方式可以表示任意大小的有序集合。 如果zset当前使用的编码方式是ZIPLIST，只要满足下面两个条件之一就可以转换为SKIPLIST编码方式。

{65%：（1）当新增加的字符串的长度超过了server.zset_max_ziplist_value的时候（默认值为64）。}

（2）当ziplist中保存的节点数超过了server.zset_max_ziplist_entries的时候（默认值为128）。

在zset的源码中这两种方式的转换可以通过zsetConvert函数来完成。 SKIPLIST编码方式的zset集合的结构是定

义在redis.h中的，其定义如下：

有了数据结构的定义，接下来就是考虑对这些数据结构的操作了。 在redis的实现中，将对zkiplist的操作都放在t_zset.c源文件中，所支持的操作有三十多种之多。 包括创建层数为某一level的跳表节点、创建一个跳表、释放跳表、向跳表中插入一个节点、删除一个节点等基本操作。 下面来看一下zset是创建一个空的跳表后是如何向跳表中插入节点的。 首先，调用zslCreate()函数创建并初始化一个新的Skip List，一个空的Skip List如图4-4 所示。

图4-4 空跳表结构图

在该跳表的结构图中，level 0到level 31是一个长度为32的zskiplistLevel结构体数组，其大小由宏ZSKIPLIST_MAXLEVEL定义，值为32。 在zskiplistLevel结构体中还包括了span和forward两个数据成员，这一点从该结构体的定义中可以看出，这里为了展示方便，忽略了span。

创建完跳表之后，调用zslInsert()函数，就该向空跳表中插入节点。 插入一个新的节点的大致过程如下：

（1）按照跳表的结构按层数从上向下遍历。

（2）在当前level的当前节点向右遍历，如果发现分值score相同就比较value的值，否则进入下一步。

（3）调用随机函数，产生随机的层数。

（4）比较当前level与随机函数产生的随机level，记录最大的level，作为下一步遍历的level。

（5）插入节点，并更新跨度span

在第三步中调用随机函数，生成随机的层数，这一点在上一小节关于Skip List的实现原理中已经做了阐述。关于如何查找插入位置，在zset的源码中是这样实现的：

{44%：下面举例说明跳表节点的插入操作，假设要向跳表中插入A、B、C、D四个节点，} 它们对应的分值为3、5、7、9，则对应的跳表结构如图4-5所示：

图4-5 跳表节点插入步骤图

从图中可以看出，跳表中的节点都是按照分值score来进行排序的。 同时，每个节点的backward指针都指向它的前一个节点，因此，跳表和双向链表类似，支持许多逆向查找，提高了灵活性和操作的效率。

4.3 基于zset的去重统计方法

去重统计，在数据分析领域是一个耳熟能详的词语，可以说去重统计在大部分数据处理过程中都要用到。 众所周知，在大部分的数据分析的中间计算过程中，最终的数据指标主要呈现以下几种形式： 最大、最小、稳定性、叠加、去重统计。 在这五种数据指标中，前四种在大部分的实时处理框架和nosql中都可以使用相对较小的开销就可以完成计算。 而对于去重统计，由于去重的数据有可能是多维的，所以不论是io效率上，还是计算的效率上都没有前四种标高。

{41%：本文所设计的实时流数据处理模型中，也对这五种数据指标的计算做了设计。} 经过前两节对Skip List的基本原理和redis有序集合的源码分析研究，本文认为利用redis的zset来做数据去重统计是可行的。 在许多流式数据处理的应用中都会涉及到最大值、最小值、累计求和等数据指标的计算，而要计算这些数据指标的基础就是去重统计， 因此，涉及一种高效的去重统计方法显得意义也十分重大。

本文所提出的基于 zset的去重统计方法，就是在流式处理模型中引入 redis数据库的访问节点（第三章所设计的 redis_ in和 redis_ out节点）， 通过这些节点在流式计算的过程中，将产生的中间结果集存储到 redis的有序集合 zset中，并根据上游节点提供的命令格式， 对指定的集合进行 zincrby操作。 在redis所提供的客户端进行 zincrby操作的命令格式是这样的： zincrby zsetkey increment member，如果在名称为 zsetkey的 zset中已经存在元素 member，那么该元素的 score增加 increment，否则向该集合中添加该元素， 其 score的值为 increment，若增加成功返回的是 member增长之后的序列号。 也就是说，在Node-red中进行去重统计的过程就是通过redis_in节点对相应结合进行zincrby操作的过程。

本文在设计redis_in节点的时候规定了上游节点传输过来的数据格式，因为redis_in节点操作数据库的命令就是从上游节点传输过来的数据中获取的。 就以实际项目中一个功能来说明这一点，关于某一网站错误页面的统计。 对于这个功能，前端页面要求展示错误页面的URL、错误类型、错误页面所属的网站的域名、该错误页面是从哪个页面跳转来的等信息。 很显然错误页面具有着四个维度，如果我们单独去统计每一个维度的信息，最后再来进行整合，这样会大大减低计算的效率。 为此，我们要将多维统计转换为一维统计，同时也不能影响展示界面要求的四维信息。 本文所采取的降低维度的做法是将这四个维度拼接在一起，每个维度之间用特殊字符间隔， 这样就形成了一个维度的指标，让后将这个指标作为 zset的 key值，当 zset在进行 zincrby操作的时候， 就会根据这个 key来进行插入操作。 图4-6 所展示的就是在Node-red中redis_in节点所要求的数据格式：

图4-6 redis_in要求的数据格式

在图4-6所编写的函数中，就用到了 Node- red的 function node，该节点将数据封装在 msg对象的 payload字段中， 同时返回 msg对象，在该节点的内部调用了 node. send()方法，将 msg对象发送给下一个节点， 供下一个节点接收处理。 在图中整个msg.payload=[’ zincrby ’ ， ’ errPageDisplay ’ ， 1，err]，就是操作redis有序集合的 zincrby命令，其中errPageDisplay是有序集合的名字，err是通过降低维度后的一维指标。

上面这个例子展示了在实际项目中利用本文所设计的redis_in节点进行去重统计的过程。 之所以说去重统计是一项基础计算，是因为，在进行去重统计的同时，只需要一些简单的操作就可以去计算最大值、最小值、累计求和等计算指标。 不如要想知道 zset中的最大值或最小值，只需要返回集合中的第一个元素或者最后一个元素，有时候需要返回排名前 N的记录， 也就是常用的 Top n操作，在去重统计的基础上也很容易实现。

4.4 本章小结

{41%：由于，实时流数据处理中会经常遇到去重统计，而本文所设计的实时流式数据处理模型中引入了redis作为数据计算中心，} 基于 redis有序集合的去重统计方法也被应用到该模型中。 本章主要是对redis的有序集合底层实现原理进行分析研究，同时研究分析了有序集合的底层实现源码，最后也阐述了基于zset的去重统计方法在该模型中的具体应用。