第46卷第1期 中国科学技术大学学报 Vol. 46, No. 1
2016年1月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Jan. 2016

文章编号:0253-2778(2016)1-0036-11

# MCDS:大规模移动通信数据计算的单机实现

刘志鹏

(南京邮电大学计算机学院,江苏南京 210003)

**摘要**:移动数据具有数量庞大、类型多样、时效性强和高价值等特点.移动通信数据是一种重要的移动数据,对高效地存储和访问移动通信数据进行研究,并在此基础上更加有效地开展移动数据挖掘的相关研究,具有重大现实意义.当前,使用并行数据挖掘技术进行数据挖掘得到普遍认可,但并行数据挖掘技术需要较高的硬件成本,并行算法代码调试和优化较为困难.为此提出大规模移动通信数据的单机实现MCDS(mobile communication data processing system).MCDS基于GraphChi,改进了数据格式、分片机制、数据分片换入换出机制.实验结果验证了MCDS的有效性,为移动数据挖掘提供了切实可行的实验环境.

**关键词**:移动数据;移动数据挖掘;移动通信数据;MCDS

**中图分类号**:TP391 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2016.01.006

**引用格式**:LIU Zhipeng. MCDS: Large-scale mobile communication data computation on just a PC[J]. Journal of University of Science and Technology of China, 2016,46(1):36-46.
刘志鹏. MCDS:大规模移动通信数据计算的单机实现[J]. 中国科学技术大学学报,2016,46(1):36-46.

# MCDS: Large-scale mobile communication data computation on just a PC

LIU Zhipeng

(*School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing* 210003, *China*)

**Abstract**: Mobile data has the characteristics of high volume, variety, velocity and value. Mobile communication data is an important part of mobile data, and it has great research value. It is of tremendous significance to efficiently store and retrieve mobile data. At present, utilizing parallel technology to perform data mining has become the main stream, but the technology is very costly in terms of hardware, and code debugging and optimization of parallel algorithms is difficult. A mobile communication data processing system operational on a single PC was proposed. MCDS is based on GraphChi, and improves GraphChi from 3 aspects: data format, sharding mechanism and memory replacement algorithm. Experimental results verify the effectiveness of MCDS, and it provides a feasible experimental environment for mobile communication data mining.

**Key words**: mobile data; mobile data mining; mobile communication data; MCDS

**收稿日期**:2015-08-27;**修回日期**:2015-09-29

**基金项目**:国家自然科学基金(61473001,71071045,71131002);安徽大学青年科学研究基金(33050054)资助.

**作者简介**:刘志鹏,男 1980 年生,博士/讲师.研究方向:数据挖掘.E-mail:liuzhipengcs@139.com

## 0　引言

移动互联网是当前信息技术的热门领域之一. 中国工业和信息化部电信研究院在 2011 年的《移动互联网白皮书》中给出了移动互联网的定义:“移动互联网是以移动网络作为接入网络的互联网及服务,包括 3 个要素:移动终端、移动网络和应用服务”.

移动互联网的发展产生了移动数据. 移动数据的范围有广义和狭义之分:①从广义上说,移动数据包括移动终端、移动网络和应用服务产生的所有数据. ②从狭义上说,移动数据仅包括移动终端产生的各种数据. 本文研究狭义上的移动数据,其处理的数据对象主要包括通话、短信数据,位置数据和移动 Web 应用数据等. 与传统数据相比,移动数据具有数量庞大、类型多样、时效性强和高价值等特点.

(Ⅰ) 数量大. 这体现在两个方面:①移动终端数量与日俱增,它是移动数据的主要产生来源. ②通信数据量不断增大. 根据 2014 年工信部数据的统计,2014 年 1 月份,全国移动电话去话通话时长 2465.8 亿分钟,全国点对点短信量 436.7 亿条.

(Ⅱ)类型多样. 移动数据来自于移动终端软硬件、移动网络和应用服务等 4 个主要方面. 以移动网络数据为例,包括:通话记录、SMS(接收/发送/失败/挂起)、联系人列表、通信基站信息、GPS 数据、蓝牙数据、WLAN 数据等.

(Ⅲ)时效性强. 移动数据是高度动态变化的. 这体现在:①移动互联网络本身动态性较强. 移动终端经常由用户随身携带,位置频繁移动. ②很多移动应用,如手机打车软件和公交位置查询软件,其数据更新频繁、时效性强.

(Ⅳ)高价值. 移动数据往往与特定的移动应用相关,而移动应用本身便具有很高的经济价值和社会价值.

移动数据由以下几种数据组成:

①移动通信数据. 该类数据主要由通话和短信记录组成. 这类数据是传统移动数据的主要组成部分.

②移动位置数据. 该类数据的来源主要为智能手机上的 GPS、WLAN 和蓝牙模块. 其中最常用的是 GPS 位置信息.

③其他数据. 该类数据种类繁多,包括智能手机上存储的照片和视频等.

当前,移动数据主要由移动通信数据组成. 高效地存储和访问移动通信数据,并在此基础上更加有效地开展移动数据挖掘的相关研究,具有重大现实意义. 当前,并行数据挖掘技术已成为数据挖掘的主流,主要采用硬件集群、使用分布式计算方法进行数据存储和运算. 其中,典型的分布式解决方案包括 Gbase[1-2]、Pregel[3]、GraphLab[4] 和 PowerGraph[5] 等. 然而,并行数据挖掘技术需要较高的硬件成本,并行算法代码调试和优化较为困难. 在数据未达到 PB 字节规模时,采用单机计算是切实可行的. 该方法使用一台 PC 机器、使用单机算法开展数据运算. 与并行数据挖掘技术相比,单机计算具有成本低、调试简单等优势. 典型的实现包括 GraphChi[6] 和 TurboGraph[7] 等.

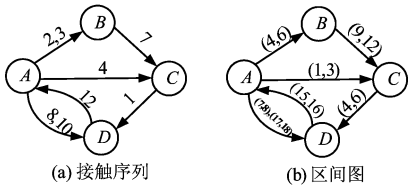
GraphChi 是一种单机图运算工具. 它是一种异步的顶点中心编程模型. GraphChi 将图中的顶点划分为 $P$ 个执行区间,其原理使用分片(Sharding)技术. 每个执行区间包含一个分片文件,其中存储了所有以该区间中顶点为目的节点的边,这些边根据其起始节点进行排序. 同时,GraphChi 使用并行滑动窗口(parallel sliding windows, PSW)技术,一次处理一个分片文件. 在处理分片文件时,有 3 个子任务:①从磁盘中加载子图;②更新相应的顶点和边;③将更新部分的图写回磁盘中. GraphChi 是一个基于 Linux 平台的开源项目,其主页提供源代码下载①. TurboGraph 与 GraphChi 相比,虽然性能更加优秀,但由于其并非开源软件,需要线程编程经验和专门的 FlashSSD 硬件支持等条件,难以在 TurboGraph 上开展进一步研究工作.

本文提出大规模移动通信数据的单机实现 MCDS. MCDS 基于 GraphChi,并改进了数据格式、分片机制、数据分片换入换出机制等 3 个方面. 实验结果验证了 MCDS 的有效性,为移动数据挖掘提供了切实可行的实验环境.

## 1　移动通信数据表示方法

移动通信数据通常使用时序网络表示. 时序网络一般用来描述节点之间的时序特征和拓扑相关性[8]. 其中主要有两种类型的时序网络:接触序列(contact sequence)和区间图(interval graph),如图 1 所示.

① http://graphlab.org/projects/graphchi.html



(a) 接触序列 (b) 区间图

**图1 两种类型的时序网络**

**Fig. 1 Two types of temporal networks**

给定一个时序网络图 $G=(V,E,T_e)$，其中，$V$ 代表节点集合，其节点个数为 $|V|$ 个，$E$ 代表有向边集合. 对图1(a)、(b)而言，$V=\{A,B,C,D\}$，$|V|=4$，$E=\{<A,B>,<B,C>,<A,C>,<C,D>,<A,D>,<D,A>\}$.

接触序列表示在某个特定的时间点 $|V|$ 个顶点的交互活动，该交互活动持续的时间可忽略不计. 该交互过程由边集合 $E$ 表示. 接触序列由三元组 $<v_i,v_j,t>$ 组成，其中 $v_i$、$v_j$ 为节点，$t$ 表示对应事件发生的时间. 为了方便起见，也可将该三元组使用二元组 $<v_i',v_j'>$ 表示. 对接触序列而言，$T_e$ 是接触序列中交互活动发生的时间点的集合，即 $T_e=\{t_1,\ldots,t_n\}$. 在图1(a)中，$T_e=\{1,2,3,4,7,8,10,12\}$. 接触序列可以表示 Email、短信、微博事件等与时间有关的活动.

与接触序列不同，区间图中的节点交互活动持续一段时间，该时间段使用区间 $(t_i,t_{i+1})$ 表示，$i\in[0,n-1]$. 其中，$t_i$ 为区间活动的起点，$t_{i+1}$ 为区间活动的终点. $T_e=\{(t_0,t_1),\ldots,(t_{n-1},t_n)\}$. 在图1(b)中，$T_e=\{(1,3),(4,6),(7,8),(9,12),(13,14),(15,16),(17,18)\}$.

相邻系数 $a(v_i,v_j,t)$ 表示时序网络中 $t$ 时刻 $v_i$ 和 $v_j$ 是否相邻. 计算公式如下：

$$a(v_i,v_j,t)=\begin{cases}1,\text{如果 } v_i,v_j \text{ 在 } t \text{ 时刻相连}\\0,\text{否则}\end{cases} \tag{1}$$

图1(a)中，$a(A,D,9)=0$，$a(C,D,1)=1$. 图1(b)中，$a(A,C,4)=0$，$a(A,B,5)=1$.

时序网络的路径并不具备传递性，即 $A\to B$、$B\to C$ 不能推导出 $A\to C$. 这是由于路径的边均有相应的时间戳.

时序网络中边的时间戳受到如下条件限制：

(Ⅰ)假定接触序列三元组中的时间戳具有唯一性，由此可以对接触序列排序.

(Ⅱ)区间图中的时间区间没有空区间或重叠区间. 给定 $(t_i,t_{i+1})$，$(t_j,t_{j+1})\in T_e$，这两个区间满足如下条件：

$$\left.\begin{array}{l}(a)\ t_i<t_{i+1}\\(b)t_j<t_{j+1}\\(c)\text{ 若 } t_i<t_j,\text{则 } t_{i+1}<t_j\end{array}\right\} \tag{2}$$

下面给出移动通信数据时序网络表示. 移动通信数据主要有两种：移动通话数据和移动短信数据.

移动通话数据往往以移动通话日志的形式保存. 给定移动通话日志文件 CL，其中的每个数据项主要由5个数据域组成：① User；② Other；③ Direction；④StartTime；⑤Duration. 相应数据域的含义如下：

①User 数据域. User 表示当前受观察通话对象的电话号码. 该号码可以是主叫方，也可以是被叫方，其具体通信角色由 Direction 的值决定.

②Other 表示另一端通话对象的电话号码. 其通话角色与 User 相反：如果 User 是主叫方，则 Other 是被叫方；反之亦然. 在移动通话数据中，User 和 Other 号码的长度通常为11位.

③Direction 表示通话方向，其值为 Incoming 或 Outgoing. 如果 Direction 的值为 Incoming，则 Other 为通话的主叫方，User 为通话的被叫方；如果 Direction 的值为 Outgoing，则 User 为通话的主叫方，Other 为通话的被叫方.

④StartTime 数据域定义为通话记录的起始时间.

⑤Duration 代表该通话活动持续的时间长度. 当 Duration 的值为0时，表明该通话并未接通.

在有些数据集中，StartTime 和 Duration 用 StartTime 和 EndTime 表示. EndTime 表示通话的结束时间点. 其中有如下关系：

$$\text{EndTime}-\text{StartTime}=\text{Duration} \tag{3}$$

给定移动短信日志文件 sl，其中的每个数据项主要由4个数据域组成：① User；② Other；③ Direction；④TimeStamp. 其中前3个数据域与移动通话数据中的含义相同，只是这里的通讯活动是短信，而不是通话. TimeStamp 数据域表示该短信数据项的发送时间.

移动通话数据既可以表示为区间图，也可以表示为接触序列，但使用接触序列只能记录通话活动的起始时间，忽略了通话时间长度信息，而移动短信数据通常用接触序列表示.

## 2 MCDS

本节给出基于 GraphChi 的移动通信数据存储

方案 MCDS. 具体的改进包括 3 个部分：①改进 GraphChi 的数据表示格式. 修改 GraphChi 能够处理的数据类型，使其支持移动时序网络的接触序列和区间图处理；②改进 GraphChi 的分片机制；③实现新的数据分片置换算法.

### 2.1　MCDS 的数据表示格式

GraphChi 支持的数据存储格式有两种：边列表格式和邻接表格式. 这两种数据存储格式均默认使用文本文件保存. 分别介绍如下：

(Ⅰ)边列表格式. 给定边的起始顶点 $v_i$ 、$v_j$ 及该边对应的权值 $w_{ij}$ ，边列表格式中每一条边均由一行数据表示：

$$v_i v_j w_{ij}.$$

如果每一条边均使用默认权值，则可以省略 $w_{ij}$ 数据域，$v_i$ 和 $v_j$ 为数值类型. 在数据文件中，使用 # 或 % 开始的行作为注释，GraphChi 自动忽略注释行的内容. 图 2(a)中的例子是不带权值有向图的边列表格式存储结果，图 2(b)中的例子是带权值有向图的边列表格式存储结果，其中的黑体数字表示该行对应的有向边权值.

(Ⅱ)邻接表格式. 与边列表格式相比，邻接表格式存储更加高效. 每个顶点用独立的行进行存储，每一行起始为边的 ID 号，紧接着为该顶点出边的数量，然后是该顶点所有的出边邻接点列表. 给定边的起始顶点 $v_i$ ，假定与其相邻的邻接点有 $k$ 个，分别为 $v_j^1 \sim v_j^k$ ，使用邻接表格式表示为：

$$v_i^k v_j^1 v_j^2 \ldots v_j^k.$$

邻接表格式无法存储边的权值信息. 图 2(c)中的例子是不带权值有向图的边邻接表格式存储结果. 其中的黑体数字表示该行对应起始顶点的出边数量.

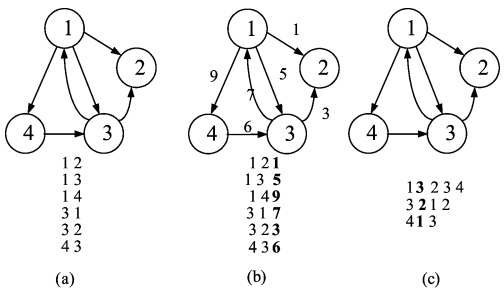



**图 2　GraphChi 数据表示格式**

**Fig. 2　Data representation format of GraphChi**

从上文分析可以看出，GraphChi 的默认数据格式无法表示接触序列和区间图. 这主要表现为：对于接触序列而言，两点之间可能存在多次通信的情况；对于区间图而言，两点之间不仅有可能存在多次通信，且其每次通信均由通信区间表示.

从上述分析可以看出，无论是接触序列还是区间图，都存在两点之间多次通信的情况，因此首先修改 GraphChi 中输入边的表示方法，允许保存带权值的多重图. 带权值的多重图有两种表示方法，以图 3 表示的接触序列为例，图 3(a)尽管更紧凑、更直接，但是该表达方式与 GraphChi 的邻接表方式冲突. 其中序列[1 2 2 3]，除了解释为节点 1 和 2 之间有两个接触序列，分别发生在 2、3 时刻外，还可以解释为邻接表方式的节点 1 和 2、1 和 3 之间存在两条有向边，因此不能使用图 3(a)中的方案. 使用图 3(b)的表示方式还有另一个好处，即该方式可以更清晰地表示区间图，如图 4 所示. 由于该区间图中的[1 2 2 3]序列同样会与邻接表格式冲突，因此 MCDS 对 GraphChi 的最终修改方法为：

(Ⅰ)修改 GraphChi，使其支持多重图表示方式；MCDS 可以表示图 3(b)中节点 1、2 之间存在两条边的情况.

(Ⅱ)取消 GraphChi 的邻接表格式；避免区间图表示方式与邻接表的格式冲突.

(Ⅲ)修改边列表格式，使其能够表示如图 4 所示的区间图数据.

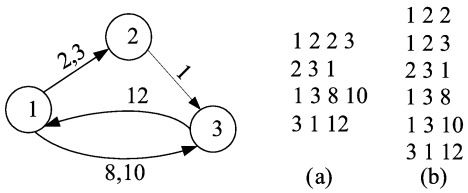



**图 3　接触序列的表示方法**

**Fig. 3　Representation of contact sequence**

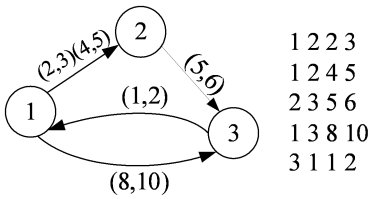



**图 4　区间图的表示方案**

**Fig. 4　Representation of interval graph**

### 2.2　MCDS 的分片机制

原始 GraphChi 仅支持某一个图的存储. 而移动通信数据通常依据某个时间间隔分割为若干个移动通信区段，每个移动通信区段均表示为接触序列或区间图. 对此，有两种解决方案：

(Ⅰ)MCDS 使用 GraphChi 原有的图存储方

案,一次处理一个移动通信区段.这样,如果有多个移动通信区段,则需要多次输入数据,使用图挖掘算法分别计算每个移动通信区段的相应结果.该方案的好处主要是:无需修改现有代码,减少工作量.该解决方案如图5(a)所示.预处理算法转换某个移动通信区段得到有向图,MCDS将该有向图的顶点分为$P$个区间,表示为$I_1 \sim I_P$.每个区间与一个对应的分片相关联.其中,$I_i$对应的分片为$S_i$($1 \leqslant i \leqslant P$).该分片中存储所有目的节点为该分片中节点的有向边.

(Ⅱ)修改GraphChi的图存储方案,典型的解决方案是图5(b)所表示的二级索引方式.第一级索引为按照时间划分的$n$个移动通信区段,第二级索引为每个移动通信区段的分片存储结构.假设有$n$个移动通信区段,则第一级索引表示为$T_1 \sim T_n$.二级索引结构针对某个移动通信区段$T_j$($1 \leqslant j \leqslant n$)的处理方式是:将$T_j$对应的图$G_j = (V_j, E_j)$分割为$P$个不连续的顶点分割区间,称为$I_1^j \sim I_P^j$.对于其中的某个分割区间$I_k^j$,$S_k^j$为其对应的数据分片.$S_k^j$中保存了目的节点在$I_k^j$中的边.$S_k^j$中的边以其源点排序.该结构的优点主要是:一次输入,其结构和使用习惯符合用户对移动时序网络的认知.该方案的最大问题在于需要修改一定数量的源代码,工作量较大.MCDS使用第二种存储方案.

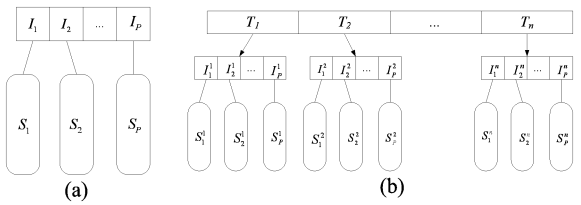



**图5 MCDS数据分片机制**

**Fig. 5 Data sharding scheme of MCDS**

GraphChi对区间大小和区间个数仅提出了一些原则上的建议:对$I_i$($1 \leqslant i \leqslant P$)分割,尽量平衡每个分片中边的数量;对$P$值大小的选择,保证每个分片可以完整地加载到内存中.

这里,使用分布式计算环境下图数据分配算法达到分片容量平衡的目的.分布式计算环境下,有两种典型的图数据分配算法:

(Ⅰ)切边法.包括谱图分割(spectral graph partitioning)和基于最小边割集算法.这些算法的基本思想是:使用某种标准选择从图中删除的边集合.从图中删除该集合中的边后,原始图形被分割为多个各自相连的子图.每一个子图分配到不同的机器上,为进一步开展并行图挖掘工作提供数据.

(Ⅱ)切点法.与切边法不同,切点法的切割并非在边上,而在顶点上.算法将切割后的子图分配给不同的机器.PowerGraph提出了均衡$P$路顶点切割算法,并提出了节点切割的贪心算法.

由于GraphChi的分割区间以顶点为依据,所以在借鉴分布式计算环境下的图数据分配算法时,仅考虑切点法.然而,切点法也无法直接应用到GraphChi中,具体由以下两个原因:

①切割线分割的图节点可能同时出现在多个切割后的子图中,而GraphChi只允许被切割的图节点存放在某个数据分片中.

②现有的切点算法大多比较复杂.图挖掘算法一般分为两个阶段:集中切割图数据并分发到不同的机器上;使用分布式图挖掘算法得到计算结果.如果采用复杂的图分割算法,则系统负载均衡较好,第二步骤中花费的时间较少.然而,复杂算法会明显增加第一阶段的计算时间,甚至抵消第二阶段带来的时间优势.

Pregel和GraphLab均使用节点随机均分法,通过哈希函数将节点均分到各机器中.与PowerGraph的切割算法相比,虽然节点随机均分法会导致各机器的数据分布不平衡,但是这种方法的优点是快速、简单、容易实现.节点随机均分法的关键在于选取何种哈希函数.常见的哈希函数包括:DJBHash、ELFHash、JSHash、RSHash及SDBMHash等.最常使用的是基于ELFHash算法的节点随机均分法.ELFHash算法在Unix的扩展库函数(extended library function)中实现,针对不同长度的字符串,该算法均有很好的鲁棒性.字符串中的每个字符作用相同,能够较均匀地散列字符串.采用ELFHash算法主要为了避免第一阶段复杂度过高.

节点随机均分法的主要思想是:将移动通信的节点标识符(电话号码作为字符串)用ELFHash算法得到其散列值.具体表示如下:

$$i = \mathrm{ELFHash}(c) \% P \tag{3}$$

式中,$c$代表当前待放入数据分片的节点标识符,$i$为该节点对应的分片位置,通过ELFHash算法,节点$c$放置到$S_i$数据分片中.MCDS中使用的ELFHash算法代码从网络上公开下载得到①.

① http://courses.cs.vt.edu/cs2604/spring02/Projects/4/elfhash.cpp

使用上述节点随机均分法，将某个移动通信区段的数据分配到数据分片后，如果每个数据分片均可独立加载到内存中，则不用调整. 否则，将该数据分片的 top- $k$ 个最大入度节点转移到拥有最少节点的数据分片中. 其中，$k$ 的值视具体情况而定.

### 2.3　MCDS 的分片置换算法

MCDS 将移动通信数据集分割为若干个移动通信区段后，再将每个区段的数据分为若干个数据分片. MCDS 在内存中尽量缓存多个数据分片. 与 TurboGraph 使用的锁-滑动机制不同，MCDS 使用修改的 ARC 算法[9].

ARC 算法维护四个链表，前两个链表保存最近访问的页面数据：LRU 页面链表，LFU 页面链表；后两个链表为幽灵链表，其中并不保存页面数据，即从 LRU 页面链表中淘汰的页面信息和从 LFU 页面链表中淘汰的页面信息.

ARC 算法的运行机制可归纳如下：

(Ⅰ)当系统调用某个页面 $q$ 进入内存时，首先使用 LRU 页面链表管理其信息.

(Ⅱ)如果 $q$ 在内存中被再次访问，则需要调整 LRU 页面链表中 $q$ 的相关信息，将 $q$ 加入 LFU 页面链表.

(Ⅲ)当需要从内存中置换页面时，LRU 和 LFU 页面链表分别使用其对应的算法淘汰表中的页面. 淘汰后的页面信息分别写入 LRU 和 LFU 幽灵链表中.

(Ⅳ)当系统查找页面 $q'$ 时，如果在 LRU 和 LFU 页面链表中找到 $q'$，则查找成功. 否则在 LRU 和 LFU 幽灵链表中查找 $q'$. 如果未找到，则表明该页面 $q'$ 不在内存中，且长期未被访问，系统在需要时执行相关的页面置换算法从外存中调入 $q'$. 如果在 LRU 幽灵链表中找到 $q'$ 信息，则表明 LRU 链表的长度不够，LRU 链表的长度加 1，LFU 链表的长度减 1. 反之亦然.

使用 ARC 算法管理 MCDS 内存分片时，需要做如下几个修改：

①依据使用的机器性能，设置一个固定的 MCDS 缓存空间. 该缓存空间不宜过大，也不宜过小. 如果过大，则影响操作系统的性能；如果过小，则 MCDS 有可能无法加载一个完整的数据分片而无法执行挖掘算法，或需要频繁置换数据分片.

②ARC 算法用来管理页面，其每个页面的大小是固定的，但是 MCDS 内存分片大小是不固定的.

③MCDS 调入某个较大的数据分片，如果此时出现内存不足的情况，有可能需要置换出一个或多个数据分片才能达到目的.

④原始的 GraphChi 系统不支持将某些数据分片锁在内存中，但 GraphChi 建议实现该功能提升系统运行效率. TurboGraph 中实现了锁机制，但是其锁定的对象是页面. 在实现 MCDS 内存分片管理时，MCDS 使用的锁机制是：在系统中额外增加一个 pin 分片链表. 一旦某个数据分片 $r$ 加入 pin 链表，则该页面信息不再出现在其他四个链表中. 仅当 MCDS 内存容量不足时，系统才将 pin 链表中的数据分片置换出内存.

在实现过程中，特别需要注意的是：在对应链表中插入、移动或删除某个节点，必须要对该链表执行加锁动作. 由于 GraphChi 的数据分片个数通常在 2～60 之间，可以使用 C++的 Placement new 机制，预先分配足够量的内存，然后在用户分配的内存上执行链表节点的构造和销毁工作. 这样做的好处主要有：在预先分配的内存上进行节点内存分配和释放，其速度较快；避免系统长期运行出现的内存碎片问题. MCDS 的数据分片置换算法如下：

**算法 2.1　CALLSHARD**

输入：
S：某个待查找的数据分片
输出：
p：指向查找的数据分片的指针
变量：
pin：pin 数据分片链表
lru：lru 页面链表
lfu：lfu 页面链表
glru：lru 幽灵链表
glfu：lfu 幽灵链表
算法：

```
1  p = NULL
2  If S is not valid Then
3  Return
4  End If
5  p = SEARCH(pin, S)//Search S in pin
6  If p Then
7  Return
8  End If
9  p = SEARCH(lru, S);//Search S in lru
10 If p Then
11 Return
12 End If
```

```
13 p = SEARCH(lfu, S);//Search S in lfu
14 If p Then
15 Return
16 End If
17 p = SEARCH(glru, S);//Search S in glru
18 If p Then
19 lru.CAPACITY = lru.CAPACITY +1
20 lfu.CAPACITY = lfu.CAPACITY -1
21 p = CACHEREP(S)//Memory swapping
22 Return
23 End If
24 p = SEARCH(glfu, S);//Search S in glfu
25 If p Then
26 lru.CAPACITY = lru.CAPACITY -1
27 lfu.CAPACITY = lfu.CAPACITY +1
28 p = CACHEREP(S)
29 Return
30 End If
```

CALLSHARD算法在内存中查找某个数据分片,如果该数据分片内容在内存中,返回;否则调用CACHEREP过程执行内存置换算法.

**算法2.2 CACHEREP**

输入:

S:待换入内存的数据分片

输出:

p:指向换入内存的数据分片的指针

算法:

```
1  p = CHECKSIZE(S)
2  If p Then
3  Return
4  End If
5  For i = 1 to lru.length
6  temp = SWAPOUT (lru, i)
7  ADDTO (glru, temp)// Add swapped shard information to glfu
8  p = CHECKSIZE(S)
9  If p Then
10 Return
11 End If
12 End For
13 For i = 1 to lfu.length
14 temp = SWAPOUT (lfu, i)
15 ADDTO (glfu, temp)// Add swapped shard information to glru
16 p = CHECKSIZE(S)
17 If p Then
18 Return
19 End If
20 End For
21 For i = 1 to pin.length
22 SWAPOUT(pin, i)
23 p = CHECKSIZE(S)
24 If p Then
25 Return
26 End If
27 End For
28 Error "S is too large to fit in memory"
```

CACHEREP算法先调用CHECKSIZE检查当前内存是否能够容纳S.如果内存不足,则依次将lfu、lru和pin链表中的数据分片调用SWAPOUT置换出内存.如果S分片仍然无法加载到内存中,则打印出错信息.CHECKSIZE和SWAPOUT的伪代码如算法2.3和算法2.4所示.其中,CHECKSIZE调用的ADDTOLRU过程和CACHEREP调用的ADDTO过程使用C++的重载函数实现,函数名均为addTo.为了伪代码的可读性,使用了不同的函数名称.

**算法2.3 CHECKSIZE**

输入:

S:待换入内存的数据分片

输出:

p:指向换入内存的数据分片的指针

全局常量:

BUFCAPACITY: MCDS缓冲空间

全局变量:

BUFSIZE: MCDS缓冲空间使用量

算法:

```
1  p = NULL
2  If S.size <= BUFCAPACITY - BUFSIZE
3  LoadSubgraph(S)// Load disk-shard to memory by GraphChi
4  p = ADDTOLRU(S)//Add S to lru list
5  End If
6  Return p
```

CHECKSIZE过程的功能较为简单,判断当前MCDS剩余缓冲空间能否存放S,如果可以,调用GraphChi的LoadSubgraph函数将S加载到内存中,并将其信息存放到lru链表中.

**算法2.4 SWAPOUT**

输入:

list:链表头指针

i: 链表中第i个数据节点

算法:

```
1  temp = SEARCH_ELEM(list, i)// Search linear list for i-th element
2  BUFSIZE = BUFSIZE + temp.size
3  temp. UpdateLastWindowToDisk ( ) //Update memory-shard to disk by GraphChi
```

SWAPOUT 过程调用 GraphChi 的 UpdateLastWindowToDisk 将对应的数据分片写回内存中.

## 3　实验结果与分析

验证 2.1 和 2.2 节的实验在 Pentium(R) D 3.0 GHz 的 PC 机器上执行,主存为 4G,运行 CentOS 4.5 操作系统. 算法使用 C++语言基于 GraphChi 0.2.1 版本编写,使用 GCC 4.5.4 编译. 这两节使用真实的移动通话数据集验证 CLRank 算法的有效性. 该数据集由中国某城市移动运营商提供. 其中包含了 30 天、12 876 437 条记录,同时给出了用户在通话期间使用的基站信息,基站数据的位置精确度在 300～500m 之间. 与 Reality 数据集[10-12]或洛桑数据集[13]相比,本数据集数据种类较少,没有 GPS 或 WLAN 等位置数据. 在验证 CLRank 之前,使用 Python 脚本从原始数据中抽取 4 个信息域:User、Other、Direction 和 StartTime. 表 1 给出了通话数据抽取结果示例. 其中,用户号码为 11 位,为了保护个人隐私,示例中没有展示后 8 位电话号码. 通话起始时间记录在 StartTime 中.

**表 1　通话数据抽取结果示例**

**Tab. 1　The example of results extracted from calls data**

| 数据域 | 内容 |
|---|---|
| User | 158******** |
| Other | 138******** |
| Direction | Incoming |
| StartTime | 08:11 |

验证 2.3 节的实验在一台 PC 机器上执行,其配置参数为:Intel i7 6 核 CPU ,主频 3.2GHz,内存 12G,配置一块 512G 的英睿达 MX100 SATA3 SSD 硬盘. 实验首先在该 SSD 上安装 CentOS 4.5 操作系统,运行 GraphChi 0.2.1 和基于 GraphChi 0.2.1 版本编写、使用 GCC 4.5.4 编译的 MCDS;然后在该 SSD 上重新安装 Windows 7 操作系统,运行 TurboGraph 得到实验结果. 这样,尽量最小化硬件所导致的结果偏差. 实验过程中,TurboGraph 默认设置 6 个执行线程.

### 3.1　不同哈希算法效果比较

MCDS 的哈希算法处理的对象主要是移动通信的节点标识符,即电话号码. 首先,系统将电话号码转换为字符串,作为哈希算法的输入. 针对不同的哈希算法,主要比较两点:不同哈希算法的性能;哈希算法的散列分布性.

实验通过从某真实电话数据集中抽取的 9 万个电话号码进行哈希运算. 参与比较的哈希函数包括:DJBHash、ELFHash、JSHash、RSHash 和 SDBMHash. 式(3)给出了 ELFHash 算法的计算公式. 其他哈希算法的计算公式将式(3)的 ELFHash 函数替换为对应的哈希函数即可. 其中,$P$ 的数值设定为 50. 由于 MCDS 的正常工作与否取决于最大数据分片能否加载到内存中,这里采用统计学上的极差(Range)概念评价哈希算法的离散分布性. 极差值为电话号码经过散列后,$P$ 个区间范围内号码的最大个数与最小个数的差.

从表 2 可以看出,ELFHash 算法运行时间最长,得到的散列分布最为集中. 这是由于实验使用的电话号码数据特性所决定的. 电话号码总长度为 11 位数字,很多电话号码之间差异较小,部分号码仅有 1 两位数字不同. 在所有的哈希函数中,仅有 ELFHash 函数能够充分利用字符串中每个字符的差异. 本文仅关注不同的哈希函数对 $I_1 \sim I_P$ 区间中节点数量极差值的影响. 由于每个电话号码所对应的入边数量是不相等的,即使 $I_1 \sim I_P$ 区间中节点数量相差不大,$S_1 \sim S_P$ 中边的数量有可能相差很大. 这里,不再使用复杂的算法进行更为准确的数据分割. 在必要的情况下,对散列结果做微调即可.

**表 2　不同哈希算法的运行时间及散列分布性比较**

**Tab. 2　Comparison of the running time and the hash distribution in different Hash algorithms**

| 哈希函数 | 运行时间(秒) | 极差值 |
|---|---|---|
| DJBHash | 624 | 5 357 |
| JSHash | 653 | 6 211 |
| SDBMHash | 676 | 5 824 |
| RSHash | 691 | 5 537 |
| ELFHash | 987 | 3 218 |

### 3.2　数据分片置换算法效果比较

数据分片置换算法的效果通常采用缓存命中率指标比较. 当系统需要使用某个数据分片时,首先在

缓存中查找,如果在缓存中找到,则命中;否则需要到外存中查找,并将其加载到内存中.该过程有可能还需要将内存中的某些数据分片置换出去.在缓存中查找到对应的数据分片次数越多,则系统的缓存命中率越高.系统访问外存的I/O开销减少,系统的整体性能得到提升.

本节使用真实的移动通信数据进行置换算法的效果比较.这里比较MCDS采用的数据分片置换算法和基于LRU的数据分片置换算法.由于数据分片大小不同,基于LRU的数据分片置换算法在原有LRU算法的基础上也要做相应的修改.将某个数据分片换入内存时,有可能置换出多个内存中的数据分片.在比较过程中,两者采用相同的分片数据和分片调用序列.

图6给出了MCDS和基于LRU数据分片置换算法的缓存命中率比较结果.在缓存空间不断增加的条件下,MCDS和LRU数据分片置换算法的缓存命中率均明显增加.其中,MCDS算法从46%增加到84%,LRU算法从41%增加到77%.在整个的增长过程中,MCDS的缓存命中率始终高于LRU算法,两者之间的差值在5%～10%之间.由于MCDS数据分片置换算法基于ARC算法,而ARC算法性能优于LRU算法,所以MCDS算法的缓存命中率结果好于LRU算法.

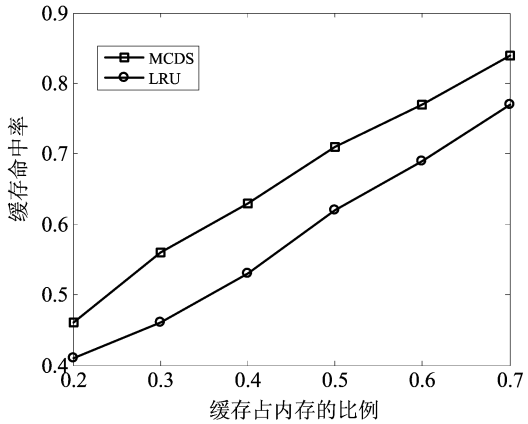



**图6 数据分片置换算法性能比较**

**Fig. 6 Performance comparison of mobile data sharding replacement algorithm**

### 3.3 MCDS系统性能比较

由于MCDS并非开源软件,很难使其支持移动时序网络数据计算任务,因此本节使用2个数据集比较MCDS、TurboGraph和GraphChi的性能:LiveJournal和Twitter.这两个数据集可以从TurboGraph网站下载①.网站上有符合TurboGraph输入要求的LiveJournal和Twitter数据集;通过对数据集提供的数据格式进行相应的调整,可以在MCDS和GraphChi中开展针对LiveJournal和Twitter数据集的运算.

在LiveJournal和Twitter数据集上运行宽度优先遍历算法,验证缓冲空间大小和执行线程的个数对MCDS、TurboGraph和GraphChi的性能影响.

图7给出了LiveJournal和Twitter数据集上改变缓冲空间大小对3个系统性能的影响.图7(a)将TurboGraph等3个系统的缓冲空间设置为3个数值:300 MB、500 MB和700 MB,分别对LiveJournal数据集执行宽度优先遍历算法.从结果可以看出,MCDS由于采用了分片置换算法,能够在系统中缓存一定数量的数据分片,运行效果优于GraphChi;但是,由于TurboGraph针对SSD做了大量的底层优化工作,MCDS的性能仍然低于TurboGraph.随着缓存空间不断增加,MCDS和

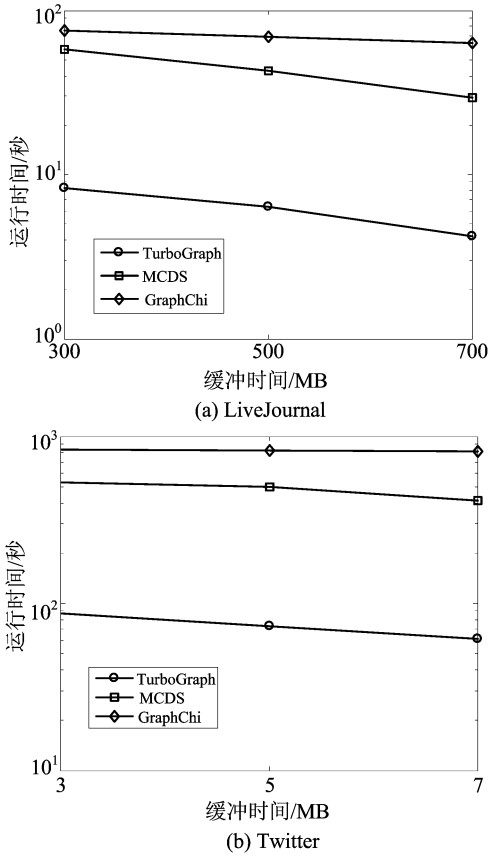


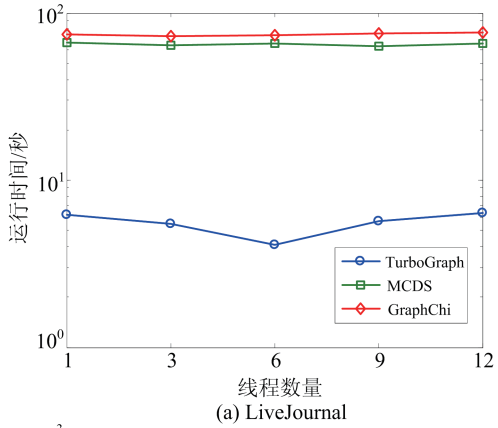

**图7 缓冲空间大小对系统性能的影响**

**Fig. 7 Influence of buffer size with system performance**

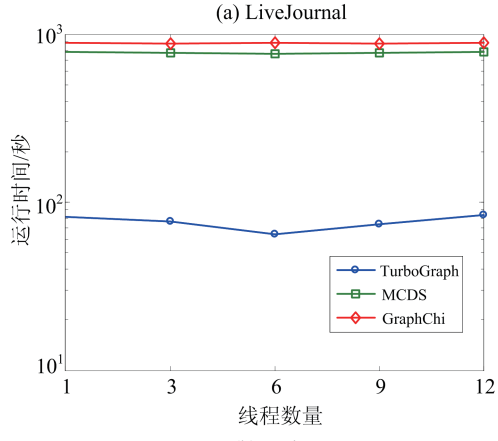
① http://wshan.net/turbograph/datasets.html

TurboGraph 的运行效率均得到明显提升，运行时间下降；但由于 GraphChi 缺乏缓存数据分片的相关机制，改变缓存空间对其系统性能影响不明显. 图 7 (b)将 TurboGraph 等 3 个系统的缓冲空间设置为 3 GB、5 GB 和 7 GB，分别对 Twitter 数据集执行宽度优先遍历算法. 其结论与在 LiveJournal 上的运行结果一致.

图 8 给出了 LiveJournal 和 Twitter 数据集上改变线程数量对 3 个系统性能的影响. 针对 LiveJournal 的实验指定所有系统的缓冲空间为 1 GB，针对 Twitter 的实验指定所有系统的缓冲空间为 8 GB. 这项设置保证 TurboGraph 和 MCDS 充分利用内存缓冲数据. 图 8 (a)和(b)将 TurboGraph 等 3 个系统的线程数量设置为 5 个数值：1、3、6、9 和 12 个，分别对 LiveJournal 和 Twitter 数据集执行宽度优先遍历算法. 从结果可以看出，MCDS 运行效果优于 GraphChi；但是 TurboGraph 的运行结果仍然优于 MCDS. 导致这项实验结果的原因在论述图 7 的过程中已经说明. 随着线程数量不断增加，




(a) LiveJournal



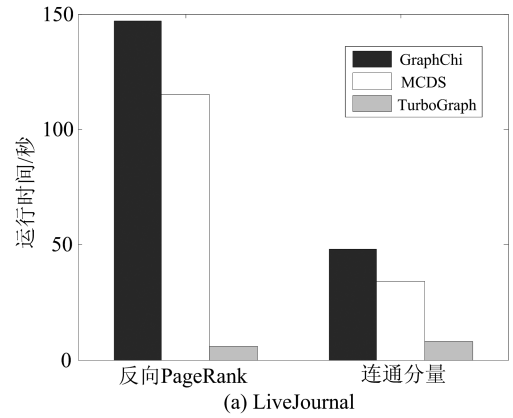

(b) Twitter

**图 8　线程数量对系统性能的影响**

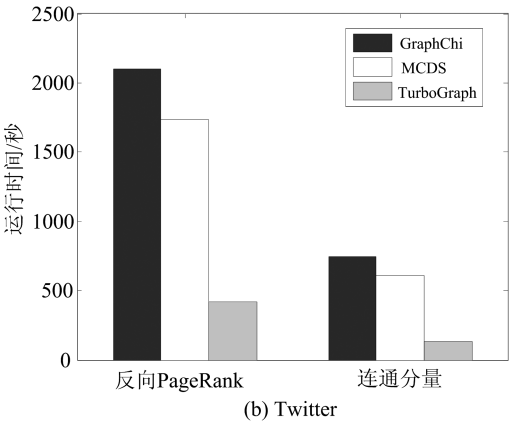
**Fig. 8　Influence of Number of Threads with System Performance**

TurboGraph 的运行效率都是先增加后降低，其运行时间先降低、后增加. 运行时间降低的原因是：TurboGraph 采用了多线程机制，提升了系统的运行效率. 然而，当线程数量增加到 6 以后，由于线程间通信需要耗费一定的 CPU 资源，所以 TurboGraph 的运行时间增加. 由于 MCDS 和 GraphChi 没有专门优化线程操作，所以改变线程数量对这两个系统性能的影响并不明显.

图 9 给出了在 LiveJournal 和 Twitter 数据集上执行反向 PageRank 和连通分量计算效率. 与图 8 所示实验相同，针对 LiveJournal 和 Twitter 的实验指定 TurboGraph 等系统的缓冲空间分别为 1 GB 和 8 GB，且 TurboGraph 使用 6 个执行线程. 在图 7 所示的两个数据集上，反向 PageRank 算法迭代了 10 次. MCDS 的运行效果优于 GraphChi；但整体性能比 TurboGraph 低. 这项结果的原因与图 7 实验结果的原因相同.




(a) LiveJournal




(b) Twitter

**图 9　反向 PageRank 和连通分量计算**

**Fig. 9　Inverse PageRank and connected component Computation**

## 4 结论

本文介绍了移动数据的类型和特点，重点介绍了移动通信数据的特性. 通过分析主流的并行和单机数据存储方案，提出了基于 GraphChi 的 MCDS 移动通信数据存储方案. 实验结果表明，MCDS 运行效率优于 GraphChi. 虽然 MCDS 移动通信数据存储方案效率低于 TurboGraph，但是 MCDS 能够支持移动时序网络需要的动态计算，具有一定的实用价值，为开展移动通信数据挖掘工作提供了基础.

**参考文献(References)**


[1] Kang U, Tong H H, Sun J M, et al. Gbase: An efficient analysis platform for large graphs [J]. The VLDB Journal, 2012, 21(5): 637-50.

[2] Kang U, Tong H H, Sun J M, et al. Gbase: A scalable and general graph management system[C]// Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Diego, USA: ACM Press, 2011: 1091-1099.

[3] Malewicz G, Austern M H, Bik A J C, et al. Pregel: A system for large-scale graph processing [C]// Proceedings of the ACM SIGKDD International Conference on Management of Data. Calgary, Canada: ACM Press, 2010: 135-146.

[4] Low Y C, Bickson D, Gonzalez J, et al. Distributed GraphLab: a framework for machine learning and data mining in the cloud[J]. Proceedings of the VLDB Endowment, 2012, 5(8): 716-27.

[5] Gonzalez J E, Low Y C, Gu H J, et al. PowerGraph: Distributed graph-parallel computation on natural graphs [C]// Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation. Berkeley, USA: USENIX Association, 2012: 1-5.

[6] Kyrola A, Blelloch G, Guestrin C. GraphChi: Large-scale graph computation on just a PC [C]// Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation. Berkeley, USA: USENIX Association, 2012: 31-46.

[7] Han W S, Lee S, Park K, et al. TurboGraph: A fast parallel graph engine handling billion-scale graphs in a single PC[C]// Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM Press, 2013: 77-85.

[8] Holme P, Saram ? ki J. Temporal networks[J]. Physics Reports, 2012, 519(3): 97-125.

[9] Megiddo N, Modha D S. ARC: A Self-Tuning, Low Overhead Replacement Cache[C]// Proceedings of the 2nd USENIX Conference on File and Storage Technologies. San Francisco, USA: USENIX Association, 2003: 115-130.

[10] Eagle N, Pentland A. Reality mining: sensing complex social systems [J]. Personal and Ubiquitous Computing, 2006, 10(4): 255-68.

[11] Ficek M, Kencl L. Spatial extension of the reality mining dataset[C]// Proceedings of the International Conference on Mobile Adhoc and Sensor Systems. San Francisco: IEEE Press, 2010: 666-673.

[12] Pentland A. Reality Mining of Mobile Communications: Toward a New Deal on Data [M]. Springer, 2009.

[13] Kiukkonen N, Blom J, Dousse O, et al. Towards rich mobile phone datasets: Lausanne data collection campaign [C]// Proceedings of the International Conference on Pervasive Services. Berlin, Germany: ACM Press, 2010: 1-7.