2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress

A Data Science Model for Big Data Analytics of Frequent Patterns

Carson K. Leung*, Fan Jiang, Hao Zhang, and Adam G.M. Pazdor
Department of Computer Science
University of Manitoba
Winnipeg, MB, Canada
**Email: kleung@cs.umanitoba.ca*

***Abstract*—Frequent pattern mining is an important data mining task. Since its introduction, it has drawn attention from many researchers. Consequently, many frequent pattern mining algorithms have been proposed, which include level-wise Apriori-based algorithms, tree-based algorithms, and hyperlinked array structure based algorithms. While these algorithms are popular and benefit from a few advantages, they also suffer from some disadvantages. In the current era of big data, a wide variety of high volumes of valuable data of different veracities can be easily collected and generated at a high velocity. These big data lead to additional challenges for frequent pattern mining. In this paper, we present a data science model for big data analytics of frequent patterns with MapReduce. We evaluated our model by using social networks, which are good examples of big data. Evaluation results show the efficiency and practicality of our data science model in mining and analyzing big data for the discovery of interesting frequent patterns from various real-life applications including social network analysis.**

Keywords—big data; data analytics; data science; frequent patterns; MapReduce

I. INTRODUCTION

As a popular data mining task, *frequent pattern mining* discovers implicit, previously unknown, and potentially useful knowledge—in the form of patterns revealing frequently co-occurring items, events, or objects (e.g., frequently purchased merchandise items in shopper market basket, frequently collocated events)—from data. Since the introduction of the research problem of frequent pattern mining [2], numerous frequent pattern mining algorithms [7], [21], [28] have been proposed. Among them, the Apriori algorithm [2] is a notable one, but it requires many database scans. The FP-growth algorithm [8] improves algorithmic efficiency, but at a price of requiring memory space to keep the global FP-tree (capturing the content of the original database) and all subsequent sub-trees (each capturing the contents of subsequent projected databases). Both the TD-FP-Growth algorithm [26] and the H-mine algorithm [24] avoid building and simultaneously keeping multiple FP-trees during the mining process. However, during the mining process, TD-FP-Growth keeps updating the global FP-tree by adjusting tree pointers, whereas the H-mine algorithm keeps updating pointers/hyperlinks in the corresponding H-struct. As alternatives to the aforementioned "horizontal" frequent pattern mining algorithms (which use a transaction-centric approach to find which pattern is supported by or contained in a transaction), both the Eclat algorithm [27] and the VIPER algorithm [25] mine frequent patterns "vertically" by using an item-centric approach to count the number of transactions supporting or containing the patterns.

To speed up the mining process for frequent patterns in the aforementioned *serial* algorithms, *parallel and distributed* frequent pattern mining algorithms [27] have also been proposed. Examples include the Count Distribution, algorithm [3], which is a parallel version of the aforementioned Apriori algorithm.

Although serial, parallel or distributed frequent pattern mining algorithms are commonly applied to shopper market basket data for analyzing consumer behaviours, they can also be applicable to wide varieties of high-volume valuable data which can be of different veracities (e.g., precise data, imprecise or uncertain data). Nowadays, these *big data* can be easily collected or generated at a high velocity from different data sources (e.g., social networks).

To mine frequent patterns from big data, several algorithms have been proposed [5]. For instance, the Single Pass Counting (SPC), Fixed Passes Combined-counting (FPC), and Dynamic Passes Combined-counting (DPC) algorithms [19] were developed—based on the Apriori and the Count Distribution algorithms—to mine frequent patterns from big precise data using the MapReduce model. Parallel FP-growth (PFP) algorithm [18], which also uses the MapReduce model, parallelizes the tree-based FP-growth algorithm for mining frequent patterns from precise big data. The MR-growth algorithm [20] uses the MapReduce model to mine uncertain big data for frequent patterns.

In this paper, we present an efficient *data science model* for big data analytics of frequent patterns. To evaluate the effectiveness of our data science model, we apply our model to the shopper market basket data. In addition, as social networks (e.g., Facebook, Google+, LinkedIn, Twitter, Weibo) are good examples of big data, we also use our model for *social network analysis*. Our *key contributions* of this paper include (i) our data science model for big data analytics of frequent patterns and (ii) its application to social network analysis.

The remainder of this paper is organized as follows. The next section provides some background materials and reviews some related works. Section III presents our data science model for big data analytics of frequent patterns. Section IV shows an application of our data science model to social network analysis. Evaluation results on both frequent pattern mining and social network analysis are shown in Section V. Finally, conclusions are given in Section VI.

978-1-5090-4065-0/16 $31.00 © 2016 IEEE
DOI 10.1109/DASC-PICom-DataCom-CyberSciTec.2016.148

II. Background and Related Works

In this section, we discuss background and related works on (A) traditional frequent pattern mining algorithms (including serial, parallel and distributed algorithms), (B) the MapReduce model for handling big data, and (C) MapReduce-based frequent pattern mining algorithms),

A. Serial, Parallel and Distributed Algorithms for Frequent Pattern Mining

The Apriori algorithm [2] is a notable frequent pattern mining algorithm, which applies a generate-and-test paradigm in mining frequent patterns in a level-wise bottom-up fashion: It first generates candidate patterns of cardinality k (i.e., candidate k-itemsets) and tests if each of them is frequent (i.e., tests if its support or frequency meets or exceeds the user-specified *minsup* threshold). Based on these frequent patterns of cardinality k (i.e., frequent k-itemsets), Apriori then generates candidate patterns of cardinality k+1 (i.e., candidate (k+1)-itemsets). This process is applied repeatedly to discover frequent patterns of all cardinalities. Hence, it requires k database scans to find a frequent pattern of cardinality k (i.e., k-itemset).

The FP-growth algorithm [8] improves efficiency by using a Frequent Pattern tree (FP-tree) to capture the content of the transaction database. Unlike Apriori (which scans the database K times, where K is the maximum cardinality of the discovered frequent patterns), FP-growth scans the database twice. To mine frequent patterns, FP-growth recursively extracts relevant paths from the FP-tree to form projected databases (i.e., collections of transactions containing some items), from which sub-trees (i.e., smaller FP-trees) capturing the content of relevant transactions are built. While FP-growth avoids the generate-and-test paradigm of Apriori (because FP-growth uses the divide-and-conquer paradigm), FP-growth requires lots of memory space to build many smaller FP-trees (e.g., for $\{a\}$-projected database, $\{a, b\}$-projected database, $\{a, b, c\}$-projected database, ..., where a, b and c are some domain items) during the mining process.

TD-FP-Growth [26] avoids building and keeping multiple FP-trees at the same time during the mining process. Unlike FP-growth (which mines frequent patterns by traversing the global FP-tree and sub-trees in a bottom-up fashion), the TD-FP-Growth algorithm traverses only the global FP-tree and in a top-down fashion. During the mining process, instead of recursively building sub-trees, TD-FP-Growth keeps updating the global FP-tree by adjusting tree pointers.

Similarly, the H-mine algorithm [24] also avoids building and keeping multiple FP-trees at the same time during the mining process. It does so by using a hyperlinked-array structure called H-struct, which captures the content of the transaction database. During the mining process, the H-mine algorithm also recursively updates many pointers/hyperlinks in the H-struct.

Instead of mining frequent patterns "horizontally" (i.e., using a transaction-centric approach to find which pattern is supported by or contained in a transaction), the Eclat algorithm [27] mines frequent patterns "vertically" (i.e., using an item-centric approach to count the number of transactions supporting or containing the patterns), in a level-wise, bottom-up, fashion. The algorithm treats the database as a collection of item lists. Each list for an item x keeps IDs of transactions containing x. The length of the list for x gives the support of a pattern $\{x\}$ of cardinality 1. If its support is not less than the user-specified minimum support (*minsup*) threshold, $\{x\}$ is considered frequent. By taking the intersection of the lists for two frequent patterns α and β, Eclat returns the IDs of transactions containing ($\alpha \cup \beta$). Again, the length of the resulting (intersected) list gives the support of the pattern ($\alpha \cup \beta$). Eclat works well when the database is sparse. However, when the database is dense, these item lists can be long.

Alternatively, the VIPER algorithm [25] represents the item lists in the form of bit vectors. Each bit in a vector for a domain item x indicates the presence (bit "1") or absence (bit "0") of a transaction containing x. The number of "1" bits for x gives the support of a pattern $\{x\}$ of cardinality 1. If its support is not less than the user-specified *minsup* threshold, $\{x\}$ is considered frequent. By computing the dot product of the vectors for two frequent patterns α and β, VIPER returns the vector indicating the presence of transactions containing ($\alpha \cup \beta$). Again, the number of "1" bits of this vector gives the support of the resulting pattern ($\alpha \cup \beta$). VIPER works well when the database is dense. However, when the database is sparse, lots of space may be wasted because the vector contains lots of 0s.

Instead of mining frequent patterns in serial, there are also parallel and distributed frequent pattern mining algorithms [27]. For example, the Count Distribution algorithm [3] is a parallelization of the Apriori algorithm. It divides transactional databases of precise data and assigns them to parallel processors. Each processor counts the frequency of patterns assigned to it and exchanges this frequency information with other processors. This counting and information exchange process is repeated for each pass/database scan.

As we are moving into the new era of big data, more efficient frequent pattern mining algorithms are needed. The reason is that wide varieties of valuable data—which can be of different veracities (e.g., precise data, imprecise or uncertain data)—can be easily collected or generated at a high velocity with volumes beyond the ability of aforementioned frequent pattern mining algorithms for data analytics [12] , [14] within a tolerable elapsed time. To handle big data, researchers proposed the use of the MapReduce programming model.

B. The MapReduce Model

High volumes of valuable data (e.g., web logs, texts, documents, business transactions, banking records, financial charts, medical images, surveillance videos, as well as streams of marketing, telecommunication, biological, life science, and social media data) can be easily collected or generated from different sources, in different formats, and at a high velocity in many real-life applications in modern organizations and society. This leads us into the new era of

big data [14], [22], which refer to high-veracity, high-velocity, high-value, and/or high-variety data with volumes beyond the ability of commonly-used software to capture, manage, and process within a tolerable elapsed time. This drives and motivates research and practices in *data science* [1]—which aims to develop systematic or quantitative processes to analyze and mine big data—for continuous or iterative exploration, investigation, and understanding of past business performance so as to gain new insight and drive science or business planning. By applying big data analytics and mining (which incorporates various techniques from a broad range of fields such as cloud computing, data analytics, data mining, machine learning, mathematics, and statistics), data scientists can extract implicit, previously unknown, and potentially useful information from big data (e.g., big social network data).

Over the past few years, researchers have used a high-level programming model—called *MapReduce* [6]—to process high volumes of big data by using parallel and distributed computing on large clusters or grids of nodes (i.e., commodity machines) or clouds, which consist of a master node and multiple worker nodes. As implied by its name, MapReduce involves two key functions: (i) the map function and (ii) the reduce function. Specifically, the input data are read, divided into several partitions (sub-problems), and assigned to different processors. Each processor executes the map function on each partition (sub-problem). The map function takes a pair of $\langle key_1, value_1 \rangle$ and returns a list of $\langle key_2, value_2 \rangle$-pairs as an intermediate result:

$$\text{map: } \langle key_1, value_1 \rangle \mapsto \langle key_2, value_2 \rangle,$$

where (i) key_1 and key_2 are keys in the same or different domains and (ii) $value_1$ and $value_2$ are the corresponding values in some domains. Afterwards, these pairs are shuffled and sorted. Each processor then executes the reduce function on (i) a single key_2 from this intermediate result $\langle key_2$, list of $value_2 \rangle$ together with (ii) the list of all values that appear with this key in the intermediate result. The reduce function "reduces"—by combining, aggregating, summarizing, filtering, or transforming—the list of values associated with a given key_2 (for all k keys) and returns a single (aggregated or summarized) $value_3$:

$$\text{reduce: } \langle key_2, \text{list of } value_2 \rangle \mapsto value_3,$$

where (i) key_2 is a key in some domains and (ii) $value_2$ and $value_3$ are the corresponding values in some domains.

An advantage of using the MapReduce model is that users only need to focus on (and specify) these "map" and "reduce" functions—without worrying about implementation details for (i) partitioning the input data, (ii) scheduling and executing the program across multiple machines, (iii) handling machine failures, or (iv) managing inter-machine communication. Examples of MapReduce applications include the construction of an inverted index and the word counting of a document for data processing [6], [29].

C. MapReduce-Based Data Analytic Algorithms for Frequent Pattern Mining

Recall from Section II.A that (i) the Apriori, FP-growth, TD-FP-Growth, and H-mine algorithms "horizontally" mine frequent patterns in serial, whereas (ii) both the Eclat and VIPER algorithms "vertically" mine frequent patterns in serial. In contrast, the Count Distribution algorithm mines frequent patterns in parallel. With the MapReduce model described in Section II.B, frequent patterns can be mined from big data. For instance, the Single Pass Counting (SPC), Fixed Passes Combined-counting (FPC), and Dynamic Passes Combined-counting (DPC) algorithms [19] were designed based on both the Apriori and the Count Distribution algorithms for mining frequent patterns from big data of precise data with the MapReduce model. Specifically, SPC first divides the databases into partitions. The worker node corresponding to each partition then executes the *map function* by finding all candidate 1-itemsets from the big data. Afterwards, the *reduce function* computes the support of these candidate 1-itemsets, and outputs frequent 1-itemsets. In each subsequent pass $k \geq 2$, SPC executes similar map and reduce functions in each pass k to generate candidate k-itemsets and count their support for finding frequent k-itemsets. As variants of SPC, both FPC and DPC apply the pass bundling technique to reduce the number of passes/database scans when generating candidate itemsets (for Passes $k \geq 3$). For instance, FPC statistically bundles a fixed number of passes (e.g., three passes) to generate all candidate k-, (k+1)-, and (k+2)-itemsets from frequent (k–1)-itemsets. In contrast, DPC dynamically bundles several passes (depending on the number of generated candidates in these bundled passes).

The Parallel FP-growth (PFP) algorithm [18] uses MapReduce to parallelize the tree-based FP-growth algorithm for mining frequent patterns from big data. Similarly, the BigFIM algorithm [23] extends both the Apriori and Eclat algorithms with MapReduce. To speed up the mining process, the Smart Cache algorithm [9] applies a cache layer to capture the intermediate results between the map and reduce functions during the mining process.

III. Our Data Science Model

In this section, we present our data science model for big data analytics of frequent patterns, with an illustrative example for a frequent pattern mining problem.

A. Big Data Analytics of Frequent Patterns

Given N transactions in a big data set, our data science model performs the following actions. The model first applies a map_1 function to each transaction t_i indexed by its unique transaction ID (shorthanded as tid) as follows:

$$\text{map}_1\text{: } \langle \text{tid } i, \text{transaction } t_i \rangle \mapsto \langle \text{tid } i, \text{item } x \in t_i \rangle,$$

in which the master node reads and divides big data in partitions. Specifically, this map_1 function can be specified as follows:

for each transaction t_i identified by tid i **do**
  for each item $x \in t_i$ **do**
    emit $\langle$tid i, item $x\rangle$

Note that this map_1 function is applied to each transaction, and results in a list of $\langle$tid i, item $x \in t_i\rangle$-pairs. Each pair captures the ID (i.e., i) of a transaction t_i and an item x (represented by its item ID) in t_i. Afterwards, our data science model applies a $reduce_1$ function to group and count the number of items. More specifically, $\langle$tid i, item $x \in t_i\rangle$-pairs from the map_1 function are shuffled and sorted. Each processor then executes the $reduce_1$ function on the shuffled and sorted pairs to count the number of items. To speed up this big data analytic process, our data science model allows users to express the interestingness of patterns via the specification of the minimum support *minsup* threshold. By incorporating this user preference, our data science model returns a list of $\langle$item x, tids containing x, support of $x\rangle$-triplets. In other words, our data science model applies the following $reduce_1$ function to convert the big data set from a horizontal representation to a vertical representation:

$reduce_1$: list of $\langle i, x \in t_i \rangle \mapsto \langle \{x\}, \text{tidlist}(\{x\}), \sup(\{x\})\rangle$

with a detailed definition as follows:

for each item x emitted by map_1 **do**
  set $\sup(\{x\}) = 0$
  set $\text{tidlist}(\{x\}) = \{\}$
  for each $\langle$tid i, $x\rangle$ emitted by map_1 **do**
    $\sup(\{x\}) = \sup(\{x\}) + 1$
    $\text{tidlist}(\{x\}) = \text{tidlist}(\{x\}) \cup \{i\}$
  if $\sup(\{x\}) \geq$ user-specified *minsup* threshold
  then emit $\langle x, \text{tidlist}(\{x\}), \sup(\{x\})\rangle$

Hence, the $reduce_1$ function returns in a list of frequent items (i.e., frequent singletons), together with its transaction list (that lists all the transactions containing the frequent items) and the number of transactions in this list (i.e., occurrence count or support of every frequent item). To summarize, after applying the first set of map_1 and $reduce_1$ functions, our data science model finds individually frequent items.

Thereafter, our data science model uses the second set of map_2 and $reduce_2$ functions to mine interesting patterns in the form of pairs of frequently co-occurring pairs of items (i.e., frequent 2-itemsets) based on the results from the first set of map_1 and $reduce_1$ functions. Moreover, our data science model makes good use of the Apriori property that "any supersets of an infrequent itemsets are guaranteed to be infrequent" to speed up the big data analytics and mining process. For instance, knowing that items D and E are infrequent, it is guaranteed that any pairs containing item D or E are also infrequent. By making the best use of this Apriori property, we effectively prune the search space for mining interesting patterns. Specifically, the map_2 function, which returns a $\langle$tid i, co-occurring item pair $\{x_1, x_2\} \in t_i\rangle$-pair for every transaction t_i, can be specified as follows:

map_2: $\langle$item x_1, $\text{tidlist}(\{x_1\})\rangle \mapsto \langle$tid i, $\{x_1, x_2\} \in t_i \rangle$,

which could be defined in details as follows:

for each $t_i \in \text{tidlist}(\{x_1\})$ emitted by $reduce_1$ **do**
  for each $x_2 \in t_i$ **do**
    emit $\langle$tid i, $\{x_1, x_2\} \in t_i\rangle$

To avoid getting $x_1 = x_2$, we impose additional checking condition to ensure that $x_1 \neq x_2$. Consequently, the resulting map_2 function could become the following:

for each $t_i \in \text{tidlist}(\{x_1\})$ emitted by $reduce_1$ **do**
  for each $x_2 \in t_i$ **do**
    if $x_1 \neq x_2$
    then emit $\langle$tid i, $\{x_1, x_2\} \in t_i\rangle$

Moreover, to avoid double counting of $\{x_1, x_2\}$ with $\{x_2, x_1\}$, our data science model imposes a practical implication condition that $x_1 < x_2$. This condition not only avoids $x_1 = x_2$ (i.e., ensure that $x_1 \neq x_2$) but also avoids double counting of $\{x_1, x_2\}$. The reason is that, if $x_1 = x_2$, then the map_2 function would not return $\{x_1, x_1\} = \{x_1\}$ because it is a singleton set (i.e., with duplicate item). If $x_1 > x_2$, then the map_2 function would also not return $\{x_1, x_2\}$ because it is identical to the $\{x_2, x_1\}$ that has already been returned by the map_2 function. Hence, the final map_2 function becomes the following:

for each $t_i \in \text{tidlist}(\{x_1\})$ emitted by $reduce_1$ **do**
  for each $x_2 \in t_i$ **do**
    if $x_1 < x_2$
    then emit $\langle$tid i, $\{x_1, x_2\} \in t_i\rangle$

Similar to the $reduce_1$ function, our data science model uses the $reduce_2$ function to shuffle and sort these $\langle i, X \in t_i\rangle$-pairs (where $X = \{x_1, x_2\}$) so as to find and count the number of transactions containing X. See below:

$reduce_2$: list of $\langle \text{tid } i, X \in t_i \rangle \mapsto \langle X, \text{tidlist}(X), \sup(X)\rangle$

with a detailed definition as follows:

for each $X=\{x_1, x_2\}$ emitted by map_2 **do**
  set $\sup(X) = 0$
  set $\text{tidlist}(X) = \{\}$
  for each $\langle$tid i, X$\rangle$ emitted by map_2 **do**
    $\sup(X) = \sup(X) + 1$
    $\text{tidlist}(X) = \text{tidlist}(X) \cup \{i\}$
  if $\sup(X) \geq$ user-specified *minsup* threshold
  then emit $\langle X, \text{tidlist}(X), \sup(X)\rangle$

This $reduce_2$ function returns every frequent pair $X=\{x_1, x_2\}$, all the transactions that contain both items x_1 and x_2, as well as the number of transactions containing both items.

Next, our data science model applies similar sets of map_k and $reduce_k$ functions to recursively find frequent triplets, quadruplets, quintuplets, etc. (i.e., frequent k-itemsets for $k \geq 3$). For instance, the map_3 function, which returns a $\langle$tid i, co-occurring item triplet $X \in t_i\rangle$-pair for every transaction t_i, can be specified as follows:

map_3: $\langle X=\{x_1, x_2\}, \text{tidlist}(X)\rangle \mapsto \langle$tid i, $\{x_1, x_2, x_3\} \in t_i \rangle$,

which could be defined in details as follows:

for each $t_i \in$ tidlist($\{x_1, x_2\}$) emitted by reduce$_2$ **do**
  for each $x_3 \in t_i$ **do**
    if $x_1 < x_2 < x_3$
    then emit $\langle$tid i, $\{x_1, x_2, x_3\} \in t_i\rangle$

Knowing the condition that $x_1 < x_2$ holds for every $\{x_1, x_2\}$-pair emitted by the reduce$_2$ function, our data science model only needs to check the condition $x_2 < x_3$. Hence, the final map$_3$ function becomes the following:

for each $t_i \in$ tidlist($\{x_1, x_2\}$) emitted by reduce$_2$ **do**
  for each $x_3 \in t_i$ **do**
    if $x_2 < x_3$
    then emit $\langle$tid i, $\{x_1, x_2, x_3\} \in t_i\rangle$

These $\langle$tid i, $\{x_1, x_2, x_3\} \in t_i\rangle$-pairs are then shuffled and sorted so as to find and count the number of transactions containing X=$\{x_1, x_2, x_3\}$. See below:

reduce$_3$: list of $\langle$tid i, X$\in t_i\rangle \mapsto \langle$X, tidlist(X), sup(X)$\rangle$

with a detailed definition as follows:

for each X=$\{x_1, x_2, x_3\}$ emitted by map$_3$ **do**
  set sup(X) = 0
  set tidlist(X) = {}
  for each $\langle$tid i, X$\rangle$ emitted by map$_2$ **do**
    sup(X) = sup(X) + 1
    tidlist(X) = tidlist(X) $\cup$ $\{i\}$
  if sup(X) $\geq$ user-specified *minsup* threshold
  then emit $\langle$X, tidlist(X), sup(X)$\rangle$

This reduce$_3$ function returns every frequent triplet X = $\{x_1, x_2, x_3\}$, all the transactions that contain X, as well as the number of transactions containing X.

These map$_3$ and reduce$_3$ functions are instances of the generalized map$_k$ and reduce$_k$ functions (for $k \geq 3$), which can be defined as follows. More specifically, let (i) Y = $\{x_1, \ldots, x_{k-1}\}$ be a frequent pattern of cardinality k–1 (i.e., a frequent (k–1)-itemset) and (ii) X=$\{x_1, \ldots, x_{k-1}, x_k\}$ be a k-itemset extension of Y. Then, the map$_k$ function, which returns a $\langle$tid i, co-occurring item k-tuple X$\in t_i\rangle$-pair for every transaction t_i, can be generalized and specified as follows:

map$_k$: $\langle$Y, tidlist(Y)$\rangle \mapsto \langle$tid i, (X=Y$\cup\{x_k\}$)$\in t_i\rangle$,

with a detailed definition as follows:

for each $t_i \in$ tidlist($\{x_1,\ldots,x_{k-1}\}$) emitted by reduce$_{k-1}$ **do**
  for each $x_k \in t_i$ **do**
    if $x_{k-1} < x_k$
    then emit $\langle$tid i, $\{x_1, \ldots, x_{k-1}, x_k\} \in t_i\rangle$

The resulting $\langle$tid i, co-occurring item k-tuple X$\in t_i\rangle$-pairs returned by this map$_k$ function are then shuffled and sorted. Each processor then executes the following reduce$_k$ function on these shuffled and sorted pairs to find frequent patterns of cardinality k (i.e., frequent k-itemsets). So, the reduce$_k$ function can be generalized and specified as follows:

reduce$_k$: list of $\langle$tid i, X$\in t_i\rangle \mapsto \langle$X, tidlist(X), sup(X)$\rangle$

with a detailed definition as follows:

for each X=$\{x_1, \ldots, x_{k-1}, x_k\}$ emitted by map$_k$ **do**
  set sup(X) = 0
  set tidlist(X) = {}
  for each $\langle$tid i, X$\rangle$ emitted by map$_k$ **do**
    sup(X) = sup(X) + 1
    tidlist(X) = tidlist(X) $\cup$ $\{i\}$
  if sup(X) $\geq$ user-specified *minsup* threshold
  then emit $\langle$X, tidlist(X), sup(X)$\rangle$

To recap, our data science model applies the first set of map$_1$ and reduce$_1$ functions to convert the horizontal (transaction-centric) representation of big data set into an equivalent vertical (item-centric) representation. As a benefit of such a conversion, our data science model also counts the support of each item and thus finds all frequent singletons. Afterwards, the model applies subsequent sets of map$_k$ and reduce$_k$ functions to find frequent patterns of higher cardinalities (i.e., frequent k-itemsets for $k \geq 2$) in a divide-and-conquer manner. For instance, based on the tidlist for frequent item x_1 (i.e., the list of transactions containing x_1) returned by the reduce$_1$ function, our model computes how many of these transactions also contain x_2 so as to find frequent pairs (i.e., frequent 2-itemsets). Based on the resulting transaction list for these frequent 2-itemsets returned by the reduce$_2$ function, our model then computes how many of the transactions also contain x_3 so as to find frequent triplets (i.e., frequent 3-itemsets). This mining process is repeated until we find all frequent patterns. Note that this mining process conducted by our data science model is different from the Apriori-based (or vertical) algorithms because the latter mine data sets in a breadth-first fashion whereas ours mines big data sets in a depth-first, divide-and-conquer fashion. Moreover, our data science model is also different from the tree-based (or hyperlinked array structure based) algorithms because the latter constantly build sub-trees or adjust hyperlinks during the mining process whereas ours does not need to do so.

B. An Illustrative Example on the Application of Our Data Science Model for Frequent Pattern Mining

To illustrate our data science model in mining frequent patterns from big data, let us consider the following portion of a big data set in a horizontal representation:

- Transaction t_1: {items 2, 5};
- Transaction t_2: {items 1, 3, 5};
- Transaction t_3: {items 1, 5};
- Transaction t_4: {items 2, 3, 5};
- Transaction t_5: {items 1, 2, 3, 4}; and
- Transaction t_6: {items 1, 2, 3, 5}.

Here, there are six transactions about five different items (items 1, 2, 3, 4 and 5). There are two items (namely, items 2 and 5) in transaction t_1.

Then, with *minsup*=2, our data science model applies the map$_1$ function to each of the six transactions in this big data set. The master node reads and divides big data set in partitions. The map$_1$ returns a list of 18 $\langle$tid i, item $x\in t_i\rangle$-pairs: $\langle t_1$, item 2$\rangle$, $\langle t_1$, item 5$\rangle$, $\langle t_2$, item 1$\rangle$, $\langle t_2$, item 3$\rangle$, $\langle t_2$, item 5$\rangle$, $\langle t_3$, item 1$\rangle$, $\langle t_3$, item 5$\rangle$, $\langle t_4$, item 2$\rangle$, $\langle t_4$, item 3$\rangle$, $\langle t_4$, item 5$\rangle$, $\langle t_5$, item 1$\rangle$, $\langle t_5$, item 2$\rangle$, $\langle t_5$, item 3$\rangle$, $\langle t_5$, item 4$\rangle$,

⟨t_6, item 1⟩, ⟨t_6, item 2⟩, ⟨t_6, item 3⟩, and ⟨t_6, item 5⟩. Each pair captures the tid i and an item $x \in t_i$. Afterwards, our data science model applies the reduce$_1$ function to these ⟨tid i, item $x \in t_i$⟩-pairs so that each processor shuffles and sorts these 18 pairs to compute the support of these items. By incorporating the user-specified threshold *minsup*=2 (which expresses that a pattern is frequent if it occurs in at least *minsup*=2 transactions), our data science model returns a vertical representation of the data set in the form ⟨{x}, tidlist({x}), sup({x})⟩-triplets:

- ⟨item 1, {t_2, t_3, t_5, t_6}, sup({item 1})=4⟩;
- ⟨item 2, {t_1, t_4, t_5, t_6}, sup({item 2})=4⟩;
- ⟨item 3, {t_2, t_4, t_5, t_6}, sup({item 3})=4⟩; and
- ⟨item 5, {t_1, t_2, t_3, t_4, t_6}, sup({item 4})=5⟩.

Note that the reduce$_1$ function does not return ⟨item 4, {t_5}, sup({item 4})=1⟩ because sup({item 4})=1 < 2=*minsup* (i.e., item 4 is infrequent).

Then, for $k \geq 2$, our data science model applies the map$_k$ and reduce$_k$ functions. When k = 2, Y is instantiated to each of the above four frequent singletons (i.e., {item 1}, {item 2}, {item 3} and {item 5}) and the map$_2$ function returns the following ⟨tid i, (X=Y∪{x_2})∈ t_i⟩-pairs: ⟨t_1, {items 2, 5}⟩, ⟨t_2, {items 1, 3}⟩, ⟨t_2, {items 1, 5}⟩, ⟨t_2, {items 3, 5}⟩, ⟨t_3, {items 1, 5}⟩, ⟨t_4, {items 2, 3}⟩, ⟨t_4, {items 2, 5}⟩, ⟨t_4, {items 3, 5}⟩, ⟨t_5, {items 1, 2}⟩, ⟨t_5, {items 1, 3}⟩, ⟨t_5, {items 2, 3}⟩, ⟨t_6, {items 1, 2}⟩, ⟨t_6, {items 1, 3}⟩, ⟨t_6, {items 1, 5}⟩, ⟨t_6, {items 2, 3}⟩, ⟨t_6, {items 2, 5}⟩ and ⟨t_6, {items 3, 5}⟩. These 17 pairs are then shuffled and sorted by reduce$_2$ function to get the following frequent pairs (i.e., frequent 2-itemsets) together with their relevant information (e.g., tidlists and support):

- ⟨{items 1, 2}, {t_5, t_6}, sup({items 1, 2})=2⟩;
- ⟨{items 1, 3}, {t_2, t_5, t_6}, sup({items 1, 3})=3⟩;
- ⟨{items 1, 5}, {t_2, t_3, t_6}, sup({items 1, 5})=3⟩;
- ⟨{items 2, 3}, {t_4, t_5, t_6}, sup({items 2, 3})=3⟩;
- ⟨{items 2, 5}, {t_1, t_4, t_6}, sup({items 2, 5})=3⟩; and
- ⟨{items 3, 5}, {t_2, t_4, t_6}, sup({items 3, 5))=3⟩.

When k = 3, Y is instantiated to each of these six frequent 2-itemsets. The map$_3$ function returns seven ⟨tid i, (X=Y∪{x_3})∈ t_i⟩-pairs: ⟨t_2, {items 1, 3, 5}⟩, ⟨t_4, {items 2, 3, 5}⟩, ⟨t_5, {items 1, 2, 3}⟩, ⟨t_6, {items 1, 2, 3}⟩, ⟨t_6, {items 1, 2, 5}⟩, ⟨t_6, {items 1, 3, 5}⟩ and ⟨t_6, {items 2, 3, 5}⟩. Shuffling and sorting them, the reduce$_3$ function returns three frequent triplets (i.e., frequent 3-itemsets):

- ⟨{items 1, 2, 3}, {t_5, t_6}, sup({items 1, 2, 3})=2⟩;
- ⟨{items 1, 3, 5}, {t_2, t_6}, sup({items 1, 3, 5})=2⟩; and
- ⟨{items 2, 3, 5}, {t_4, t_6}, sup({items 2, 3, 5})=2⟩.

When k = 4, Y is instantiated to each of these three frequent 3-itemsets. The map$_4$ function returns one ⟨tid i, (X=Y∪{x_4})∈ t_i⟩-pair, namely, ⟨t_6, {items 1, 2, 3, 5}⟩. However, the reduce$_4$ function does not return any frequent quadruplet (i.e., frequent 4-itemset) because {items 1, 2, 3, 5} is infrequent.

Hence, at the end of this big data analytics and mining process, our data science model found four frequent 1-itemsets, six frequent 2-itemsets, and three frequent 3-itemsets, for a total of 13 frequent patterns from the interesting portion of this big data set.

IV. Social Network Analysis

So far, we have described our data science model for big data analytics of frequent patterns and have illustrated how our data science model finds all frequent patterns. In this section, we apply our data science model to *social networks*—which are important sources of big data—for *social network analysis*.

A. Social Networks as Big Data Sources

Social networks [15], [17] are generally made of social entities (e.g., individuals, corporations, collective social units, or organizations) that are linked by some specific types of interdependencies (e.g., kinship, friendship, common interest, beliefs, or financial exchange). A social entity is connected to another entity as his next-of-kin, friend, collaborator, co-author, classmate, co-worker, team member, and/or business partner.

Social network analysis [10], [16], [30] helps create business value by identifying and managing internal or external social networks. In particular, big data analytics of social networks computationally facilitates social studies and human-social dynamics in these big data networks, as well as designs and uses information and communication technologies for dealing with social context in various social networking websites such as Facebook and Twitter [4], [11], [13], [14].

In Facebook, users can create a personal profile, add other Facebook users as friends, and exchange messages. In addition, Facebook users can also join common-interest user groups and categorize their friends into different customized lists (e.g., classmates, co-workers). The number of (mutual) friends may vary from one Facebook user to another. It is not uncommon for a user p to have hundreds or thousands of friends. Note that, although many of the Facebook users are linked to some other Facebook users via mutual friendship (i.e., if p is a friend of another user q, then q is also a friend of p), there are situations in which such a relationship is either not mutual or no longer mutual. To handle these situations, Facebook added the functionality of "subscribe" in 2011, which was later relabelled as "follow" in 2012. Specifically, a user can subscribe or follow public postings of some other Facebook users without the need of adding them as friends. So, if many of p's friends followed some individual users or groups of users, then p might also be interested in following the same individual users or groups of users. Furthermore, the "like" button is another social networking feature that allows users to express their appreciation of content such as status updates, comments, photos, and advertisements.

Twitter is another online social networking and blogging service that allows users to read the tweets of other users by "following" them. As such, a Twitter user p may be interested in knowing popular followees. In other words, if many of p's friends follow some individual users or groups

of users, then *p* might also be interested in following the same individual users or groups of users.

In recent years, the number of users in these social networking sites has grown rapidly (e.g., 1.65 billion monthly active Facebook users[1] and 310 million monthly active Twitter users[2] at the end of March 2016). This big number of users creates an even more massive number of relationships in the social networks. Hence, application of our data science model helps analyze the social networks and reveal interesting patterns.

B. An Illustrative Example on the Application of Our Data Science Model for Social Network Analysis

To illustrate our data science model in analyzing mining social network, let us consider the following portion of a big social data set capturing the following information:

- Alice likes social networking pages 2 & 5;
- Bob likes social networking pages 1, 3 & 5;
- Carol likes social networking pages 1 & 5;
- Dan likes social networking pages 2, 3 & 5;
- Eva likes social networking pages 1, 2, 3 & 4; and
- Frank likes social networking pages 1, 2, 3 & 5.

By using sets of map_k and $reduce_k$ functions in a fashion similar to that described in Section III.B, our data science model conducts a social network analysis. Let *minsup*=2 (expressing that a collection of social networking pages is *popular* if they are liked by at least 2 fans). By applying the map_1 function to the above social information, our model returns a list of 18 ⟨fan *f*, social networking page liked by *f*⟩-pairs, which include ⟨Alice, page 2⟩, ⟨Alice, page 5⟩, …, ⟨Frank, page 1⟩, ⟨Frank, page 2⟩, ⟨Frank, page 3⟩, and ⟨Frank, page 5⟩. Then, applying the $reduce_1$ function to these 18 pairs, our model discovers four popular individual social networking pages 1, 2, 3 and 5 (e.g., social networking page 1 is liked by Bob, Carol, Eva & Frank). Afterwards, by applying the map_2 function, a processor finds Alice's favourite pair of social networking pages (i.e., pages 2 & 5). Another processor finds Bob's favourite pairs of social networking pages (e.g., pages 1 & 3). Similar actions performed by other processors to find favourite pairs of social networking pages of other four fans. By applying the $reduce_2$ function, our model discovers six popular pairs of social networking pages (e.g., pages 1 & 2). Similarly, by applying the map_3 function, our model finds Bob's favourite triplets of social networking pages (i.e., pages 1, 3 & 5). The model applies the $reduce_3$ function to discover three popular triplets of social networking pages (e.g., pages 1, 3 & 5). The map_4 function then finds Frank's favourite triplet of social networking pages (i.e., pages 1, 3 & 5), which happens to be not popular.

To summarize, our data science model discovers following 13 collections of popular social networking pages:

- Four popular individual social networking pages {1}, {2}, {3} and {5};
- six popular pairs of social networking pages {1, 2}, {1, 3}, {1, 5}, {2, 3}, {2, 5} and {3, 5}; and
- three popular triplets of social networking pages {1, 2, 3}, {1, 3, 5}, and {2, 3, 5}.

Discovery of these 13 collections help social networking sites make recommendations of social networking pages to their users or fans. Moreover, although we illustrated an application of our data science model in social network analysis on the discovery of popular social networking pages liked by their fans, our model can also be applicable to many other tasks in social network analysis (e.g., discovery of interesting social networking pages followed by many followers).

V. Evaluation

To evaluate our data science model, we compared it with many of the algorithms mentioned in Section II (e.g., Eclat, FP-growth, H-mine, TD-FP-Growth, VIPER) using (i) IBM synthetic data sets (e.g., with 100,000 transactions), (ii) the online retail data set (with 541,909 transactions) from UCI Machine Learning Repository[3], and (iii) Stanford Network Analysis Project (SNAP) ego-Facebook data set (with 88,234 connections among 4,039 social entities) and ego-Twitter data sets (with 1,768,149 connections among 81,306 social entities)[4]. All experiments were run using either (i) a single machine with an Intel Core i7 4-core processor (1.73 GHz) and 8 GB of main memory running a 64-bit Windows 7 operating system, or (ii) the Amazon Elastic Compute Cloud (EC2) cluster—specifically, 11 High-Memory Extra Large (m2.xlarge) computing nodes[5]. We implemented our data science model in the Java programming language. The stock version of Apache Hadoop 0.20.0 was used. The results are shown in Figure 1 are based on the average of multiple runs. Runtime includes CPU and I/Os in the mining process. Figures 1(a) and 1(b) show that our data science model is more efficient than five existing algorithms. Figure 1(a) also shows that our data science model is scalable with respect to the number of transactions or connections. Figure 1(c) shows the speedup of using EC2 cluster over a single machine.

VI. Conclusions

As frequent pattern mining is an important data mining task, many serial, parallel and distributed frequent pattern mining algorithms have been proposed. While these algorithms are popular and benefit from a few advantages, they also suffer from some disadvantages. In the current era of big data, a wide variety of high volumes of valuable data of different veracities can be easily collected and generated at a high velocity. These big data lead to additional challenges for frequent pattern mining. In this paper, we present a data science model for big data analytics of frequent patterns with MapReduce. We evaluated our model by using social networks, which are good examples of big data. Evaluation results show the efficiency and practicality of our data science model in mining and analyzing big data for the discovery of interesting frequent patterns from

[1] http://newsroom.fb.com/company-info/
[2] https://about.twitter.com/company
[3] https://archive.ics.uci.edu/ml/
[4] http://snap.stanford.edu/data/
[5] http://aws.amazon. com/ec2/

various real-life applications including social network analysis. As ongoing work, we are conducting more analyses and experiments, including an in-depth study on the complexity analysis of our model. Moreover, we are also extending our model to (i) incorporate visual analytics and (ii) run on the Spark environment.

ACKNOWLEDGMENT

This project is partially sponsored by NSERC (Canada) and University of Manitoba.

REFERENCES

[1] L. Abidi, C. Cérin, G. Fedak, and H. He, "Towards an environment for doing data science that runs in browsers," Proc. SmartCity (DataCom) 2015, pp. 662–667.

[2] R. Aggarwal and R. Srikant, "Fast algorithms for mining association rules," Proc. VLDB 1994, pp. 487–399.

[3] R. Agrawal and J. C. Shafer, "Parallel mining of association rules," Fast algorithms for mining association rules," IEEE TKDE, 8(6), pp. 962–969, Dec. 1996.

[4] É. Antoine, A. Jatowt, S. Wakamiya, Y. Kawai, and T. Akiyama, "Portraying collective spatial attention in Twitter," Proc. ACM KDD 2015, pp. 39–48.

[5] A. Cuzzocrea and C. K. Leung, "Computing theoretically-sound upper bounds to expected support for frequent pattern mining problems over uncertain big data," Proc. IPMU 2016, Part II, pp. 379–392.

[6] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters." CACM, 51(1), pp. 107–113, Jan. 2008.

[7] X. Feng, D. Wang, M. Huang, and X. X. Sun, "An approach of discovering casual knowledge for alert correlating based on data mining," Proc. IEEE DASC 2014, pp. 57–62.

[8] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," Proc. ACM SIGMOD 2000, pp. 1– 12.

[9] D. Huang, Y. Song, R. Routray, and F. Qin, "Smart Cache: an optimized MapReduce implementation of frequent itemset mining," Proc. IC2E 2015, pp. 16–25.

[10] F. Jiang, and C. K. Leung, "Mining interesting 'following' patterns from social networks," Proc. DaWaK 2014, pp. 308–319.

[11] H. K. Kong, Y. W. Wu, B. P. Bailey, and K. Karahalios, "Culture, imagined audience, and language choices of multilingual Chinese and Korean students on Facebook," Proc. SocInfo 2015, pp. 1–16.

[12] M.-H. Kuo, D. Chrimes, B. Moa, and W. Hu, "Design and construction of a big data analytics framework for health applications," Proc. SmartCity (DataCom) 2015, pp. 631–636.

[13] C. Lauschke and E. Ntoutsi, "Monitoring user evolution in Twitter," Proc. IEEE/ACM ASONAM 2012, pp. 972–977.

[14] C. K. Leung, "Big data mining and analytics," Encyclopedia of Business Analytics and Optimization, pp. 328–337, Feb. 2014.

[15] C. K. Leung, I. J. M. Medina, and S. K. Tanbeer, "Analyzing social networks to mine important friends," Social Media Mining and Social Network Analysis: Emerging Research, pp. 90–104, 2013.

[16] C. K. Leung, S. K. Tanbeer, A. Cuzzocrea, P. Braun, R. K. MacKinnon, "Interactive mining of diverse social entities," KES Journal, 20(2), pp. 97–111, May 2016.

[17] C. K. Leung and H. Zhang, "Management of distributed big data for social networks," Proc. IEEE/ACM CCGrid 2016.

[18] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang, "PFP: parallel FP-growth for query recommendation," Proc. ACM RecSys 2008, pp. 107–114.

[19] M.-Y. Lin, P.-Y. Lee, and S.-C. Hsueh, "Apriori-based frequent itemset mining algorithms on MapReduce," Proc. ICUIMC 2012, art. 76.

[20] X. Lin, "MR-Apriori: association rules algorithm based on MapReduce," Proc. IEEE ICSESS 2014, pp. 141–144.

[21] X. Luo, H. Yuan, and Q. Luo, "On discovering feasible periodic patterns in large database," Proc. IEEE DASC 2013, pp. 344–351.

[22] S. Madden, "From databases to big data," IEEE Internet Computing, 16(3), pp. 4–6, May-June 2012.

[23] S. Moens, E. Aksehirli, and B. Goethal, "Frequent itemset mining for big data," Proc. IEEE Big Data 2013, pp. 111–118.

[24] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, "HMine: hyper-structure mining of frequent patterns in large databases," Proc. IEEE ICDM 2001, pp. 441–448.

[25] P. Shenoy, J. R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah, "Turbo-charging vertical mining of large databases," Proc. ACM SIGMOD 2000, pp. 22–33.

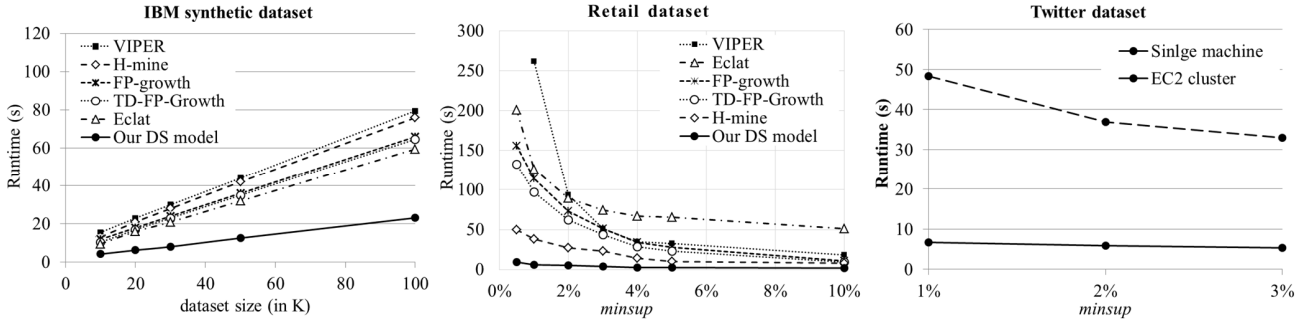
[26] K. Wang, L. Tang, J. Han, and J. Liu, "Top down FP-growth for association rule mining," Proc. PAKDD 2002, pp. 334– 340.

[27] M. J. Zaki, "Parallel and distributed association mining: a survey," IEEE Concurrency, 7(4), pp. 14–25, Oct.–Dec. 1999.

[28] M. J. Zaki, "Scalable algorithms for association mining," IEEE TKDE, 12(3), pp. 372–390, May/June 2000.

[29] Y. Zhang, Q. Li, and B. Liu, "MapReduce implementation of XML keyword search algorithm," Proc. SmartCity (DataCom) 2015, pp. 721–728.

[30] C. Zhou, P. Zhang, J. Guo, X. Zhu, and L. Guo, "UBLF: an upper bound based approach to discover influential nodes in social networks," Proc. IEEE ICDM 2013, pp. 907–916.



(a) IBM synthetic dataset (b) UCI ML online retail data set (c) SNAP ego-Twitter data set

Figure 1. Evaluation results.