Acceleration of FP-Tree construction based on MapReduce

Yu-Chen Meng*[1], Chen-Lun Yang†[2], and Shao-ting Hsu‡[3]

[1]Institute of Computer Science and Engineering, National Yang Ming Chiao Tung University, Hsinchu City 300, Taiwan

1 Introduction/Motivation

The motivation behind this project is to parallelize the process of frequent itemset and association rule mining while reducing communication costs during parallelization. Currently, the classical algorithms for mining frequent itemsets are FP-Growth and Apriori. These algorithms first identify frequent itemsets and then derive association rules. Through comparison, FP-Growth, which is based on depth-first search, is more suitable for parallelization than Apriori, which uses breadth-first search, due to the following reasons: (1) High Decomposability: FP-Growth's conditional FP-tree can be divided into independent subproblems, which can be processed in parallel without frequent communication or synchronization. This makes FP-Growth more feasible for large-scale parallel computation. (2) Fewer I/O Operations: FP-Growth requires only two scans of the dataset, while Apriori requires multiple scans to generate candidate sets. As a result, FP-Growth significantly reduces I/O overhead, making it more efficient in utilizing parallel processing resources.

Although there are existing studies that use MapReduce to efficiently process large-scale datasets, each node runs on different machines, which necessitates frequent data transmission and serialization. This introduces high costs when exchanging data between conditional FP-trees. We aim to reduce communication costs through alternative approaches while effectively handling smaller datasets.

2 Statement of the problem

In this project, I chose to apply the FP-Growth algorithm to the field of market analysis and parallelize its execution. The goal is to analyze the impact of the combination of two factors—variety of store products and diversity of customer purchases—on the performance of association rule mining in parallel processing. Additionally, I will test how different parallelization methods affect performance growth across these four combinations.

3 Proposed approaches

We will focus on how to parallelize the two components of the FP-Growth Algorithm: Data Compression and Tree Construction. Our goal is to utilize the concept of MapReduce for parallelization, aiming to accelerate the time required to construct the FP tree from large transaction datasets.

- **Data Compression**: Scan the dataset once to determine the frequency of individual items and create a list of frequent itemsets.
  - **Map function**: In the Map phase, each Mapper processes a portion of the dataset. The input transactions are read, and the frequency of each item is counted. Then, intermediate key-value pairs are generated where the key is the item and the value is its frequency. This allows for a preliminary count of item frequencies across different partitions of the data.

*jett.ii12@nycu.edu.tw
†chenlun.yang.cs12@nycu.edu.tw
‡hst0411.cs12@nycu.edu.tw

– **Reduce function**: In the Reduce phase, the Reducers receive the key-value pairs emitted by the Mappers, grouped by the item (key). The Reducer aggregates the counts for each item by summing up the frequency values. The output of this phase is a list of items and their corresponding frequencies.

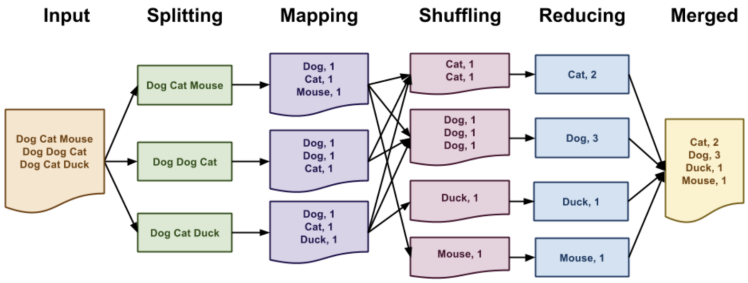

Figure 1: Similar to the word count example in the concept of MapReduce

- **Tree Construction**: In the second pass through the dataset, we sort the data and filter out any items that do not meet the minimum support threshold, reducing the amount of data. The sorted data is then distributed across different nodes. Each node creates a sub-FP-Tree by inserting transactions into the tree, which are then merged into a complete FP-Tree.

  – **Map function**: Based on the results of the Reduce phase 1, filter out the frequent items for all items and sort them in descending order based on their frequency. Traverse each transaction and insert the filtered and sorted items into a local FP-Tree, which contains only the transactions from the data block processed by the Mapper. Each Mapper generates a local FP-Tree (subtree).

  – **Reduce function**: Receive localFP-Trees from multiple Mappers. Merge these local FP-Trees to generate a global FP-Tree. During the merging process, it is essential to ensure that paths with the same prefix are compressed into the same node, and their frequencies are accumulated. The output is the traversal of all data and the constructed FP-Tree.

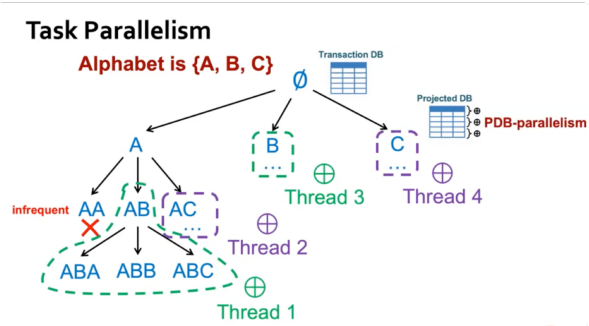

Figure 2: Build the FP-Tree

We will also analyze and compare the results obtained by applying different parallelization methods using the MapReduce concept.

4 Language selection

Although MapReduce is effective at processing large-scale datasets, it incurs high performance costs when handling fine-grained synchronization, dynamic data updates, and computation tasks that require frequent interactions. Therefore, FP-Growth selects the following 5 approaches:

1. **pthread**
   While MapReduce excels at processing large distributed datasets, its architecture presents performance bottlenecks in fine-grained tasks like building conditional FP-trees. On a single machine, using pthread can fully leverage the power of multi-core CPUs, particularly in handling shared memory within the processor. Multi-threaded computations operate in shared memory, reducing communication delays compared to MapReduce, which involves communication between distributed nodes.

2. **OpenMP**
   OpenMP allows developers to implement parallel processing in existing sequential code using compiler directives without needing to write extensive synchronization or thread management code. This is particularly appealing for recursive and compute-intensive algorithms like FP-Growth. OpenMP's support for dynamic load balancing automatically adjusts workloads across threads, providing strong support for the recursive structure of FP-trees and the uneven computation distribution during the construction of conditional FP-trees.

3. **MPI**
   While MapReduce handles distributed data, it imposes excessive costs for fine-grained parallel computations. MPI is specifically designed for efficient distributed computing, giving developers fine-grained control over communications and offering lower communication latency. This is especially useful for FP-Growth variants that require extensive data exchange across multiple machines, enabling better performance in distributed environments.

4. **CUDA**
   CUDA provides the ability to perform large-scale parallel computations on GPUs, significantly improving the performance of FP-Growth when working with very large datasets or tasks that demand parallel processing. Specifically, during the construction and handling of FP-trees, operations such as node updates and frequent conditional FP-tree operations can be parallelized and accelerated using CUDA.

5. **OpenCL**
   OpenCL is a cross-platform heterogeneous computing framework that can operate across various hardware architectures (CPU, GPU, FPGA). This means that the parallelization of the FP-Growth algorithm can be deployed in different hardware environments, enhancing flexibility and portability. In heterogeneous computing environments, OpenCL enables FP-Growth to utilize all available hardware resources (CPUs, GPUs, and even FPGAs) to accelerate computations.

5 Related work

5.1 Associating Rule

Association eules reflect the interdependence and relationships between one thing and others. They are an important technique in data mining, used to extract valuable relationships among data items from large datasets. The goals include two aspects: 1. Discovering frequent itemsets 2. Discovering association rules. Apriori and FP-Growth algorithm are common methods for learning association rules.

- **Apriori Algorithm** [1]

  The Apriori Algorithm operates on the principle of finding frequent itemsets in a transactional database. These are sets of items that appear together in transactions more frequently than a specified minimum support threshold. The algorithm follows a level-wise search approach, iteratively generating candidate itemsets and pruning those that do not meet the support threshold. After identifying the frequent itemsets, the algorithm generates association rules that describe relationships between the items.
  While the Apriori Algorithm is effective for finding association rules, its requirement for multiple scans of the dataset can be a significant drawback in terms of computational efficiency and scalability.

- **FP-Growth Algorithm** [2]

  Unlike the Apriori Algorithm, which relies on multiple passes over the dataset, the FP-Growth algorithm adopts a more efficient approach that allows it to find frequent itemsets without candidate generation, requiring only two scans of the data. The FP-Growth Algorithm can be broken down into three steps:

1. **Data Compression**: The FP-Growth algorithm begins by scanning the dataset once to determine the frequency of individual items and create a list of frequent items. It then constructs a compact data structure called the FP-tree (Frequent Pattern tree). This tree represents the dataset in a way that preserves the itemset association while reducing the overall data size.
2. **Tree Construction**: The FP-tree is built by inserting transactions into the tree. Each path in the tree represents a transaction, and shared prefixes among transactions are combined to save space. The tree structure allows for efficient counting of item frequencies.
3. **Frequent Pattern Mining**: Once the FP-tree is constructed, the algorithm recursively mines for frequent itemsets by exploring the tree. It does this by focusing on individual items and their conditional patterns, which are subsets of the FP-tree that relate to a specific item.

5.2 Parallel Algorithm

- **MapReduce Algorithm** [3]

  MapReduce is a programming model and processing framework designed for efficiently processing large datasets across distributed computing environments. Developed by Google, this algorithm simplifies the processing of vast amounts of data by breaking down tasks into smaller, manageable components that can be executed in parallel. It consists of two main functions: Map and Reduce.

  - **Map Function**: The process begins with the Map function, which takes a set of input key-value pairs and processes them to generate a set of intermediate key-value pairs. Each Mapper works independently and can run in parallel on different chunks of data. During this phase, the algorithm performs operations such as filtering, sorting, and transforming the input data.
  - **Reduce Function**: The Reduce function takes the intermediate key-value pairs produced by the Map function and aggregates them to produce the final output. Each Reducer processes a specific key and its associated values, combining them to form a summary result.

6 Statement of expected results

We hope that our parallelized method can be faster than the serial version and expect to improve the efficiency.

Method	Speedup
Serial	1
Our Method	> 5

Table 1: Expected improvements of MapReduce

7 Timetable

We will follow the timeline below to implement and test our method.

Week	Date	Description
1	11/04~11/08	Implement the FP-growth algorithm
2	11/11~11/15	Collect datasets for generating the FP-tree
3	11/18~11/22	Implement the MapReduce algorithm using pthreads to accelerate FP-growth
4	11/25~11/29	Compare the time spent on different types of datasets
5	12/02~12/06	Compare the advantages and disadvantages of different types of datasets
6	12/09~12/13	Finish report

Table 2: Timetable

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499. Citeseer, 1994.

[2] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM sigmod record*, volume 29, pages 1–12. ACM, 2000.

[3] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Communications of the ACM*, volume 51, pages 107–113. ACM, 2008.