基于多核处理器的VTD-XML节点查询执行性能优化

郭宪勇[1] 陈性元[1] 邓亚丹[2]
(解放军信息工程大学电子技术学院 郑州 150001)[1] (北方信息技术研究所 北京 100072)[2]

摘 要 针对目前主流的多核处理器,研究了基于VTD-XML的节点查询执行性能优化,即基于预读策略从多线程并发执行和提高线程内存访问性能两个方面优化XML节点查询的性能。实验结果表明,提出的多线程XML文档解析框架可以充分利用多核处理器的计算资源,并有效地提高线程的内存访问性能,大大提高了XML节点查询的性能。

关键词 VTD-XML,多核处理器,XML节点查询执行优化,多线程
中图法分类号 TP301 **文献标识码** A

VTD-XML Node Query Execution Performance Optimization Based on CMP

GUO Xian-yong[1] CHEN Xing-yuan[1] DENG Ya-dan[2]
(Institute of Electronic Technology, the PLA University of Information Engineering, Zhengzhou 150001, China)[1]
(Northern Institute of Information Technology, Beijing 100072, China)[2]

Abstract For mainstream multi-core processors, the VTD-XML's node query execution performance was optimized, based on preload method, and from the concurrent execution of multiple threads and thread memory access performance. The experimental results show that the multihreaded XML document parsing framework proposed in this paper can take full advantage of the computing resources of multi-core processors, and effectively improve thread memory access performance, greatly improve the performance of XML node query.

Keywords VTD-XML, Chip multiprocessor, XML node query execution optimization, Multithread

1 引言

XML(Extensible Markup Language)是可扩展标记语言的简称。XML作为Web应用的核心数据格式受到越来越广泛的应用,比如SOA架构中的核心数据交换协议SOAP就是基于XML数据格式的。但是XML文档处理是一种处理代价很大的运算[1],比如XML文档解析、XML查询执行和XML数据序列化/反序列化操作等。因此,目前有大量学者致力于提高XML文档的处理性能。

2 相关研究现状

2.1 XML文档处理模式分析

目前常见的XML文档处理模式有DOM和SAX,而新兴的VTD-XML(Virtual Token Descriptor XML)则是目前性能最优的一种XML文档处理模式[2]。与DOM不同,VTD-XML中XML文档解析的结果不是树状结构,而是一维数组[2],数组中每个数据项存储的是XML文档中Token的信息,比如Token在XML文档中起始偏移量、长度和深度等关键信息。由于VTD-XML是最新的XML文档处理模式,针对它的性能优化的研究较少。综上所述,本文采用的XML文档处理模式为VTD-XML。

2.2 多线程XML查询执行研究现状

随着多核处理器的普及,目前已有不少基于并行处理开展XML查询执行性能优化的研究工作。文献[3,4]针对分布式的XML处理开展研究;文献[5]对半结构化的数据通过分布式查询优化其性能;文献[6]基于工作队列组织XML查询的执行,当某条工作队列中的XML查询执行完毕,可以从其它工作队列中获取未执行的XML查询,以提高处理器的利用效率;文献[7]基于多核处理器,针对XPath查询提出数据划分、查询划分以及混合划分3种并行XPath执行策略,实验表明并行XPath性能相比串行有了较大提高,性能提高幅度与处理器数量成线性递增关系。

目前VTD-XML支持的查询种类有节点查询和XPath查询,节点查询是指用户在无法获知XML文档路径信息的情况下,以节点的节点名和节点属性值作为查询条件,VTD-XML支持标准的XPath查询执行。本文针对节点查询执行优化展开研究,目前尚未有基于VTD-XML的查询执行性能优化研究成果。

3 多线程XML节点查询执行框架

下面首先给出多线程XML节点查询执行框架,以四核

到稿日期:2013-03-27 返修日期:2013-07-10 本文受973计划前期研究专项(2011CB311801),国家自然科学基金项目(60903220)资助。
郭宪勇(1965—),男,博士生,高级工程师,主要研究方向为信息安全技术,E-mail:xianyongguo@yahoo.cn;陈性元(1963—),男,博士生导师,主要研究方向为信息安全技术。

处理器为例，多线程 XML 节点查询执行框架如图 1 所示。

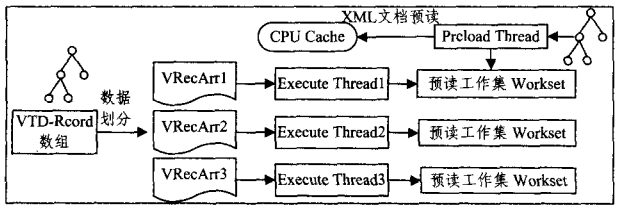

图 1　多线程 XML 节点查询执行框架

该框架包括两类线程：查询执行线程(Execution Thread)和预读线程(Preload Thread)，其中查询执行线程执行 XML 节点查询处理。同时，由于目前内存访问速度远远低于处理器，造成 Cache 访问缺失导致的内存访问也已经成为代价昂贵的运算[9-11]，在数据密集型的程序中更是如此，而 XML 文档处理则是典型的数据密集型运算。该框架中的预读线程则是为了提高查询执行线程的内存访问性能，将查询执行线程要访问的 VTD-Record 数组和 XML 文档数据提前取至处理器的 Cache 中，通过减少查询执行线程的 Cache 访问缺失提高其内存访问性能。下面详细介绍多线程 XML 节点查询执行框架的细节。

为了能够并行执行 XML 查询，需要将 VTD-Record 数组划分成多个子集(如图 1 所示)，每个 *VRecArr* 为 VTD-Record 数组的一部分，每个查询执行线程处理对应的 *VRecArr*。根据 VTD-XML 的设计，每个 VTD-Record 对应 XML 文档中的一个 Token，不同 Token 的字符串长度存在差异，如果简单地将 VTD-Record 数组均匀划分，则会造成查询执行线程处理的 XML 文档数据量不均衡而影响整个查询执行的性能。针对此问题，本文提出了基于 Token 大小的动态数据划分策略。首先给出 VTD-Record 的形式化定义。

定义 1(VTD-Record，VTD 数据项)　$VTD\text{-}Record=(TokenType, Offset, TokenLen, Depth)$，其中 $TokenType$，$Offset$，$TokenLen$，$Depth$ 分别表示 Token 的类型、在 XML 文档中的起始偏移量、长度和深度。

令查询执行线程个数为 *QENum*，该数据划分策略首先将 VTD-Record 数组均匀划分为 *QENum* 份($VRecArr^i, i=1\cdots QENum$)，并根据 $VRecArr^i$ 中最后一个 VTD-Record 中的 $Offset$ 和 $TokenLen$ 计算出 $VRecArr^i$ 对应的 XML 文档大小，如果存在 $VRecArr^i$ 对应的 XML 文档大小差异较大，则减少 XML 文档较大的 $VRecArr^i$ 所包含的 VTD-Record 数量，相应地增加 XML 文档较少的 $VRecArr^i$ 所包含的 VTD-Record 数量，直到 XML 文档大小在 $VRecArr^i$ 间均匀分布为止。数据划分完成后便可启动多个查询执行线程，每个线程处理一个 *VRecArr*。

查询执行线程执行的同时，预读线程将查询执行线程要访问的 VTD-Record 数组和 XML 文档中的 Token 预读至处理器 Cache 中。具体方法为：假设 XML 文档中 Token 对应数据的起始地址为 p，预取时只需通过执行 $temp=*((int*)p)$，即可将包含 p 在内的 B(B 为处理器 Cache Line 的大小)个字节大小的数据取至处理器 Cache 中。如图 1 所示，查询执行线程将需要预读的 VTD-Record 数组和 Token 的相关信息写入预读工作集中，该工作集的定义如下。

定义 2(预读工作集)　$WorkSet=(DataItem, ItemNum)$，其中 *DataItem* 为存储预读项的数组，*ItemNum* 为 *WorkSet* 中存储的预读项个数，$DataItem[i]=(VRec, Status)$，其中 *VRec* 为 VTD-Record 数据项(如定义 1 所示)，*Status* 表示该 $DataItem[i]$是否已经被预读。

WorkSet 中的 *DataItem* 被定义为一维数组，并形成环状访问方式。预读线程轮流处理每个查询执行线程的 *WorkSet*，以保证每个查询执行线程均有较好的内存访问性能。预读线程在完成所有 *WorkSet* 的预取后，查询执行线程一般尚未处理完被预取的数据，如果预读线程开始新一轮的 *WorkSet* 预取，便可能将已经预读的尚未处理的数据替换出处理器 Cache，造成 Cache 访问缺失。针对此问题，本文采取的策略是由查询执行线程为预读线程赋予预读工作集，通过设置合理的线程参数减少上述问题的出现。下面是查询执行线程的线程函数描述，其中输入 *VRecArr* 表示该查询执行线程对应的工作集，*ItemNum* 为 *VRecArr* 中 VTD-Record 的个数。

算法 1　QueryExecThreadFun

```
输入：VRecArr, ItemNum
1. i=0;j=0; PreloadPos=0;
2. Status=UnPreload;
3. For i=0 to WorkSet. ItemNum-1
4.    WorkSet. DataItem[i]=VRecArr[i];
5.    WorkSet. DataItem[i]. Status=Status;
6. End For
7. PreloadPos=WorkSet. ItemNum;
8. While(True)
9.    Process VTD-Record from j * PreloadPos to (j+1) * PreloadPos-1
      of VRecArr
10.   j++;
11.   If(PreloadPos<=ItemNum)
12.      PreloadPos=PutDataToWS(PreloadPos, VRecArr);
13.   Else
14.      Break;
15.   End If
End While
```

如算法 1 所示，首先为 *WorkSet* 赋予数据，作为初始预读数据供预读线程提前将数据预读至 Cache(如算法 1 第 3—6 行所示)，然后查询执行线程开始执行查询处理。当已经预读的数据被查询执行线程处理完毕后，才调用 PutDataToWS 函数将新的需要预读的数据加入 *WorkSet* 中。预读线程的线程函数 PreloadThreadFun 如下。

算法 2　PreloadThreadFun

```
输入：VRecArr[QENum], WorkSet[QENum]
1. i=0;j=0;k=0;
2. Initiate PreloadPos[QENum] with 0;
3. While(True)
4.    For i=0 to QENum-1
5.    WorkSet=WorkSet[i];
6.    For j=PreloadPos[i] to WorkSet. ItemNnm-1;
7.       If(WorkSet. DataItem[j]. Status==UnPreload)
```

```
8.        Preload WorkSet. DataItem[j]' s corresponding data to
          Cache
9.        WorkSet. DataItem[j]. Status=Preloaded;
10.     End If
11.   End For
12. End While
```

预读线程轮流处理每个查询执行线程对应的预读工作集(如算法2第4—11行所示),从而确保所有的查询执行线程均可以提高内存访问性能。在一个 *WorkSet* 处理过程中首先处理 VTD-Record 数据(*DataItem. VRec*)自身的预读操作,*DataItem. VRec* 预读完成后,再根据 VTD-Record 所包含的 Token 位置信息(如定义1所述)将 XML 文档中的数据预读至处理器 Cache 中。

查询执行线程和预读线程在访问 *WorkSet. DataItem* 中的数据项前无需申请加锁,虽然可能会出现读取"脏数据"的情况(*DataItem* 中的 *Status* 状态不一致),但上述情况发生的机率很小,而且预读线程需要轮流访问各个 *WorkSet*,更降低了读取"脏数据"的可能性,实验3的结果证明了上述分析。

此外,还需要设置合理的系统参数。令 *QENum* 个 *WorkSet* 对应的预读数据量为 *TotalDataSize*:

$$TotalDataSize = QENum \cdot WorkSet.ItemNum \cdot \left(8+\left(\sum_{i=0}^{QENum-1}\frac{VRecArr^i.size}{VRecArr^i.num}\right)/QENum\right)$$

令 $TotalDataSize < C$,使得预读线程将所有 *WorkSet* 对应的数据预读至 Cache 也不会超过 Cache 容量,从而减少 Cache 访问缺失,其中 $VRecArr^i.size$,$VRecArr^i.size$ 分别表示对应的 XML 文档大小和其包含的 Token 的个数。此外,由于预读线程处理一个 *WorkSet* 的代价小于查询执行线程,因此,该框架只需要启动一个预读线程即可,同时为了提高查询执行的并发程度,启动 $N-1$ 个查询执行线程(即 $QENum=N-1$,N 为处理器核心个数),实验2的结果也证实了该设置的正确性。

4 实验结果与分析

4.1 实验设置

本文的研究和实验基于 Intel i5 3.1GHz 四核处理器展开,操作系统为 WinXP SP3,基于 VTD-XML[2] 实现 XML 处理引擎。实验数据集 $DataSet^i$($i=1\cdots3$)所采用的文档大小分别为 90M、60M 和 30M。实验执行 XML 节点查询,为了减少输出缓存对测试结果的影响,每个 XML 文档中满足查询条件的节点均相同。

4.2 实验内容及结果分析

本节测试了多线程 XML 节点查询执行框架的性能,以及参数取值对其性能的影响,以证明本文提出的各种优化策略的有效性。由于 VTD-XML 自身的性能优于其它几种 XML 文档处理模式,因此本文未将多线程 XML 节点查询执行框架与其它 XML 文档处理模式进行性能比较。

实验1 XML 节点查询性能测试。本实验测试多线程 XML 节点查询执行框架与未经优化的 XML 节点查询的性能。在四核处理器条件下,多线程 XML 节点查询执行框架启动3个查询执行线程和1个预读线程,未经优化的 XML 节点查询执行是指单线程执行。如图2所示,Origin 表示未经优化的 XML 节点查询执行性能,Multithreaded 表示多线程 XML 节点查询执行框架的性能。实验结果表明,Multithreaded 的性能要远远好于 Origin,其加速比平均达到了3.65左右,表明多线程 XML 节点查询执行框架通过多线程和线程 Cache 访问性能优化,使其能够充分地利用多核处理器的并行计算资源,极大地提高了 XML 节点查询执行的性能。

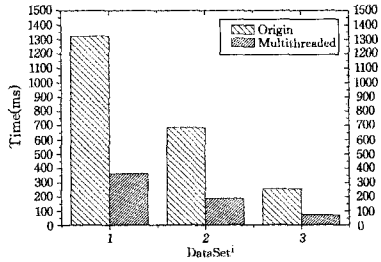

图2 XML 节点查询性能测试

实验2 不同线程模式设置性能测试。本实验测试各种不同线程设置方式对性能的影响。测试采用的线程模式如下:(a)3个查询执行线程和1个预读线程;(b)4个查询执行线程,1个预读线程;(c)4个查询执行线程;(d)2个查询执行线程和2个预读线程。如图3所示,虽然模式(b)启动的查询执行线程较模式(a)多,但是预读线程与查询执行线程共用一个处理器核心,预读线程的预读进度低于查询执行线程的处理进度,模式(c)启动的查询执行线程较模式(a)多,但是查询执行线程运行时 Cache 访问缺失较模式(a)严重,虽然其并行程度较高,但由于线程执行效率较低,性能低于模式(a);模式(d)启动两个预读线程,查询执行并行程度较模式(a)和模式(b)低,处理器利用效率不高,其性能最差。

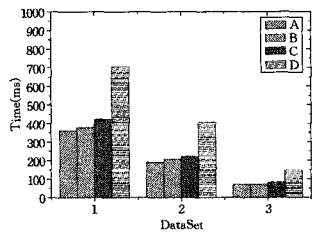

图3 不同线程模式性能测试

实验3 查询执行线程和预读执行线程访问 *WorkSet* 时加锁对性能的影响。该实验验证线程在访问 *WorkSet* 时是否加锁对性能的影响。实验分别采用两种模式:(a)访问 *WorkSet* 时申请锁;(b)访问 *WorkSet* 时不申请锁。如图4所示,由于模式(a)避免了频繁的锁操作,其性能略优于模式(b)。

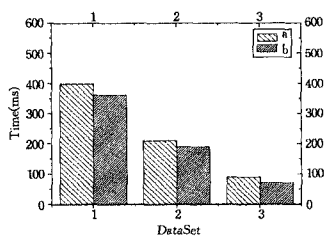

图4 锁机制对性能的影响

(下转第190页)

2004:190-202

[4] Gustaffsson F, Gunnarsson F. Mobile Positioning Using Wireless Networks[J]. IEEE Signal Processing Magazine, 2005, 22(4)

[5] Kusy B, Ledeczi A, Koutsoukos X. Tracking Mobile Nodes Using RF Doppler Shifts[C]//Sensys'07. 2007:29-42

[6] Mohanty S. VEPSD: A Novel Velocity Estimation Algorithm for Next-Generation Wireless Systems[J]. IEEE Trans. on Wireless Com., 2005, 4(6)

[7] Terzis A, Anandarajah A, More K, et al. Slip Surface Localization in Wireless Sensor Networks for Landslide Prediction[C]// IPSN'06. 2006:109-116

[8] Gaddi B, Bracha H, Tal A, et al. Continuous Close-Proximity RSSI-Based Tracking in Wireless Sensor Networks[C]//Proc. of 2010 Int'l Conf. on Body Sensor Networks. 2010:234-239

[9] Moore D, Leonard J, Rus D, et al. Robust distributed network localization with noisy range measurements[C]//Proc. SenSys'04. 2004

[10] Whitehouse K, Karlof C, Culler D. A Practical Evaluation of Radio Signal Strength for Ranging-based Localization[J]. SigMobile'07, 2007, 11(1)

[11] Wang Z J, Bulut E, Szymanski B K. Distributed Energy-Efficient Target Tracking with Binary Sensor Networks[J]. TOSN'10, 2010, 6(4)

[12] Li X R, Jilkov V P. A Survey of Maneuvering Target Tracking: Approximation Techniques for Nonlinear Filtering[C]//SPIE'04. 2004

[13] Zhang P, Martonosi M. LOCALE: "Collaborative Localization Estimation for Sparse Mobile Sensor Networks[C]//IPSN'08. 2008

[14] Kalman R E. A New Approach to Linear Filtering and Prediction Problems[J]. Trans. ASME, Journal of Basic Engineering, 1960, 82(1):35-45

[15] Ristic B, Arulampalam S, Gordon N. Beyond the Kalman Filter: Particle Filters for Tracking Applications[M]. Artech House, 2004

[16] Liu J, Reich J, Zhao F. Collaborative In-Network Processing for Target Tracking[J]. EURASIP Journal on Applied Signal Processing, 2003:378-391

[17] Ting J, Snoussi H, Richard C. Decentralized Variational Filtering for Target Tracking in Binary Sensor Networks[J]. IEEE Trans. on Mobile Computing, 2010, 9(10):1465-1477

[18] Yedavalli K, Krishnamachari B. Sequence-Based Localization in Wireless Sensor Networks[J]. IEEE Trans. on Mobile Computing, 2008, 7(1):81-94

[19] Zhong Z, He T. MSP: Multi-Sequence Positioning of Wireless Sensor Nodes[C]//Sensys'07. 2007

[20] Zhong Z, Zhu T, Wang D, et al. Tracking with Unreliable Node Sequences[C]//IEEE InfoCom'09. April 2009:1215-1223

[21] Zanca G, Zorzi F, Zanella A, et al. Experimental Comparison of RSSI-based Localization Algorithms for Indoor Wireless Sensor Networks[C]//RealWSN'08. 2008:1-5

[22] Wang W, Srinivasan V, Wang B, et al. Coverage for Target Localization in Wireless Sensor Networks[J]. IEEE Trans. on Wireless Comm., 2008, 7(2):667-676

[23] Xi W, He Y, Liu Y H, et al. Locating Sensors in the Wild: Pursuit of Ranging Quality[C]//Sensys'10. 2010:295-308

[24] Ren Q Q, Li J Z, Cheng S Y. Target Tracking under Uncertainty inWireless Sensor Networks[C]//Mobile Ad hoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on. 2011:430-439

[25] Ogilvy C S. Excursions in Geometry[M]. Dover, 1990

(上接第 181 页)

结束语 本文基于多核处理器提出了多线程 XML 节点查询执行框架，该框架基于 VTD-XML 开展研究工作。该框架通过并行执行提高节点查询执行的性能，同时通过预读策略将查询执行线程将要访问的数据提前取至处理器 Cache 中，提高了查询执行线程的性能。在实验中，基于 VTD-XML 开源代码实现了多线程 XML 节点查询执行框架和本文提出的各种性能优化措施。实验结果表明，本框架能够充分利用多核处理器的计算资源，大大提高了 XML 节点查询执行的性能。

参 考 文 献

[1] Zhao L, Bhuyan L. Performance Evaluation and Acceleration for XML Data Parsing[C]//Proc. 9th Workshop Computer Architecture Evaluation Using Commercial Workloads (CAECW 06). 2006

[2] VTD-XML: The Future of XML Processing[OL]. http://vtd-xml.sourceforge.net/

[3] Buneman P, Cong G, Fan W, et al. Using Partial Evaluation in Distributed Query Evaluation[C]//VLDB (2006). 2006:211-222

[4] Cong G, Fan W, Kementsisetsidis A. Distributed Query Evaluation with Performance Guarantees[C]//SIGMOD Conference 2007. 2007:509-520

[5] Suciu D. Distributed Query Evaluation on Semistructured Data [J]. ACM Trans. Database Syst., 2002, 27(1):1-62

[6] Lu W, Gannon D. Parallel XML Processing by Work Stealing [C]//SOCP' 07:Proceedings of the 2007 workshop on Service-oriented computing performance: aspects, issues, and approaches 2007. 2007:31-38

[7] Bordawekar R, Lim L, Shmueli O. Parallelization of XPath Queries using Multi-core Processor[C]//Challenges and Experiences. EDBT 2009. 2009

[8] Zhou Jing-ren, Cieslewicz J, Ross K A, et al. Improving Database Performance on Simultaneous Multithreading Processors[C]// Proceeding of the 31nd International Conference on Very Large Databases. VLDB Endowment, 2005:49-60

[9] Hardavellas N, Pandis I, Johnson R. Database servers on chip multiprocessors limitations and opportunities[C]//3rd Biennial Conference on Innovative Data Systems Research (CIDR). 2007

[10] He Bing-sheng, Luo Qiong. Cache-oblivious Database: Limitations and Opportunities[J]. ACM Transactions on Database Systems, 2008, 33(2):1-42

[11] Hennessy J L, Patterson D A. Computer Architecture (4th edition)[M]. Morgan Kaufman, 2007