一种有效的并行频繁子图挖掘算法

陈晓云　赵娟　陈鹏飞　邢乔金　刘国华

（兰州大学信息科学与工程学院　　兰州　730000）

摘要：在分析并行频繁项集挖掘算法的基础上，提出了一种有效的并行频繁子图挖掘算法，该并行算法采用主从节点处理模式，　　将主节点处理后生成的频繁子树集放到从节点上并行的生成频繁子图。通过实验验证了算法的可行性和有效性。

关键字：并行化；频繁项集；频繁子图；

Abstract：Based on the analysis of the parallel frequent itemset mining algorithm, a kind of effective parallel algorithm for mining frequent subgraph is introduced in this paper. The parallel algorithm adopts a master-slave node processing mode, put the frequent subtree sets which generated by the master node on the slave node which parallel generate frequent subgraph.The results of the experiments show that our algorithm has good effectiveness and feasibility.

Key words: parallel; frequent itemset; frequent subgraph

1.　引言：

频繁子图挖掘与其他比较成熟的频繁模式挖掘相比，　　图结构数据所包含的信息比一般的数据类型的数据量更大　,其数据结构比线性表和树更为复杂。在图形结构中，结点之间的关系是任意的，　任意两个数据元素之间都可能相关。　　尽管其结构很复杂，　但是由于基于图的应用越来越广泛，其已经渗入到诸如语言学、逻辑学、物理、化学、计算机科学及数学的这些分支学科中。 如通过对已有的生物分子结构与未知的生物结构的研究，　　来确定未知生物分子与已知生物分子之间的联系与区别。例如在　　PTE(predictive toxicology evaluation challenge)　项目中找到频繁出现的而且与已有的有毒物质具有相同子结构的物质，　　这样就可以发现新的对人体有害的物质。因此，对基于图的频繁子图挖掘的算法研究是非常必要的，而且具有很好的实际应用价值。

在通常情况下，由于没有任何先验知识做参考，频繁子图的挖掘工作量会很大。对于海量数据的挖掘任务来讲，　现有的频繁子图挖掘算法速度比较慢，　　而且效率不高，　因此没有得到广泛的应用。 所以研究出一个算法效率高，　　执行速度快的频繁子图挖掘算法是目前一个非常热门的话题。

本文在分析以往的并行频繁项集挖掘算法的基础上，　　提出了一种并行频繁子图发掘算法 PG-Miner 。该挖掘算法采用主从模式，将整个过程分为两部分，第一部分由主处理节点来生成频繁子树集，　然后将生成的子树集分发到其他从处理节点。　　第二部分将频繁子图边扩展及同构判断这部分频繁子图挖掘算法中时间复杂度最高的部分并行处理。　　文章在最后通过实验验证了本算法的有效性和可行性。

2.　并行频繁项集挖掘综述

频繁模式挖掘的搜索空间可以被模拟成类似格的结构，　　其中由模式的大小来决定它处于格中的哪一层，　每一层又以某种顺序进行排列。　模式格的维数决定了问题的指数级别。　　例如，对于一个有着　n 个不同项的事务数据库，可能的模式就会有　　2^n。也就是说，如果一个事务数据库有　100 个不同的项，搜索空间就达到　　$2^{100} \approx 10^{30}$。巨大的搜索空间使得在大型数据库上的频繁模式的挖掘成为一个计算密集型问题。　　然而传统的频繁模式挖掘算法被单一处理器和有限的内存空间所限制，　　不适用于大型数据库。　因此，利用高性能并行计算来改善现有频繁模式挖掘算法的瓶颈，使其适用于大规模数据库是非常必要的。

在 FP-Growth 算法的基础上，　2001 年 Osmar R. Zaiane 等人提出了并行频繁项集挖掘算法 MLFPT ，　2004 年 Javed 和 Khokhar 等人提出了并行频繁项集挖掘算法　　PFP-tree。

2.1 MLFPT 算法

Za?ane.O.R 等人于 2001 年提出基于共享式内存的类　　FP-Growth 的并行频繁项集挖掘算

法 MLFPT 。此算法对 FP-Growth 算法中的 FP-Tree 进行了并行化的改进。在整个执行过程中仅需扫描数据库两次，建立了一个特殊的数据结构叫做频繁模式树（ Frequent Pattern Tree），之后在上面挖掘频繁项集。

由于一颗在并行节点上共享的树结构势必需要在叶子或节点甚至路径上设置锁机制， 这样就会导致严重的瓶颈。于是， MLFPT 算法中采取了将 FP-Tree 分成块到每个节点上，而保持结果树是各节点上共享的来避免假负现象。 建立这样的树大大减少了生成频繁项集的开销。这些树上的频繁项都是交叉连接起来的（与 FP-Tree 中相同），并且总体上连接在一个全局头表上。

每块树森林（ tree forest）被分配到各个处理节点上，分开后的 FP-Tree 在挖掘过程中被自下向上的顺序快速遍历。 树的位置降低了并行节点间由于错误的操作而覆盖其他节点更新的可能性，同时使得死锁的可能性最小化。

但是基于共享式内存的并行频繁项集不能够适用于广泛使用的集群系统， 因此局限性比较大。

2.2 PFP-tree 算法

Javed和 Khokhar 等人于 2004 年提出了分布式内存环境下的类 FP-Growth 并行频繁项集挖掘算法 PFP-Tree。算法中将整个数据库分割成不重合的 p 块，其中 p 是处理节点的数量。然后将各个数据块分配给相应的处理节点。 PFP-tree 算法的具体步骤如下：

1 各节点 p 扫描被分配的数据块并且计算本地数据库块中的频繁项的支持度。

2 所有处理节点做同步得到总体的频繁项支持度计数。

3 各节点依据总的支持度来排序频繁项，并删除非频繁项。

4 各节点第二次扫描本地数据库块并建立本地 FP-tree。

5 头表被分成 p 个互不相交的子集， 每个处理节点为被分配到的项的集合并行挖掘频繁模式。

6 由于第 5 步中的划分是静态的， 每个处理节点必须通过其他的节点来确认本地树上的信息。在第 4 步被分配到一个节点上的单频繁前缀分支构成了挖掘步的完整信息。 利用一次自底向上的扫描来进行确认信息。

7 第 6 步中确认的信息利用一个递归的归并树结构在各节点上将需要进行 log p 次通讯。在每一次通讯的最后，一个节点需要解包收到的信息到本地的 FP-tree 树上，然后为下一次归并准备新的信息。

8 每个处理节点挖掘被分配的频繁项集。

由于 PFP-tree 需要各节点交换基于每个频繁 1-项集的条件模式基来得到本地所需的数据分块，所以整个算法的通讯还是比较多的，降低了并行的效率。

3. 并行频繁子图挖掘算法 PG-Miner

为了提高并行效率并且减少算法的通讯量， 我们提出了一个频繁子图挖掘算法， 该挖掘算法采用主从模式， 将整个过程分为两部分， 第一部分由主处理节点来生成频繁子树集， 然后将生成的子树集分发到其他从处理节点。 第二部分将频繁子图边扩展及同构判断这部分频繁子图挖掘算法中时间复杂度最高的部分并行处理。

3.1 主节点频繁子树挖掘算法 FTGen

FTGen(frequent subtree generation)算法第一步判断要扩展的频繁子树是否为最小 DFS 编码，来避免对重复生成的子树做进一步扩展。 第二步通过对频繁边集中的每条边进行扩展判断。 第三步将已提取的边从边集中去掉， 减小需要扩展的边集， 第四步判断当前提取的边是否为树边 e_e，如果是，将其加入到 DFS 编码最小的频繁子树 t 中。第五步判断在频繁子树 T 中是否存在 t 的同构子树。如果不存在，将 t 加入到结果集 T 中。第六步对扩展后的 t 执行 FTGen 算法以得到全部频繁子树。 FTGen 的挖掘结果是图集 GS 中出现的频繁子树，

储存在集合 T 中。频繁子树将被用于进一步对边扩展以形成图，同时，频繁子树作为图的一种，是最终挖掘结果的子集。算法伪代码如 Algorithm 3-1 所示：

```
Algorithm3-1：FTGen
Input：        Graph_Set GS, max_Sup_DFS t
               Edge_Set E, Frequent_tree T
Output：       Frequent_tree T
1)if(tl=minDFS(t))
2)    return t is not the minimum DFS edge;
3)foreach e_e of edge_set E
  {
4)   E=E- e_e;
5)   if(istreeedge(t,e_e))
6)        insert(t,e_e)
7)        if(isomorphism(T,t)){
8)             insert(T,t)
9)              FTGen(GS,t,E,t)
  }
```

3.2 从节点子图拓展挖掘算法 PFGGen

算法 PFGGen 在 FTGen 算法的基础上做进一步挖掘，主要是从频繁子树向频繁子图的扩展过程。第一步由于 FTGen 本身生成的频繁子树也是最终结果的一部分，故需要将其也加入到结果集中。第二步将 T 中的频繁子树按降序 DFS 编码排序，并且对 T 中每颗子树做扩展成图的处理。第三步先将当前需要扩展的子树初始化到图 g，然后对频繁边集中的每条边 e_e 做扩展处理，将已提取要扩展的边从边集中去掉。第四步判断当前边是否可扩展，如果当前边可以扩展，扩展这条边。第五步判断扩展后的图是否满足最小 DFS 编码的要求，避免重复操作。 第六步判断扩展的图在结果集中是否存在同构子图， 如果不存在， 将扩展的子图加入到结果集。算法伪代码如 Algorithm 3-2 所示：

```
Algorithm3-2：PFTGen
Input：        Graph_Set GS, minimum_suport minsup, frequent_subtree T
Output：       frequent_subgraph FSG
1)initresult(FSG,T)
2)sort(T);//order by DFS
3)foreach t of T
  {
4)   initgraph(g,t);
5)foreach e_p of edge_set E
  {
6)   subtract(E,e_p);
7)   if(isextention(g,e_p)){
8)        insertgraph(g,e_p);
9)        if(g!=min(g))
              break;
10)       if(isomorphism(FSG,g))
11)           insertresult(FSG,g);
      }
   }
```

3.3 PG-Miner 算法

PG-Miner 算法第一步初始化 MPI 并行环境并且计算用于此次并行计算的所有节点。第二步判断是否为主节点，扫描图集并找到图集 GS 中所有频繁边，并删除图集中所有非频繁边，将所有频繁边集按 DFS 编码和支持度降序排列，初始化扩展子树 t。第三步调用 FTGen 算法，获得频繁子树集。第四步调用 PFGGen 算法，扩展频繁子树到频繁子图。第五步结束 MPI。PG-Miner 算法伪代码如 Algorithm 3-3 所示：

```
Algorithm3-3：PG-Miner
Input：          Graph_Set DS, minimum_suport minsup
Output：         frequent_subgraph_set FGS
{
1)MPI_Init(&argc,&argv);
2)MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
3)MPI_Comm_rank(MPI_COMM_WORLD,&myid);
4)MPI_File_open(MPI_COMM_WORLD,"poutput",MPI_MODE_RDWR,MPI_INFO_NULL,&fh);
5)   If(myid==0)//mainnode
     {
6)      scan(GS,minsup,E);
7)      decsort(E);
8)      init(t,T);
9)      FTGen(GS,t,E,T);
10)     foreach t of T
11)       MPI_Send(message(t),length(t),MPI_INT,depth(t)%numprocs+1,99,MPI_COMM_WORLD);
     }
    else
     {
12)     T=NULL;
13)      MPI_Recv(message,maxlength,MPI_INT,0,99,MPI_COMM_WORLD,&status);
14)      insert(T,t);
15)      PFGGen(GS,minsup,T);
16)      foreach g of FGS
17)        MPI_File_write_shared(fh,buffer(g),length(g),MPI_CHAR,status);
     }
18)MPI_Barrier(MPI_COMM_WORLD);
19)MPI_Finalize();
return;
}
```

4. 实验结果与分析

实验环境：由 5 台 2.8 GHz Intel Core CPU ， 2GB 内存的 PC 机搭建 MPI 环境，机子之间的通信采用千兆路由器实现，操作系统为内核 2.6.31-14 的 Ubuntu Karmic Koala (Ubuntu 9.10)。所有程序均用 GCC3.4.3 和 MPI 库 mpich2-1.0.7 编译实现。

在实验中，我们分别选取真实数据和仿真数据来与 gSpan, FFSM 作比较，gSpan 和 FFSM 都是在单台机子运行，我们的算法启动 4 个从节点来计算。数据可从以下网址获得：http://dtp.nci.nih.gov/docs/aids/aids_data.html 。图 4-1，4-2，4-3 给出了在不同数据集上 gSpan, FFSM， PG-Miner 的运行结果

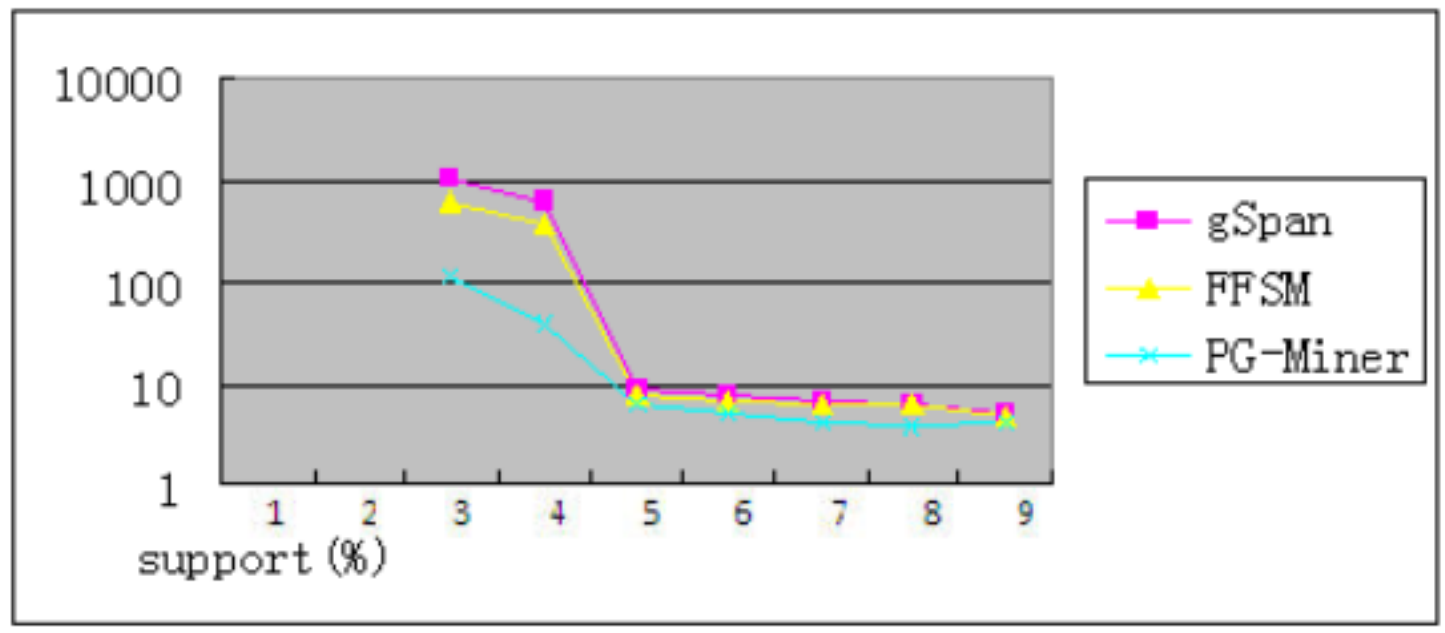

图 4-1 CA 数据集

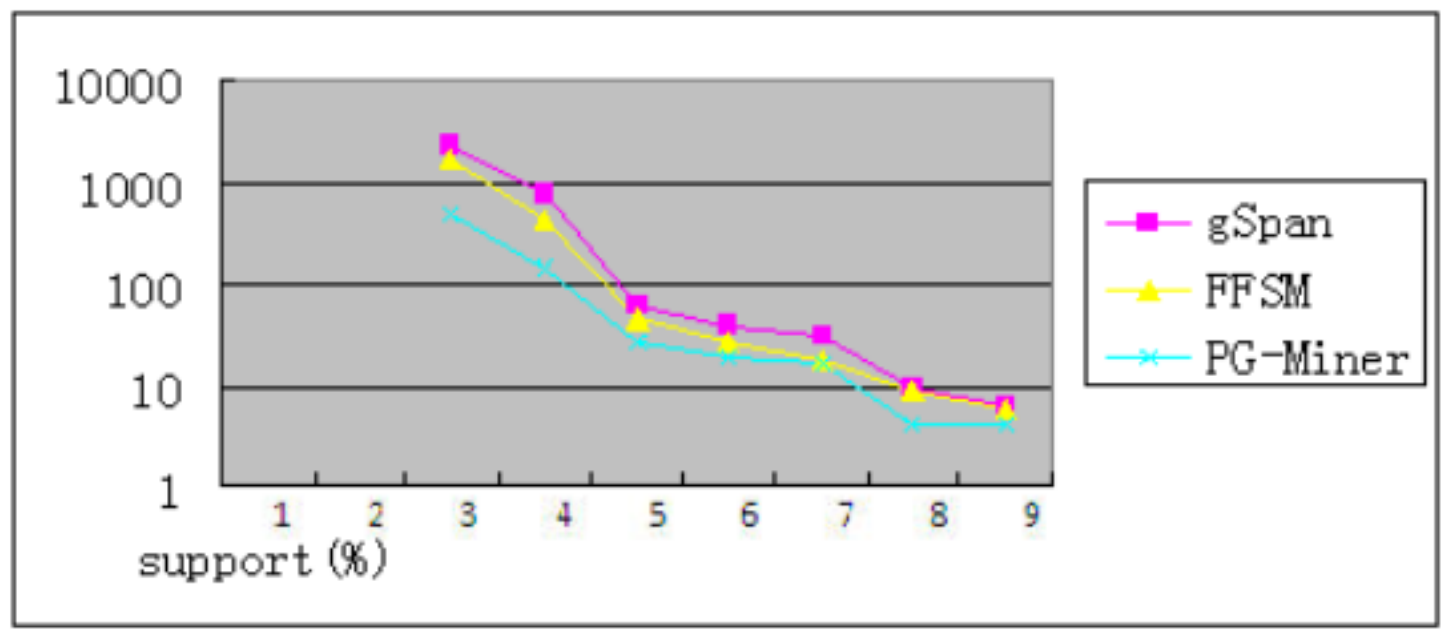

图 4-2 CM 数据集

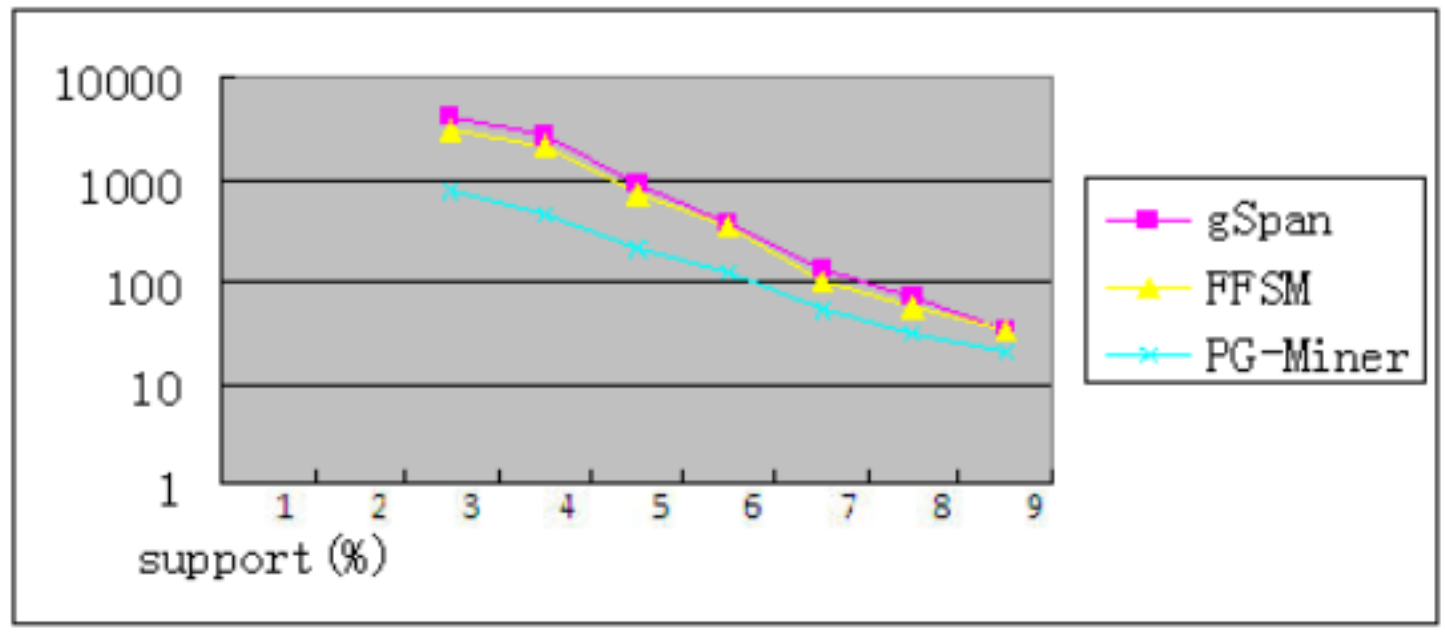

图 4-3 CI 数据集

仿真数据我们采用文献 [16]Kuramochi 和 Karypis 提供的工具产生的数据集。我们选取了两个参数集来测试我们的结果，第一个设置 N=10(N 代表可能的节点标识数)， T=40(T 代表每个图的平均边数)， D=10000(D 代表生成图的数量)， I=7(I 代表可能生成的频繁子图的平均大小)；另一个数据集设置 T=40， N=5， D=10000， I=7。图 4-4， 4-5 在不同数据集上 gSpan， FFSM， PG-Miner 的运行结果

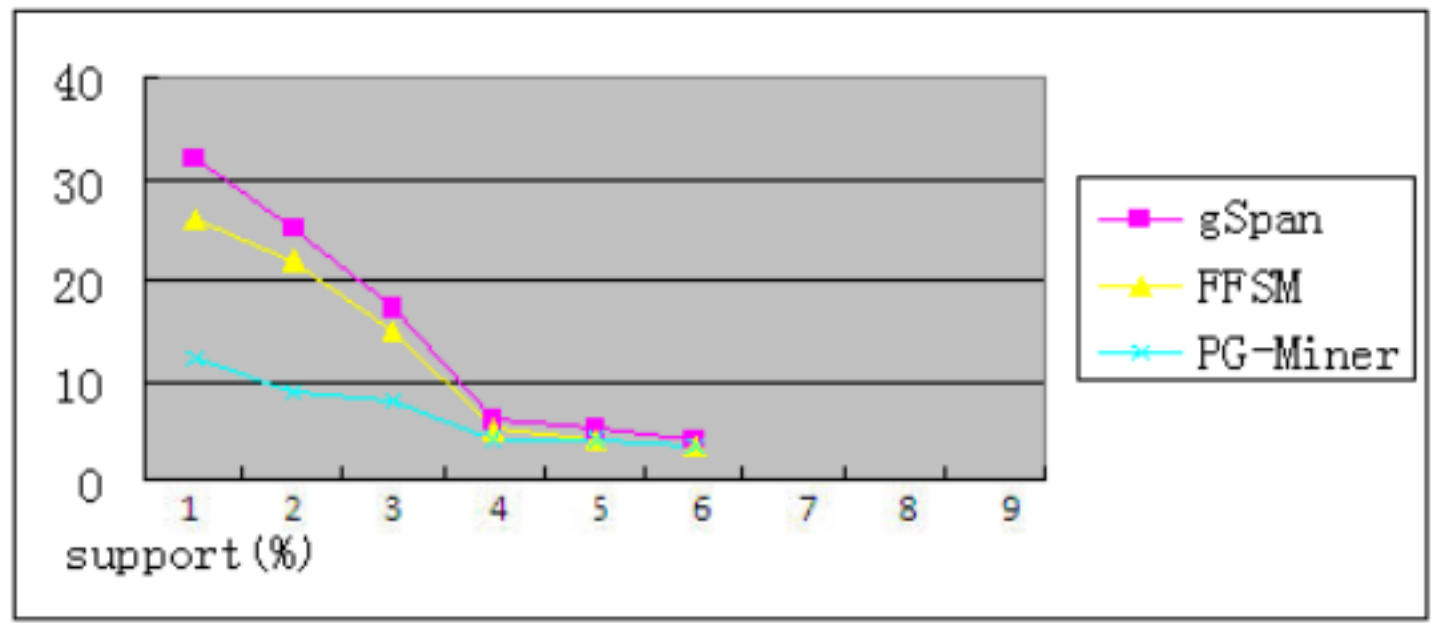

图 4-4 N10I7T40 数据集

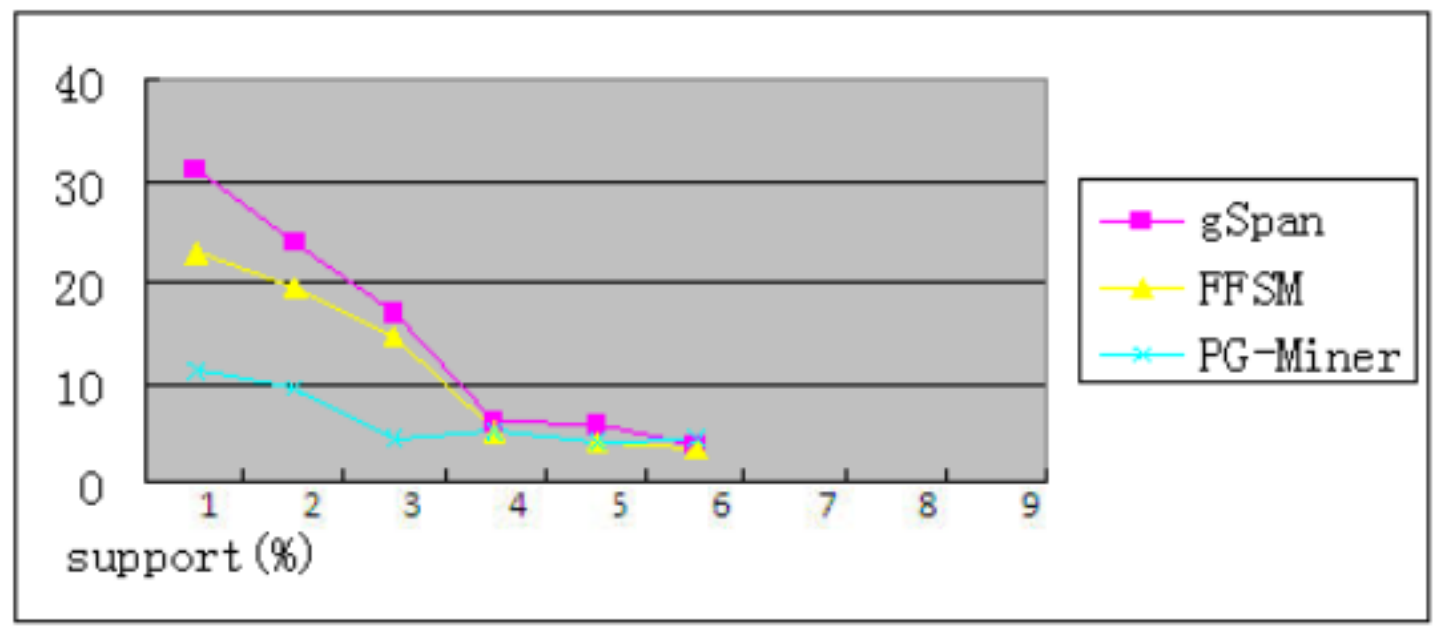

图 4-5 N5I7T40 数据集

在下面，我们将 PG-Miner 在从节点个数为 2 和从节点为 4 时在 CA 上的运行结果也展示出来，用来分析我们的算法在处理节点扩展时的可扩展性，图 4-6 给出了运行结果。

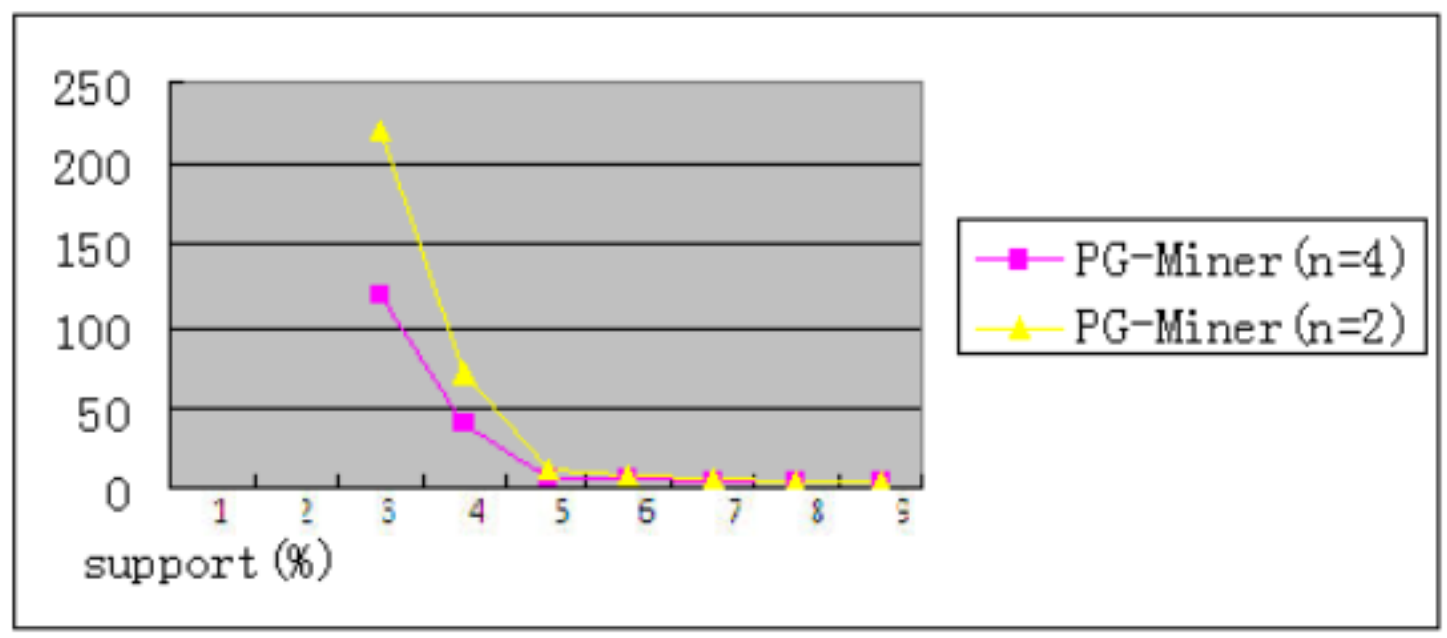

图 4-6 CA 数据集

图 4-1 到 4-3 为了能更加清晰的列出结果，我们采用了对数值。在这几个真实数据集上，我们比较了从支持度 3%到 9%，在图上我们可以看到，在支持度很小时，PG-Miner 算法的运行时间仅是 FFSM 的 1/4 多一点，这说明 PG-Miner 运行效率提高了接近 n(处理节点数)倍。但当支持度增加时，PG-Miner 算法的运行时间在 4s 左右，甚至出现高于 FFSM 的情况，这说明此时，MPI 环境的启动开销和通信开销已经成为程序运行的主要时间，在这种情况下，并不适合 PG-Miner 算法。

图 4-4 和 4-5 是在模拟数据集上得到的运行结果，我们比较了支持度从 1%到 6%，由结果可以得到与真实结果集上相同的结论。

图 4-6 在 CA 数据集上处理节点分别为 2 和 4 时，我们比较了支持度从 1%到 6%的运行

结果，从运行结果可以看出，　　PG-Miner 具有很好的可扩展性。

5. 结论

本文在研究以往并行频繁项集挖掘算法的基础，提出了并行频繁子图挖掘算法 PG-Miner 。该算法成功的将频繁子图挖掘算法中时间复杂度最高的子图同构判断过程分发到多台处理器上并行处理，　　使得算法的执行时间随处理节点的线性增加而线性减少。　　并且在不同的真实和模拟数据集上验证了算法的可行性。

另外算法在并行化方面，　　如何提高并行粒度，　　使得各处理节点负载更加均衡以及如何减少子图同构的判断是我们下一步的研究方向。

文献：

[1] 范明，孟晓峰 . 数据挖掘概念与技术第　　2 版

[2] 王丹阳，田卫东，胡学刚 . 一种有效的并行频繁项集挖掘算法 . 计算机应用研究 . 1001-3695(2008)11-3332-03

[3] 王永恒，杨树强，贾　　焰 . 海量文本数据库中的高效并行频繁项集挖掘方法　　. 计算机工程与科学 . 1007-130X(2007)09-0110-04

[4] 张大为，黄丹，嵇敏，谢福鼎　　. 利用模式指导树的并行频繁项集挖据方法　　. 计算机工程与应用 . 1002-8331（2010）22-0147-04

[5] Fiedler, M. and Borgelt, C.: 2007, Support computation for mining frequent subgraphs in a single graph, in P. Frasconi, K. Kersting and K. Tsuda (eds), MLG.

[6] X. Yan and J. Han, “gSpan: Graph-Based Substructure Pattern Mining,” Proc. 2002 Int’l Conf. Data Mining (ICDM’02), pp. 721-724, Dec. 2002.

[7] Wu, J. and Chen, L.: Mining Frequent Subgraph by Incidence Matrix Normalization. Journal of Computers 3(10)(2008) 109–115.

[8] C. Wang and S. Parthasarathy. Parallel algorithms for mining frequent structural motifs in scientific data. In Proceedings of the ACM International Conference on Supercomputing (ICS), 2004.

[9] T. Meinl, I. Fischer, and M. Philippsen. Parallel mining for frequent fragments on a shared-memory multiprocessor -results and java-obstacles-. In M. Bauer, A. Kroner, and B. Brandherm, editors, LWA 2005 - Beitrage zur GI-Workshopwoche Lernen, Wissensentdeckung, Adaptivitat , pages 196–201, 2005.

[10] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In Proceedings of the Internation Conference on Data Mining (ICDM) , pages 313–320, 2001.

[11] 王艳辉，吴斌，王柏　　. 频繁子图挖掘算法综述　　. 计算机科学 . 2005 Vol.32 No.10.

[12] 胡作霆 ,董兰芳 ,王洵 . 图的数据挖掘算法研究　　[J] . 计算机工程 . 2006 ,32 (3) :76278.

[13] A. Javed and A. A. Khokhar. Frequent pattern mining on message passing multiprocessor systems. Distributed and Parallel Databases, 16(3):321-334, 2004.

[14] 李继腾，骆志刚，丁凡，田文颖。赵琦　　. 最大频繁子图挖掘算法研究　　. 计算机工程与科学.1007-130X . 2009.12.020

[15] 唐德权，朱林立　. 频繁子图挖掘算法研究　　. 计算机工程 . 1000-3428(2009)09-0052-03

[16] Zaiane O.R, EI-Hajj M, Lu P. Fast Parallel Association Rule Mining Without Candiadate Generatioin. Technical Report TR01-02. University of Alberta, Canada: Department of Computing Science. 2001: 665-668P.

[17]R. Agrawal, J. C. Shafer. Parallel Mining of Association Rules. IEEE Trans. Knowledge and Data Eng., 1996, 8(6): 962-969.