EEXPSort－树形数据的高能效外存排序算法①

王 靖, 周煜超, 龚卫华, 杨良怀

(浙江工业大学 计算机科学与技术学院, 杭州 310014)
(浙江省可视媒体智能处理技术研究重点实验室, 杭州 310014)

摘 要: 树形数据排序是XML数据处理中一个基本问题. 提出了一种XML文档高能效排序算法－EEXPSort. 该算法扫描XML文档产生相互独立的排序任务, 利用多核CPU对排序任务进行并行处理; 同时采用数据压缩、单临时文件存储以及避免子树匹配等策略, 有效地减少磁盘IO和CPU操作时间. 对不同特性的XML文档开展了大量比较实验, 结果表明所提算法能效优于现有性能最好的树形数据排序算法HERMES.

关键词: XML文档; 树形数据; 能效; 排序算法; 优化策略

EEXPSort-an Energy-Efficient Sorting Algorithm for Hierarchical Data in External Memory

WANG Jing, ZHOU Yu-Chao, GONG Wei-Hua, YANG Liang-Huai

(School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310014, China)
(Key Laboratory of Visual Media Intelligent Process Technology of Zhejiang Province, Hangzhou 310014, China)

Abstract: A fundamental problem in XML data handling is hierarchical data sorting. This paper presents an energy-efficient sorting algorithm called EEXPSort for XML document. It exploits multi-core CPU to parallelize the executions of the mutually independent tasks generated by scanning the XML document; For energy-efficiency, it employs data compression, single temporary-file storage and avoidance of tree-matching to effectively reduce disk IOs and CPU process. Extensive experiments on XML documents with different characteristics show that EEXPSort outperforms the existing quickest XML sorting schemes HERMES significantly in energy-efficiency and performance.

Key words: xml document; hierarchical data; energy-efficient; sorting algorithm; optimization strategy

1 引言

能效计算[1]是当今计算机领域面临的一个迫切问题. 近年来, 数据库领域在能效方面开展了相应的研究. 排序在计算机领域具有极其重要的作用, 是最广泛研究的热点之一[2,3]. 目前面向能效的排序算法研究都是针对扁平数据, 对大规模数据的排序通常采用外部归并排序法[2,4].

随着XML成为因特网数据表示、交换的标准, 基于XML的应用和系统也越来越多, 树形数据排序有其重要应用. 树形数据排序已有一些研究, 如排序工具XSort[5], XML树形数据排序算法NEXSORT[6]和HERMES[7].

但现有的树形数据排序方法存在大量的随机I/O对存储空间的利用有待优化, 没有考虑多核计算的应用, 且未考虑排序对能效的影响. 为解决以上问题, 本文提出面向能效的树形数据排序算法EEXPSort(Energy-Efficient XML Parallel Sort). EEXPSort采取优化内存中子树的管理策略, 避免子树匹配, 减少内存树的维护代价; 利用单文件存储中间结果、数据压缩、归并阶段引入缓存、多核计算等策略, 减少磁盘I/O和加速文档排序; 克服了NEXSORT没能有效利用内存空间、存在大量随机I/O的问题以及难以处理“右深树”的缺陷, 也克服了HERMES的数据冗余、大量磁盘开销等缺点. 实验结果显示了EEXPSort在能效上的有效性, 它优于现存算法.

剩余部分组织如下: 第2节详细介绍了EEXPSort

① 基金项目:国家自然科学基金(61070042);浙江省自然科学基金(Y1090096)
收稿时间:2012-05-03;收到修改稿时间:2012-06-07

的算法原理和实现; 第 3 节对 EEXPSort 进行能效评价; 最后进行小结.

2 EEXPSort算法

树形数据排序的基本单元是一个节点的直系子节点集, 这些排序任务是相互独立的. EEXPSort 首先通过先序方式遍历 XML 文档, 将扫描获得的排序任务(节点集)保存到内存的任务队列中; 然后在内存不足时, 将这些独立的排序任务进行并行排序后, 输出相关节点集或子树到外存的临时文件中, 并释放相应内存; 继续上述排序过程直到文档扫描结束, 作最后的归并.

2.1 基本概念

EEXPSort 扫描 XML 文档, 按先序方式建立内存树. 对应 XML 文档的每个元素, EEXPSort 在内存中建立树节点 Node, 节点结构及属性如下:

```
Struct   Node   // XML 节点结构
{
 String label;   //标签名
 Attribute attributes[];   //属性
 String text;   //排序字段
 Node children[];   //子节点
 Posi_info posi[];   //子节点集在外存的位置信息
};
```

在后续叙述中要用到的几个概念: (1)完全扫描树: 若子树 T 的所有节点都已被扫描, 且子树 T 在内存中的节点个数大于 1, 则称子树 T 是一棵完全扫描树. (2)完全节点: 若一个节点的所有直系子节点都已被扫描且全部位于内存, 则称该节点是一个完全节点. 我们规定叶节点不属于完全节点. (3)结束标签对应子树: 在 XML 文档中, 由一个结束标签</label>和与它对应的开始标签<label>之间的所有节点组成的子树, 称为该结束标签对应子树. (4)完全扫描树集合(TC): TC 是当前内存中的所有互不包含的完全扫描树集合, 满足: $\forall T_1,T_2 \in TC$, 必定不存在 $T_1 \subseteq T_2$ 或 $T_2 \subseteq T_1$ 的子树包含关系. 维护 TC 中子树互不包含的目的是在将子树保存到外存时避免不必要的重复存储. 因此, 在将新获得的完全扫描树 T 添加到 TC 前, 必须检查 TC 中已有子树与 T 的包含关系: $\forall T' \in TC$, 若满足 $T' \subseteq T$, 则从 TC 中剔除 T', 称这一过程为完全扫描树集合(TC)的相容性检查.

2.2 EEXPSort 实现

EEXPSort 整体算法描述如图 2 所示. 若 EEXPSort 扫描到的元素是一个 XML 开始标签或者是一串纯文本, 创建新节点并在内存树中插入; 若内存空间不足, 则先将部分内存树节点经过并行排序后保存到外存, 腾出空间, 然后创建新节点并在内存树中插入(第 2~9 行), 其中涉及的有关内存清理的过程(第 7 行)将在稍后进行说明.

若扫描遇到一个 XML 结束标签, 则该结束标签对应子树 T 的所有节点都已被扫描. 若子树 T 是一棵完全扫描树, 则记其根节点为 T.root, 令 node←T.root, 进行以下操作:

① 若 node 是一个完全节点, 则进行如下处理(第 11~14 行): (1)将 node 放入任务队列 TaskQueue, 表示 node 的子节点集是一个待排序的排序任务; (2)将完全扫描树 T 添加到完全扫描树集 TC, 检查 TC 的相容性. 通过记录子树包含的完全扫描子树的个数, 然后剔除 TC 尾端相应个数的直系完全子树的方法可以很好地避免相容性检查时子树间的穷举地检查包含关系, 减少 CPU 负载, 这一过程称为优化的相容性检查.

② 若 node 不是一个完全节点, 对其进行如下处理(15~20 行): (1)对 node 的内存子节点进行排序; (2)保存到外存; (3)在 node 的 posi[]中记录所保存节点集的位置信息.

③ 若 node 是内存树的根节点, 表明 XML 文档扫描完毕. 最后根据树节点中保存的地址信息 posi[], 将内存和外存上的节点进行连接, 归并为一个完整、有序的 XML 文档(第 24 行). 在归并期间, EEXPSort 为 XML 树的每一层分配了一个页大小的内存缓存空间, 每次从外存预读取一定规模的子节点集数据, 从而减少磁盘访问和 I/O 随机访问.

如前所述, 随着扫描工作的进行, 内存中的树节点不断增多, 内存空间不断减少. 当内存空间不足时, 为了能在内存中继续保存扫描到的节点, 必须将内存中的部分树节点保存到外存, 以腾出内存空间, 这个过程称为内存清理(第 5~7 行).

内存清理的具体操作如下: (1)调用线程池并行排序任务队列 TaskQueue 中的所有任务; (2)保存 TC 中的每棵完全扫描树 T 到外存, 并将树 T 在外存的位置信息记录在 T.root 的 posi[]之中; (3)释放除根节点 T.root 外树 T 占用的内存空间; (4)待 TC 中的所有完全扫描树

输出完毕后, 清空 TC. 例如, 对图 1 左侧的内存树进行内存清理操作时, EEXPSort 根据当前 TC 中的完全子树记录{A, B, E}, 将子树 A, B, E 保存到外存, 分别将每棵子树在外存的位置信息保存到子树根节点中, 最后释放相应节点的内存空间, 清空 TC.

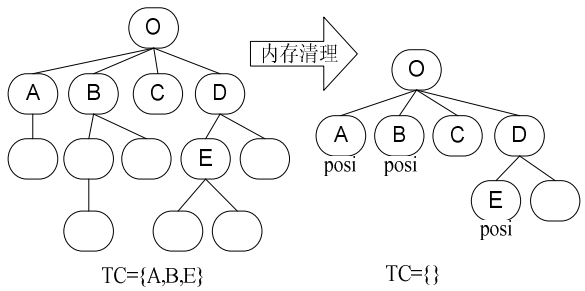

图 1　对含有完全扫描树的内存树进行内存清理

```
Algorithm : EEXPSort
Input: XML 文档;
Output: 有序的 XML 文档;
变量: TC: 完全扫描树的集合;
TaskQueue: 排序任务队列;
1.扫描 XML 文档直到完成:
2. 读取一个 XML token;
3. if token∈{开始标签, 纯文本}
4.   if out of memory
5.     传递 TaskQueue 到线程池进行排序;
6.     清空 TaskQueue;
7.     OutputNodes(TC); //内存清理
8.   endif
9.   为 token 创建节点并插入到内存树;
10. else /*token 是一个结束标签*/
11.   node ← T.root; //它是 token 对应的子树 T 的根节点
12.   if  node 是完全节点
13.     TaskQueue ← node; //排序任务保存到到队列
14.     TC ← T; //进行相容性检查
15.   else /* node 不是完全节点*/
16.     Sort(node 的内存子节点);
17.     保存排好序的内存子节点(及其子树)到外存;
18.     node 的 posi[]之中记录相应外存位置信息;
19.     释放 node 那些已经保存在外存的节点所占用的内存空间;
20.   endif
21.   if node 是内存树的根节点 /*XML 文档扫描完毕*/
22.     传递 TaskQueue 到线程池进行排序;
23.     清空 TaskQueue;
24.     LastMerge(node); //进行最终归并
25.     Return;
26.   endif
27. endif
```

图 2　EEXPSort 算法伪码

假设 XML 文档占 B 内存页, 可用内存 M 页, 其中一页做输出缓存. EEXPSort 与 XPSort 具有相同的 IO 开销 $O(2B(1+\log_{M-1}\lceil B/(M-1)\rceil))$.

3　能效评价

本节对 EEXPSort 进行实验评价. 利用 Xerces-C++XML Parser, 用 C++实现了 EEXPSort、HERMES[7], 比较了这两种 XML 排序算法的能效. 实验中使用 Intel 酷睿 2 四核 Q8200 处理器, 2.33GHz 主频, 4GB 内存, Windows 7 操作系统, 硬盘希捷(7200RPM). 实验对 EEXPSort 与目前性能最好的 XML 排序算法 HERMES 进行能效比较, 同时考察不同优化策略对 EEXPSort 能效的影响.

3.1 多任务模式下 EEXPSort 的能效统计方法

能效(能量效率)=所做工作/所用能量, 即每能量单位所完成的工作. 给定工作量, 最大化能效就是最小化能耗, 所以可以通过比较排序算法在相同工作量下的能耗情况来比较能效. 与排序操作密切相关的能耗设备主要有 CPU、内存和外存.

(1) CPU 能耗统计

CPU 是主要的能耗设备. 在统计多任务模式下的多核 CPU 能耗时, 不同任务的多个线程会同时占用 CPU, 这里只考虑多任务的理想状态, 即忽略其他任务对当前任务排序线程的影响, 分别统计排序任务的每个线程占用物理核的工作时间以及单个物理核的功率即得到排序算法的 CPU 能耗: $Energy_{CPU}=Power_{core}*Time_{thread}$, 其中 $Energy_{CPU}$ 表示 CPU 能耗, $Power_{core}$ 表示 CPU 上单个物理核的功率, $Time_{thread}$ 表示参与排序的所有线程的总工作时间. 实验使用的处理器为四核, 功率为 95W, 所以单个物理核的功率 $Power_{core}$ =95 W/4 =23.75W.

(2) 内存能耗统计

为获得排序算法的内存能耗, 假定内存能耗与内

存使用量成正比, 可通过采集不同时刻排序进程对内存的占用量, 累计获得整个排序过程的内存能耗: 内存能耗 $Energy_{ram}=Power_{ram}*Time_{ram} = \int Power_{MB}*Size(t)dt=Power_{MB}*\int Size(t)dt$, 其中 Energyram 表示内存能耗, Powerram 表示内存的功率, Timeram 表示内存使用的时间, $Power_{MB}$ 表示每兆字节(MB)内存的功率, Size(t)表示在 t 时刻排序进程对内存的占用量, $\int Size(t)dt$ 表示排序期间排序进程的内存占用量和时间的积分(以下简称内存占用面积). 实验使用的内存功率为 3W, 总大小为 4GB, 所以每兆字节的内存功率 $Power_{MB} = 3W/4GB = 7.5*10^{-4}W/MB$.

(3) 外存能耗统计

在理想状态的多任务模式下, 排序任务的外存能耗包括读取外存数据的能耗和写入外存数据的能耗, 而其他时间段外存处于空闲状态或者被其他任务占用所消耗的能量不计入当前任务的外存能耗. 外存能耗 $Energy_{disk} = Power_{read} * Time_{read} + Power_{write} * Time_{write}= Power_{read} * (File_{read} / Rate_{read}) + Power_{write} * (File_{write} / Rate_{write})$, 其中 $Energy_{disk}$ 表示外存能耗, $Power_{read}$ 表示外存读功率, $Power_{write}$ 表示外存写功率, $Time_{read}$ 表示外存读操作花费的时间, $Time_{write}$ 表示外存写操作花费的时间, $File_{read}$ 表示从外存读取的数据量, $File_{write}$ 表示向外存写入的数据量, $Rate_{read}$ 表示外存读操作的速度, Ratewrite 表示外存写操作的速度. 实验使用的外存是型号为希捷 st3500418as 的硬盘, 其 $Power_{read}$=7W, $Power_{write}$=7W, $Rate_{read}$=100MB/s, $Rate_{write}$=100MB/s.

表 1 生成 XML 文档例子及文档特征

元素个数(million)	文件大小(MB)	平均扇出度
10	243	56
20	487	59
40	974	60
60	1454	61
80	1945	59
120	2918	60

3.4 CPU 能耗

若无特别说明, 本节实验在 100MB 内存下进行.

(1) EEXPSort 与 HERMES 的 CPU 能耗比较

EEXPSort 与 HERMES 对表 1 的 XML 文档组进行排序时在 CPU 上的能耗比较结果如图 3 所示. 其中, 表 1 的 XML 文档组深度上限为 6, 平均文本长度为 20bytes. 由实验结果可知, 随着文档规模的增加, EEXPSort 的 CPU 能耗总是少于 HERMES, 即 EEXPSort 在 CPU 方面更具能效. 对 80×10^6 个元素个数的 XML 文档排序时, HERMES 的 CPU 能耗是 EEXPSort 的 1.85 倍.

HERMES 采用替换选择的方式进行排序, 整个排序阶段需要对每个兄弟节点集做维护, 保证输出序列的有序性, 该方式增加了 CPU 的处理负载. 另外 HERMES 存在冗余节点和引入唯一标识符, 带来一些额外开销; 而 EEXPSort 毋须维护兄弟节点集, 也不存在冗余节点和唯一标识符的问题, 所以 EEXPSort 比 HERMES 花费更少的时间, 消耗更少的 CPU 能量.

(2) 优化的相容性检查策略对 CPU 能耗的影响

本实验考察 EEXPSort 中优化的相容性检查策略对 CPU 能耗的影响, 对表 1 的 XML 文档组进行排序, 结果如图 4 所示. 在对 80×10^6 个元素的 XML 文档排序时, 使用相容性检查优化策略前后的 CPU 能耗相差 14.7 倍. 在未使用相容性检查优化策略时, XPSort 检查完全扫描树集合 TC 中的每棵子树是否包含于当前子树 T 的直系子节点中, 树匹配的代价是 O(nm), 其中 n、m 分别是 TC 和 T 包含的子树个数; 而优化后的检查策略只需根据先前记录的子树 T 包含的直系完全扫描树个数剔除 TC 尾端相应个数的子树就可直接剔除 TC 中包含于 T 的子树, 检查代价是 O(m). 优化后的相容性检查策略可以避免大量的子树匹配操作, 减少 CPU 的负载和能量消耗.

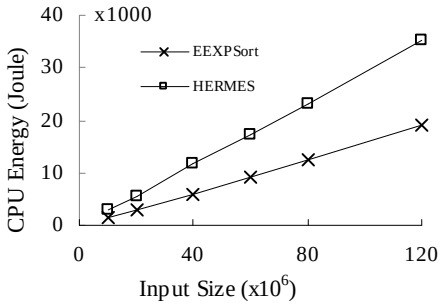

图 3 EEXPSort 与 HERMES 的 CPU 能耗比较

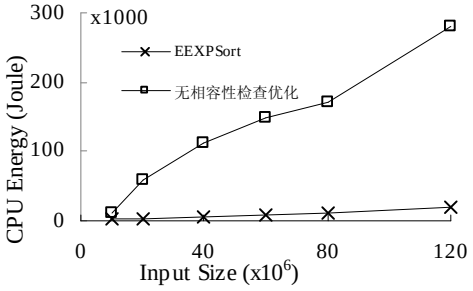

图 4 优化的相容性检查策略对 CPU 能耗的影响

3.5 内存能耗

为比较 EEXPSort 与 HERMES 的内存能耗, 在不同的内存空间下(100MB~1024MB)对表 1 中元素个数为 80×10^6 的 XML 文档排序, 结果如图 5 所示. 可知, 随着内存空间的增加, EEXPSort 的内存能耗总是少于 HERMES. 当内存为 100MB 时, HERMES 的内存能耗是 EEXPSort 的 3.59 倍; 当内存增加到 1024MB 时, HERMES 的内存能耗是 EEXPSort 的 2.81 倍. HERMES 在内存空间不足时, 只将一棵子树保存到外存, 内存清理后内存中仍然存在大量树节点, 内存占用量很大; 而 EEXPSort 在清理内存时, 将内存中所有的完全子树保存到外存, 清理后的内存占用量很小. 因此, HERMES 会比 EEXPSort 占用更大的内存面积, EEXPSort 在内存上的能耗更少.

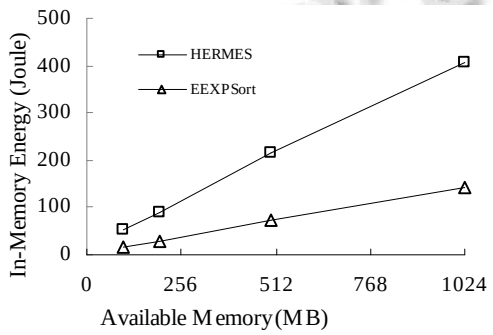

图 5　EEXPSort 与 HERMES 的内存能耗比较

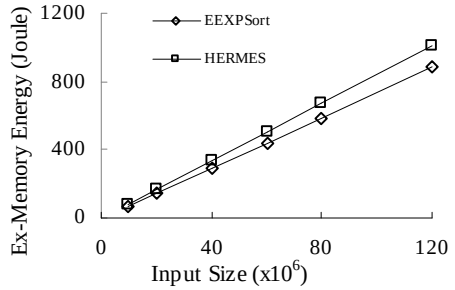

图 6　EEXPSort 与 HERMES 的外存能耗比较

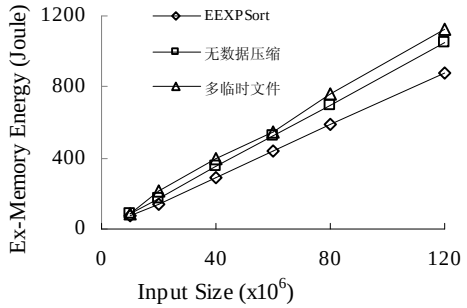

图 7　优化策略对外存能耗的影响

3.6 外存能耗

若无特别说明, 本节实验在 100MB 内存下进行.

(1) EEXPSort 与 HERMES 的外存能耗比较

EEXPSort 与 HERMES 分别对表 1 的 XML 文档组进行排序时外存上的能耗比较果如图 6 所示. 随着文档规模的增加, EEXPSort 的外存能耗总是少于 HERMES. 在对 80×10^6 个元素个数的 XML 文档排序时, EEXPSort 在外存上的能耗比 HERMES 节省了 12%. 究其因, HERMES 在排序过程中会产生较多的冗余节点, 并且每个节点需要分配一个唯一标识符, 这使得 HERMES 的临时文件占用更多的外存空间, 从而增加了外存数据的读写时间, 消耗更多的外存能量.

(2) 优化策略对外存能耗的影响

实验考察数据压缩策略和单临时文件策略对 EEXPSort 外存能耗的影响, 对表 1 的 XML 文档组进行排序, 结果如图 7 所示. 可知, 随着文档规模的增加, 使用数据压缩策略和单临时文件策略的 EEXPSort 总是比未使用的 EEXPSort 花费更少的外存能耗. 对 80×10^6 个元素的 XML 文档排序时, 使用两种优化策略的 EEXPSort 在外存上的能耗比未使用压缩策略 EEXPSort 节省了 16%, 比采用多临时文件策略的 EEXPSort 节省了 23%. 这是因为数据压缩策略通过减少每棵子树在外存中的存储空间, 降低了临时文件在外存中的占用空间, 从而减少了外存数据的读写时间, 节省了外存能耗; 单临时文件策略通过将所有子树保存到一个临时文件, 增加外存数据访问的顺序性与紧凑性, 进而降低了临时文件在外存中的占用空间, 降低了外存能耗.

4　结语

本文提出了面向能效的 XML 文档排序算法 EEXPSort, 克服了已有 XML 排序算法 NEXSORT 和 HERMES 存在的不足, EEXPSort 有效利用内存空间, 充分利用多核 CPU 对独立的排序任务进行并行处理, 同时利用数据压缩、归并时引入缓存、单临时文件存储以及避免子树匹配等优化策略, 有效地减少磁盘 I/O 和 CPU 处理负载, 取得了较好的排序能效. 实验表明, EEXPSort 的能效在各方面(CPU、内存、外存)均优于目前最快的树形数据排序算法 HERMES, 所提算法是有效的.

(下转第 107 页)

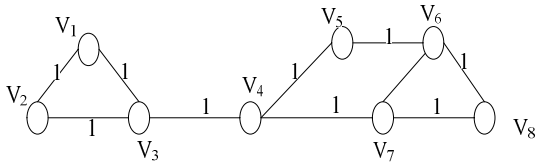

图 3 DS 实例图

5 结语

网络的传输特性和节点间最短距离是判断网络节点重要性的两个重要因素. 本文结合这两个因素, 提出了一种评价网络中节点重要性的新方法, 并给出了公式化的表达式. 通过比较各节点的通信流量和节点失效后网络间最短距离总和的变化, 可以有效地区分网络中任意节点的相对重要性. 与节点删除法相比, 具有更高的实用性、精确性, 是一种可靠的节点重要性评价方法.

参考文献

1 Chen Y, Hu AQ, Yip KW, Hu J, et al. Finding the most vital node with respect to the number of spanning trees. IEEE Int. Conf. neural Networks & Signal Processing, 2003, 12: 1670–1673.

2 陈勇,胡爱群,胡骏.通信网中最重要节点的确定方法.高技术通讯,2004,14(1):21–24.

3 Wu RZ, Hu XY, Tang LR. Node Importance Evaluation Based on Triangle Module for Optical Mesh Networks. IEEE Conferences. 2011,7:1–4.

4 Hu J, Wang B, Lee DY. Evaluating Node Importance with Multi-criteria. IEEE Conferences. 2010,12:792–797.

5 董志远,张品,陈磊.一种基于两测度的无线链路重要性评价方法.杭州电子科技大学学报,2011,31(5):159–162.

6 王建伟,荣莉莉,郭天柱.一种参数可调的网络节点重要性度量方法.科研管理,2009,30(4):74–79.

7 张品,陈磊,姜亚光.无线网络中节点重要性的研究.电子器件,2011,34(4):395–397.

8 姜禹,胡爱群,潘婷婷.一种评价通信网节点重要性的新方法——节点孤立法.高技术通讯,2008,18(7):673–678.

9 王延庆.复杂网络节点重要性评估.网络安全技术与应用,2008,3(3),59–61.

10 赫南,李德毅,朱熙,等.复杂网络中重要性节点发掘综述.计算机科学,2007,34(12):1–5.

11 Kubat P. Estimation of reliability for communication/ computer networks simulation/analytic approach. IEEE Trans. on Communication, 1989,37(9):927–9.

(上接第 112 页)

参考文献

1 Rivoire S, Shah MA, Ranganathan P, Kozyrakis C. JouleSort: A Balanced Energy-Efficiency Benchmark. ACM SIGMOD Conference, 2007:365–376.

2 Knuth DE. The Art of Computer Programming. Sorting and Searching. 2nd ed, Addison-Wesley, 1998, 3.

3 Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to Algorithms. 2nd ed., MIT Press, 2001.

4 Graefe G. Query Evaluation Techniques for Large Databases. ACM Computing Survey, 1993,25(2):73–169.

5 Campillo IA, Green TJ, Gupta A, Onizuka M. XMLTK: An XML toolkit for scalable XML stream processing. Proc. of PLAN-X: Programming Language Technologies for XML, 2002:1–10.

6 Silberstein A, Yang J. NEXSORT: Sorting XML in External Memory. Proc. of the 20th International Conference on Data Engineering. 2004: 695–707.

7 Koltsidas I, Müller H, Viglas S. Sorting hierarchical data in external memory for archiving. Proc. of the VLDB Endowment Archive, 2008,1(1):1205–1216.

(上接第 195 页)

ECU 通讯协议编写后续的数据分离程序, 便可以得到发动机各个参数的实时数据, 本系统在汽车发动机台架试验监测具有一定的参考价值.

参考文献

1 廖明明,林伟建.发动机台架试验浅析.装备制造技术,2010:167–169.

2 余淼,刘胜龙,朱李晰,等.汽车发动机 ECU 的可靠性试验研究.内燃机工程,2010,22(11–1):90–93.

3 陈锡辉,张银鸿. LabVIEW 8.20 程序设计从入门到精通.北京:清华大学出版社,2008.

4 张亿雄,顾海明.基于调用 DLL 的 LabVIEW 数据采集的实现.微计算机信息,2008,24(12–1):78–90.

5 秘晓元,张严斌,薛德庆,等.LabVIEW 中利用 LabSQL 访问数据库.微计算机信息,2004,20(10):53–54.