

Parallel Mining Frequent Patterns: A Sampling-based Approach

CS 498: Program Optimization
Fall 2006
(slides from Shengnan Cong)

University of Illinois at Urbana-Champaign



Talk Outline

- ◆ Background
 - Frequent pattern mining
 - Serial algorithm
- ◆ Parallel frequent pattern mining
 - Parallel framework
 - Load balancing problem
- ◆ Experimental results
- ◆ Optimization
- ◆ Summary



Frequent Pattern Analysis

- ◆ Frequent pattern
 - A pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
- ◆ Motivation
 - To find the inherent regularities in data
- ◆ Applications
 - Basket data analysis
 - What products were often purchased together?
 - DNA sequence analysis
 - What kinds of DNA are sensitive to this new drug?
 - Web log analysis
 - Can we automatically classify web documents?



3

Frequent Itemset Mining

- ◆ Itemset
 - A collection of one or more items
 - Example: {Milk, Juice}
 - k-itemset
 - An itemset that contains k items
- ◆ Transaction
 - An itemset
 - A dataset is a collection of transactions
- ◆ Support
 - Number of transactions containing an itemset
 - Example: $Support(\{Milk, Juice\}) = 2$
- ◆ Frequent-itemset mining
 - To output *all* itemsets whose support values are no less than a predefined threshold in a dataset

Transaction	Items
T1	Milk, bread, cookies, juice
T2	Milk, juice
T3	Milk, eggs
T4	Bread, cookies, coffee

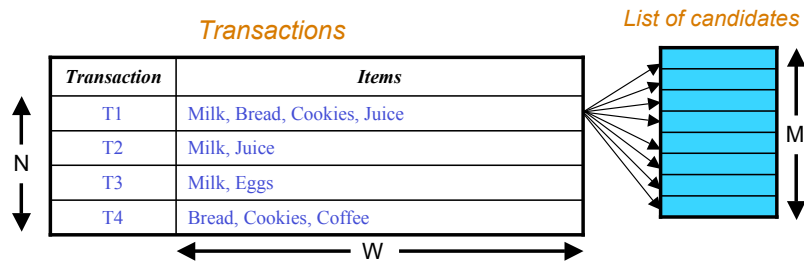
```
Support threshold = 2
{milk}:3
{bread}:2
{cookies}:2
{juice}:2
{milk, juice}:2
{bread, cookies}:2
```



4

Frequent Itemset Mining

- ◆ Frequent itemset mining is computationally expensive
- ◆ Brute-force approach:
 - Given d items, there are 2^d possible **candidate** itemsets
 - Count the support of each candidate by scanning the database
 - Match each transaction against every candidate



- Complexity $\sim O(NMW) \Rightarrow$ Expensive since $M = 2^d$

5

Mining Frequent-itemset In Serial

- ◆ FP-growth algorithm [Han et al. @SIGMOD 2000]
 - One of the most efficient serial algorithms to mine frequent-itemset. [FIMI'03]
 - A divide-and-conquer algorithm.
- ◆ Mining process of FP-growth
 - Step 1:
 - Identify frequent 1-items with one scan of the dataset.
 - Step 2:
 - Construct a tree structure (FP-tree) for the dataset with another dataset scan.
 - Step 3:
 - Traverse the FP-tree and construct a projection (sub-tree) for each frequent 1-item. Recursively mine the projections.



6

Example of FP-growth Algorithm

◆ Example

<i>TID</i>	<i>Items</i>
100	{f, a, c, d, g, i, m, p}
200	{a, b, c, f, l, m, o}
300	{b, f, h, j, o}
400	{b, c, k, s, p}
500	{a, f, c, e, l, p, m, n}

Support Threshold = 3

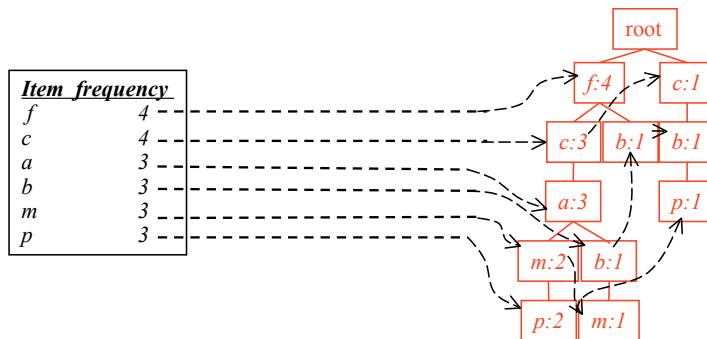
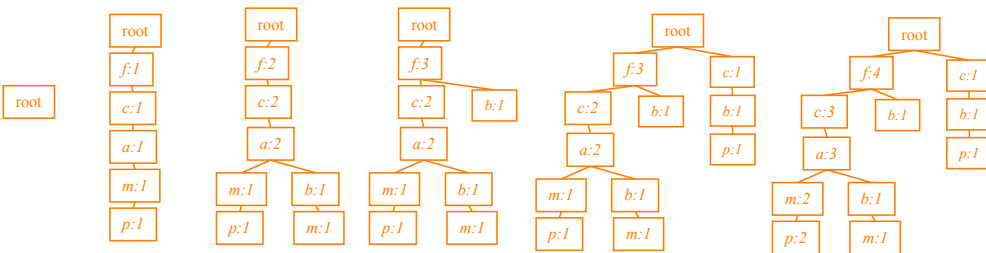
Step 1:

<i>Item</i>	<i>frequency</i>
f	4
c	4
a	3
b	3
m	3
p	3



Step 2:

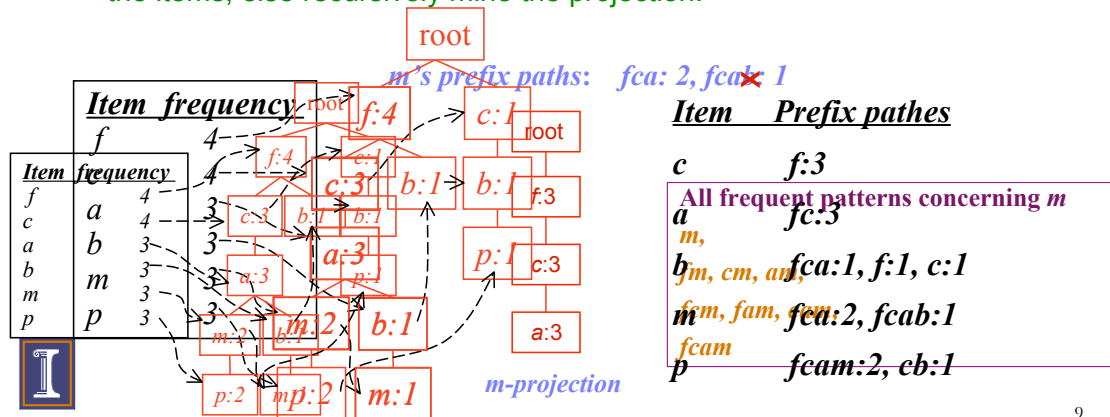
<i>TID</i>	<i>Items bought</i>	<i>Ordered Frequent Items</i>
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}



Example of FP-growth Algorithm (cont'd)

Step 3:

- Traverse the FP-tree by following the side link of each frequent 1-item and accumulate the prefix paths.
- Build FP-tree structure (projection) for the accumulated prefix paths
- If the projection contains only one path, enumerate all the combinations of the items, else recursively mine the projection.



9

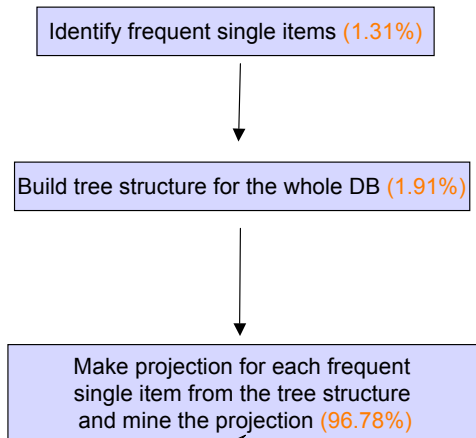
Talk Outline

- ◆ Background
 - Frequent-itemset mining
 - Serial algorithm
- ◆ Parallel frequent pattern mining
 - Parallel framework
 - Load balancing problem
- ◆ Experimental results
- ◆ Optimization
- ◆ Summary

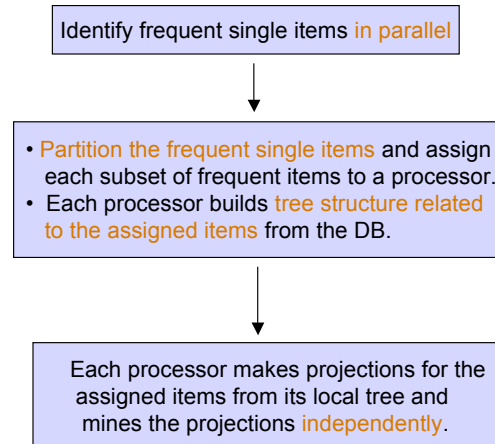


Parallel Mining Frequent Itemset

- ◆ FP-growth algorithm:

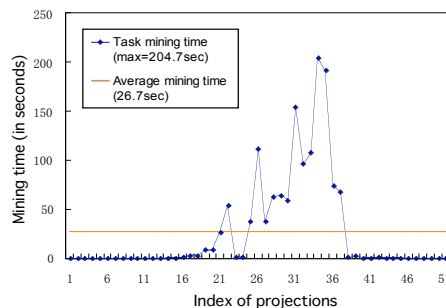
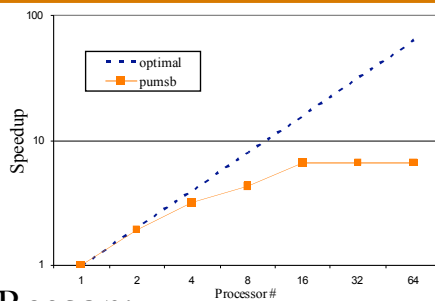


- ◆ Parallelization framework for FP-growth algorithm



Divide and conquer

Load Balancing Problem



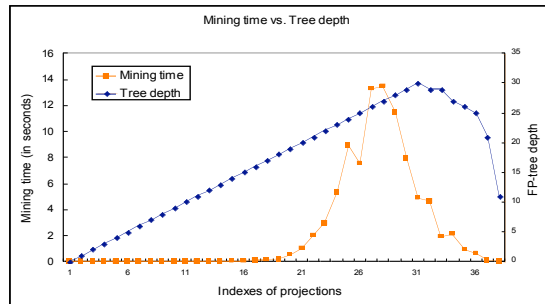
- ◆ Reason:
 - The large projections takes too long to mine relative to the mining time of the overall dataset.
- ◆ Solution:
 - The larger projections must be partitioned.
- ◆ Challenge:
 - Identify the larger projections.



How To Identify The Larger Projections?

◆ Static estimation

- Based on dataset parameters
 - Number of items, number of transactions, length of transactions, ...
- Based on the characteristics of the projection
 - Depth, bushiness, tree size, number of leaves, fan-out, fan-in, ...



Result --- No correlation found with any of the above.



13

Dynamic Estimation

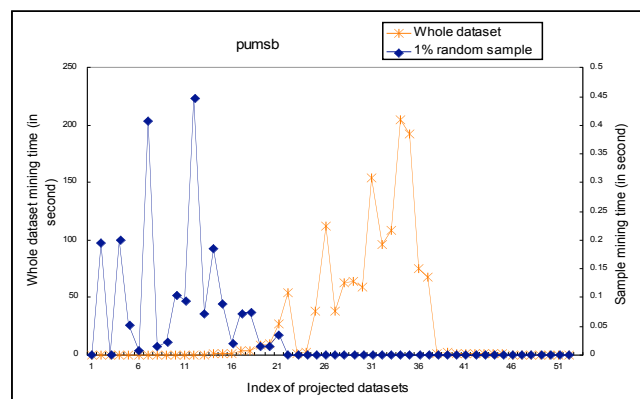
◆ Runtime sampling

- Use the relative mining time of a sample to estimate the relative mining time of the whole dataset.
- Accuracy vs. overhead

◆ Random sampling: random select a subset of records.

- Not accurate

e.g. Pumsb
1% random sampling
(overhead 1.03%)



14

Selective Sampling

- ◆ Sampling based on frequency.

- Discard the infrequent items
- Discard a fraction t of the most frequent 1-items.

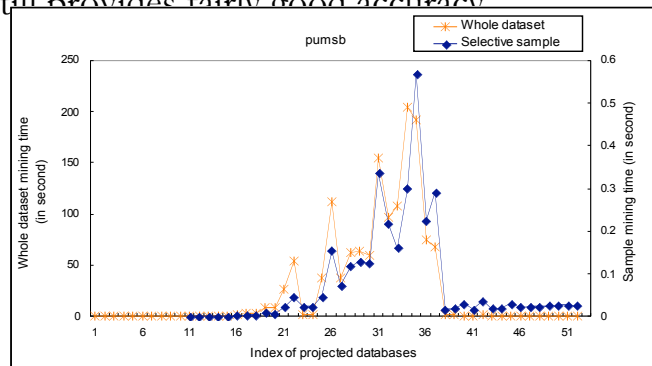
e.g. *Frequent-1* items: $\langle (f:4), (c:4), (a:3), (b:3), (m:3), (p:3), (l:2), (s:2), (n:2), (q:2) \rangle$,
 $(Sup_{min}=2)$,

$t=20\%$

~~$\{f, a, c, d, g, i, m, p\}$~~
 $\{f, a, b, g, i, p\}$

- ◆ When filtering top 20%, sampling takes average 1.41% of the sequential mining time still provides fairly good accuracy

e.g. Pumsb Top 20%
 (overhead 0.71%)



Why Selective Sampling Works?

- ◆ The mining time is proportional to the number of frequent itemsets in the result. (from experiments)
- ◆ Given a frequent L -itemset, all its subsets are frequent itemsets. There are $2^L - 1$ subsets.
- ◆ Removing one item at the root reduces the total number of itemsets in the result and, therefore, reduces the mining time roughly by half.
- ◆ The most frequent items are close to the root. The mining time of their projections are negligible but their presence increases the number of itemsets in the results.



Talk Outline

- ◆ Background
 - Frequent-itemset mining
 - Serial algorithm
- ◆ Parallel frequent pattern mining
 - Parallel framework
 - Load balancing problem
- ◆ Experimental results
- ◆ Optimization
- ◆ Summary



17

Experimental Setups

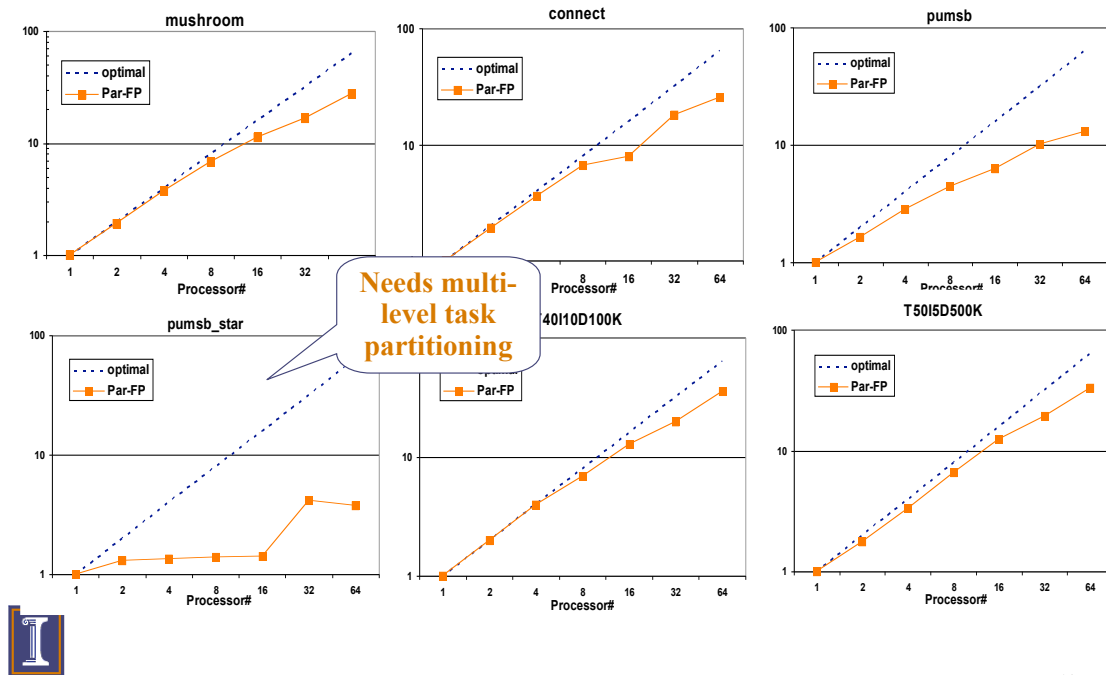
- ◆ A Linux cluster with 64 nodes.
 - 1GHz Pentium III processor and 1GB memory per node.
- ◆ Implementation: C++ and MPI.
- ◆ Datasets:

Dataset	#Transactions	#Items	Max Trans. Length
mushroom	8,124	23	23
connect	57,557	43	43
pumsb	49,046	7,116	74
pumsb_star	49,046	7,116	63
T40I10D100K	100,000	999	77
T50I5D500K	500,000	5,000	94



18

Speedups: Frequent-itemset Mining



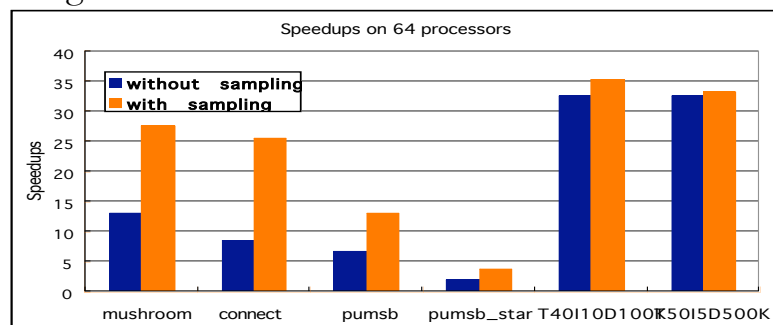
19

Experimental Results For Selective Sampling

- ◆ Overhead of selective sampling (average 1.4%)

Dataset	Mushroom	Connect	Pumsb	Pumsb_star	T40I10D100K	T50I5D500K
Overhead	0.71%	1.80%	0.71%	2.90%	2.05%	0.28%

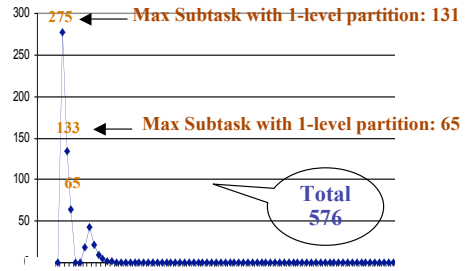
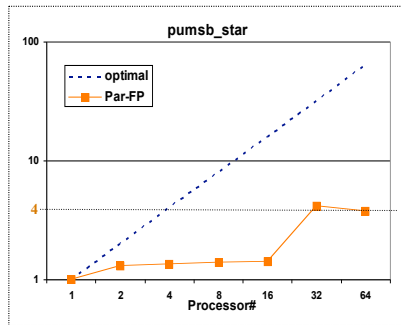
- ◆ Effectiveness of selective sampling
 - Selective sampling can improve the performance by 37% on average.



20

Optimization: Multi-level Partitioning

- ◆ Problem analysis:



Optimal speedup with 1-level partitioning: $576/131=4.4$

- ◆ Conclusion:

- We need multi-level task partitioning to obtain better speedup.

- ◆ Challenges:

- How many levels are necessary?
- Which sub-subtasks to be further partitioned?

