Scalable Mining of High-Utility Sequential Patterns with Three-Tier MapReduce Model

JERRY CHUN-WEI LIN*, Western Norway University of Applied Sciences, Norway
YOUCEF DJENOURI, SINTEF Digital, Norway
GAUTAM SRIVASTAVA, Brandon University, Canada, Canada and China Medical University, Taiwan
YUANFA LI, Harbin Institute of Technology (Shenzhen), China
PHILIP S. YU, University of Illinois at Chicago, USA

High-utility sequential pattern mining (HUSPM) is a hot research topic in recent decades since it combines both sequential and utility properties to reveal more information and knowledge rather than the traditional frequent itemset mining or sequential pattern mining. Several works of HUSPM have been presented but most of them are based on main memory to speed up mining performance. However, this assumption is not realistic and not suitable in large-scale environments since in real industry, the size of the collected data is very huge and it is impossible to fit the data into the main memory of a single machine. In this paper, we first develop a parallel and distributed three-stage MapReduce model for mining high-utility sequential patterns based on large-scale databases. Two properties are then developed to hold the correctness and completeness of the discovered patterns in the developed framework. In addition, two data structures called sidset and utility-linked list are utilized in the developed framework to accelerate the computation for mining the required patterns. From the results, we can observe that the designed model has good performance in large-scale datasets in terms of runtime, memory, efficiency of the number of distributed nodes, and scalability compared to the serial HUSP-Span approach.

Additional Key Words and Phrases: High-utility sequential pattern mining, MapReduce, large-scale, parallel and distributed.

ACM Reference Format:
Jerry Chun-Wei Lin, Youcef Djenouri, Gautam Srivastava, Yuanfa Li, and Philip S. Yu. 2021. Scalable Mining of High-Utility Sequential Patterns with Three-Tier MapReduce Model. *ACM Trans. Knowl. Discov. Data.* 1, 1, Article 1 (January 2021), 25 pages. https://doi.org/10.1145/3487046

1 INTRODUCTION

Data mining, which also can be referred as Knowledge Discovery in Databases (KDD) [1, 8], has been widely studied and utilized in many applications and domains. The fundamental knowledge in KDD can be classified as many representations, e.g., association-rule mining (ARM) [2, 17], sequential pattern mining (SPM) [3, 14, 16, 33, 35], high-utility itemset mining (HUIM) [9, 15, 20, 22, 28, 43], among others. For generic ARM, it only takes the occurrence frequency of the items into account, but the other factors, such as interestingness, weight, or importance are not considered; the discovered information from ARM may become incomplete. To address

*This is the corresponding author

Authors' addresses: Jerry Chun-Wei Lin, Western Norway University of Applied Sciences, Bergen, Norway, jerrylin@ieee.org; Youcef Djenouri, SINTEF Digital, Oslo, Norway, Youcef.Djenouri@sintef.no; Gautam Srivastava, Brandon University, Canada, Brandon, Canada and China Medical University, Taichung, Taiwan, srivastavag@brandonu.ca; Yuanfa Li, Harbin Institute of Technology (Shenzhen), Shenzhen, China, liyuanfa@stu.hit.edu.cn; Philip S. Yu, University of Illinois at Chicago, Chicago, USA, psyu@uic.edu.

1556-4681/2021/1-ART1 $15.00
https://doi.org/10.1145/3487046

this problem, HUIM considers two factors such as unit profit of the items and the quantity of the items into account to find more meaningful patterns than that of ARM. It thus has become an important topic in the field of KDD; however, it is not suitable for time-series or sequential data in many realistic domains and applications, for example, stock market analysis or DNA sequence analysis. Besides, there is a large number of time-series and sequence characteristic data with different meanings and effects at different times in fields like consumer behaviour analysis, business intelligence, fault risk prediction, and medical diagnosis, that cannot be analyzed by the traditional HUIM nor ARM.

To solve the limitation of the traditional ARM or HUIM, SPM is used to find the interesting subsequences in a set of sequences, where the interestingness of a subsequence can be measured in terms of various criteria such as its occurrence frequency, length, and profit. SPM shows numerous real-life applications because data is naturally encoded as sequences of symbols in many fields such as bio-informatics, e-learning, market basket analysis, texts, and web-page click-stream analysis. SPM has, however, the limitation by only considering the occurrence frequency of the sequence, thus if a sequence is with low frequency but high utility, it could be ignored in SPM. For example, although the sales volume of a sequence behaviour $\mathbb{A}$ (= buying diamond rings first then buying necklaces afterward) is lower than the sales volume of a sequence behaviour $\mathbb{B}$ (= buying bread first then buying milk afterward), the profit of a sequence $\mathbb{A}$ is much higher than the profit of a sequence $\mathbb{B}$. Clearly, $\mathbb{A}$ sequence behaviour is more conducive to merchants. However, in general, the frequency of $\mathbb{A}$ sequence is very low, and frequent sequence pattern mining cannot find such important information.

High-utility sequential pattern mining (HUSPM) [42, 46, 48] has broader application prospects and needs when compared with traditional SPM and HUIM. For example, HUSPM can find a high-margin product sequence by analyzing sales data of a supermarket, thereby helping the supermarket to formulate commodity promotion strategies and provide a more reasonable commodity procurement plan. In bio-informatics [51], HUSPM can simultaneously consider temporal characteristics and importance of genes, and it can analyze the relationship between the top-k efficient gene sequences and diseases (such as pneumonia) through inter-gene interactions in disease diagnosis. As HUSPM is an emerging field that has attracted the attention of an increasing number of researchers, several works [42, 46] have been initiated on HUSPM. However, the existing methods are memory-based, which means it is assumed that all the data can fit into the main memory of a single machine. Current trends show that high volumes of data are produced in real-life applications. Memory-based algorithms are not realistic for application areas with large-scale datasets. However, mining high-utility sequential patterns (HUSPs) from large-scale datasets is an emerging topic but not a simple task. The limitations of the current works are stated below, which are the motivation of this paper for further improvement.

- It is impossible to carry out the task of mining HUSPs in one machine due to the rapid growth of data. Designing distributed and parallel methods plays an important role in dealing with this large-scale problem.
- The utility of a sequence needs to be calculated and the input sequences are distributed in different work nodes; the local utility of a sequence of each node cannot determine whether a sequence is a global high-utility pattern or not in an entire database. Therefore, a method to sum all the local utilities of a sequence needs to be designed so that the global utility value of a sequence can be obtained efficiently.
- Traditional memory-based algorithms are mostly "generate-and-test"; that is, first, they produce the candidates, and then it is tested as to whether it is a HUSP or not. The above procedure is recursively performed until the set of candidates is empty (level-wise approach). Thus, the computational cost and memory usage to mine the required patterns are relevant high.

In HUIM, the distributed and parallel methods are Apriori-based [26] or using sampling model [10] to mine HUIs. The former is the same as the methods in frequent itemsets mining using iterative MapReduce that requires higher computational cost. The latter parallelizes the HUI-Miner algorithm by adopting the sampling model to obtain the approximate number of HUIs. To better solve the above limitations for efficiently and accurately

revealing the number of HUSPs in the large-scale datasets, we firstly propose a distributed and parallel high-utility sequential pattern mining framework to handle large-scale datasets. Four contributions of this paper are then stated below:

- A three-stage MapReduce framework based on the Spark platform is first designed to efficiently and accurately mine HUSPs from big datasets.
- Two properties are then investigated and designed to ensure the correctness and completeness of the discovered HUSPs from distributed and parallel environments, which can greatly improve mining efficiency.
- Two data structures named `sidset` and `Utility-Linked List` are developed in this paper to reduce the time complexity, as well as speed up mining performance.
- Extensive experiments on various large-scale datasets are conducted to show that the proposed MapReduce-based model and utilized two structures achieve better performance than the generic and serial HUS-Span algorithm.

Section 2 provided a detailed survey of the relevant works in this paper. Section 3 stated the basic preliminary and problem statement of this paper. Section 4 mentioned the proposed MapReduce models and the developed algorithms. Section 5 showed the experiments to evaluate the performance of the designed model compared to the other works. Finally, section 6 concluded the achievements of this paper and extended directions for future works.

2 LITERATURE REVIEW

Agrawal *et al.* [2] and Han *et al.* [17] respectively presented the Apriori and FP-growth algorithms to solve the Association-rule mining (ARM) problem. To handle the realistic situations regarding sequence ordering, Agrawal *et al.* [3] then first proposed the concept of sequential-pattern mining (SPM) and designed the AprioriSome, AprioriAll, and DynamicSome algorithms for SPM. Srikant *et al.* [35] proposed a GSP algorithm that uses a hash tree to keep the candidate sets for improving the efficiency of the AprioriAll algorithm. The FreeSpan [16] and PrefixSpan [33] were also respectively presented to speed up the mining performance of SPM.

Since ARM and SPM only explore the occurrence frequency of the items in the database, it ignores many important factors, e.g., importance, interestingness, weight, unit profit of items, among others, to mine the association rules. Chan *et al.* [9] first introduced the concept of utility into frequent itemset mining to help decision-makers develop more favourable strategies. Yao *et al.* [45] proposed a formal definition of efficient itemset mining, using utility values instead of support as a measure of itemsets. Liu *et al.* [20] proposed the transaction-weighted utility (TWU) concept for estimating the upper bound of the itemset utility value. Tseng *et al.* extended the FP-tree and proposed the UP-growth+ [40] algorithm to exploit the nature of the tree for compressing the search space. Lin *et al.* [21] proposed HUP-tree, which is based on the TWU concept and FP-tree, and they used the tree structure to save the database, which speeds up the mining process of the proposed HUP-growth algorithm. Liu *et al.* [22] proposed the HUI-Miner algorithm, which converts the original database into a list structure and mines efficient itemsets from the list and thus avoids the generation of candidate sets. Zida *et al.* [50] designed a novel algorithm EFIM, proposed two new utility upper bounds, and more effectively reduced the search space. Presently, the research on high-utility itemset mining is still in development. Kim *et al.* [18] then developed a utility model for handling the large-scale stream data for discovering the high-utility patterns. The designed model divides the stream data into several fixed-sized data and processes each batch of data in a window according to the added time by the designed decaying factor to differently show its importance. Vo *et al.* [41] suggested having the dynamic profit tables for the itemsets in real applications and presented a multi-core framework for efficiently mining the high-utility itemsets. The designed model can then greatly reduce the cost of database rescans thus the performance can be improved. Nam *et al.* [32] considered the influence of the recent data compared to the old one, a model focused on finding the high-utility itemsets from the time-sensitive

databases was presented by applying the damped-window model. Mai *et al.* [31] presented a model to mine the high-utility association rules, which enables users to iteratively choose the preferred weights for the discovered rules based on the developed semi-lattice structures. To speed up mining performance, Yun *et al.* [47] presented a pre-large-based concept for mining high-utility itemsets in dynamic databases. The deletion operation is considered here to maintain and update the discovered patterns by 9 cases of the pre-large concept to reduce the number of database rescans. Moreover, the pre-large concept is also adapted to the sensor network situation [38] to combine all the discovered high-utility itemsets in a fusion model, which is applicable in industrial applications. Several works [7, 15, 27, 29, 43] in the direction of HUIM have been extensively presented and discussed but most of them can only be performed on a single machine for running small datasets.

High-utility sequential-pattern mining (HUSPM) is a field that has emerged in recent years. HUSPM was first used in the sequence mining of website logs [49]. Shie *et al.* proposed the UMSP [36] algorithm and the UM-span [37] algorithm for mining high-utility mobile sequences in mobile business applications. To exploit the usefulness of web page access sequence data, Ahmed *et al.* [4] proposed two tree structures, called UWAS-tree and IUWAS-tree, for processing static and dynamic databases, respectively. Subsequently, Ahmed *et al.* [5] proposed a high-utility sequential pattern mining algorithm for processing general sequences, namely, the layer-by-layer search UL algorithm and the pattern-extended US algorithm. Yin *et al.* [46] officially defined high-utility sequential pattern mining and proposed an efficient algorithm, USpan, for mining general sequence patterns with utility values. To simplify the parameter setting, Yin *et al.* [48] then proposed the TUS algorithm for discovering the top-k high-utility sequential patterns. Lan *et al.* [25] first introduced the concept of fuzziness into sequence mining and then proposed a high-utility sequential pattern mining algorithm to simplify the mining results and reduce the search space. Alkan *et al.* [6] proposed a high-utility sequential pattern extraction (HuspExt) algorithm. It calculates a Cumulated Rest of Match (CRoM) to obtain a smaller upper bound to reduce the complexity of the algorithm. Wang *et al.* [42] subsequently proposed the HUS-Span algorithm to reduce useless candidate sets by two utility upper bound PEUs and RSUs. The HUS-Span is the generic and serial algorithm that can be used to discover the set of HUSPs from the database based on the developed high sequence weighted utility (SWU) to maintain the downward closure property, which is the standard and the state-of-the-art algorithm for HUSPM. Their paper also proposes a TKHUS-Span algorithm based on top-k and its performance was tested under three search strategies.

MapReduce [11], which was proposed by Dean and Ghemawat, is a programming framework designed to handle big datasets. It is a parallel and distributed algorithm on a cluster and contains two major procedures, Map and Reduce. Overall, MapReduce provides a reliable, dynamic, and parallel programming framework to deal with big data environments. Regarding the MapReduce framework in pattern mining, Lin *et al.* [24] proposed three algorithms, respectively named SPC, FPC and DPC, by implementing the Apriori in MapReduce framework. The SPC algorithm is used to find the frequent k-itemsets at each level based on the generate-and-test model. The FPC is used to improve the performance of the baseline SPC model using a mapper to calculate k, (k+1), and (k+2) itemsets altogether, and the DPC is used to collect the candidates at different lengths. Those three models are based on the Apriori-like approach thus more execution time is required. Li *et al.* [19] then proposed PFP algorithm, which parallelizes the FP-Growth algorithm on distributed machines without candidate generation. This developed model is based on novel data with the distribution scheme and MapReduce framework to virtually eliminate the communication among several parallel and distributed computers. Moens *et al.* [30] introduced Dist-Eclat and BigFIM algorithms for mining the frequent itemsets based on the MapReduce framework. The first Dist-Eclat is used to speed up mining performance and the latter BigFIM is then used to optimize the execution progress on the large databases. Duong *et al.* [12] presented a two-phase approach for frequent itemset mining in large-scale databases based on the MapReduce and distributed Apriori-like approach. The projection model is also used in the developed model to gradually reduce the database size during the MapReduce phase. In addition to frequent itemset mining, Ge *et al.* [13] considered the uncertainty in sequential databases and

presented a MapReduce framework for mining the uncertain sequential patterns iteratively. A vertical data structure is then used to keep the necessary information of the uncertain sequence databases that can greatly reduce the computational complexity. For HUIM, Lin *et al.* [26] proposed PHUI-Growth for mining HUIs from big data, which is Apriori-based Apache Hadoop framework. However, this approach requires huge computational costs, thus it lacks the efficiency to handle very large databases. Chen *et al.* [10] presented a parallel algorithm of HUI-Miner implemented based on Apache Spark using sampling technologies to reduce the size of input data and approximately mine the HUIs. Based on this model, the approximate set of HUIs is then discovered but the performance can be greatly improved by the sampling model. However, this model could not provide accurate results in terms of the number of HUIs or even the utility of the itemset. It is thus a limitation for making the accurate and precise decision. Wu *et al.* [44] then applies the Hadoop framework for mining the fuzzy high-utility patterns, which is the first work to adapt the fuzzy-set theory into the high-utility itemset mining. However, this model cannot handle large-scale databases. As the rapid growth for the research of HUSPM, it is necessary to develop an efficient model to discover the set of HUSPs from a large-scale efficiency. Sumalatha and Subramanyam [39] then presented a distributed high utility time interval sequential pattern mining (DHUTISP) algorithm based on the MapReduce framework. Two upper-bound models are then designed to reduce the computational cost. However, this model is mainly focused on distributing data into several nodes and the two designed upper-bounds mainly replied on the past works.

3 PRELIMINARIES AND PROBLEM STATEMENT

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of m different items. A quantitative sequence database (q-sequence database) is a set of transactions (or s_{id} in the running example, where id is the transaction id in the database) $D = \{s_1, s_2, \ldots, s_n\}$, where each transaction $s_q \in D$ is a quantitative sequence (q-sequence) and q is its unique identifier (= id). A quantitative itemset (q-itemset) denoted as $X = [(i_1, q_1), (i_2, q_2), \ldots, (i_t, q_t)]$ is a subset of s_q and each item in a q-itemset is a quantitative item that is a pair of the form (i, q), where $i \in I$ and q is a positive integer representing the internal weight locally associated with an item in a transaction/sequence (internal utility). The quantity of a q-item i in a q-itemset X is denoted as $q(i, X)$. Each item $i_k \in I\,(1 \leq k \leq m)$ is also associated with a weight denoted as $pr(i_k)$ representing the external weight globally associated with an item (external utility). In addition, without a loss of generality, since the items are unordered in an itemset, it is assumed that q-items in a q-itemset are sorted in lexicographical order. A quantitative sequence (q-itemset) is composed of multiple itemsets in an ordered arrangement, which is denoted as $s = <X_1, X_2, \ldots, X_d>$. The order of q-itemsets in a q-sequence, containing temporal order and spatial order, can represent the order of purchase, building order, among others.

Table 1 shows a quantitative sequential database. This database has five quantitative sequences and six items. Table 2 shows a utility table of the items that appear in Table 1. In Tables 1 and 2, (a), (b), (c), etc., represent the items; (a: 2) indicates that the purchased quantity of item a is 2 (q-item for short); [(a: 2) (c: 3)] indicates a set of items with a purchased quantity 2 of item a and purchased quantity 3 of item c (referred to as q-item set); and <[(a: 2)(c: 3)], [(e: 3)]> means that it is a sequence containing two q-itemsets [(a: 2)(c: 3)] and [(e: 3)] (q-sequence for short), where [(a: 2)(c: 3)] and [(e: 3))] have a sequential relationship in the sequence.

Take s_1 in Table 1 as an example to give the concrete explanations, apple(a) is purchased with cake(c) together respectively with the amounts of 2 and 3 (e.q., [(a:2)(c:3)]). After that, apple(a), bread(b) and cake(c) are purchased together respectively with the amounts of 3, 1, and 2 (e.q., [(a:3)(b:1)(c:2)]). In addition, apple(a), bread(b), and donuts(d) ae purchased together respectively with the amounts of 4, 5, and 4 (e.q., [(a:4)(b:5)(d:4)]). Finally, egg(e) is then purchased with the amount of 3 (e.q., [(e:3)]). Thus, it can be seen that four sequential orders are in s_1. First, the utility of an item i_r in a q-itemset X can be defined as follows.

Definition 1. *$u(i_r, X)$ is used to denote the utility of an item i_r in a q-itemset X, and is defined as follows:*

$$u(i_r, X) = q(i_r, X) \times pr(i_r), \tag{1}$$

Table 1. A quantitative sequence database

s_{id}	**sequence**
s_1	<[(a:2)(c:3)],[(a:3)(b:1)(c:2)],[(a:4)(b:5)(d:4)],[(e:3)]>
s_2	<[(a:1)(e:3)],[(a:5)(b:3)(d:2)],[(b:2)(c:1)(d:4)(e:3)]>
s_3	<[(e:2)],[(c:2)(d:3)],[(a:3)(e:3)],[(b:4)(d:5)]>
s_4	<[(b:2)(c:3)],[(a:5)(e:1)],[(b:4)(d:3)(e:5)]>
s_5	<[(a:4)(c:3)],[(a:2)(b:5)(c:2)(d:4)(e:3)]>

Table 2. A profit table

item	a	b	c	d	e	f
profit	5	3	4	2	1	6

where $q(i_r, X)$ is the quantity in a q-itemset X and $pr(i_r)$ is the profit of an item i_r.

Example 1. The utility of an item a in s_1 of Table 1 is calculated as: $u(a,[(a{:}2)(c{:}3)]) = q(a,[(a{:}2)(c{:}3)])\times pr(a) = 2 \times 5 = 10$

To calculate the utility of an itemset X (or q-itemset) in a q-sequence s, the following definition and an example are given below.

Definition 2. *$u(X, s)$ is used to denote the utility of a q-itemset in a q-sequence s, and is defined as follows:*

$$u(X, s) = \sum_{X \in s \wedge i_r \in X} u(i_r, X) \tag{2}$$

Example 2. The utility of a q-itemset $[(a{:}1)(e{:}3)]$ in q-sequence s_2 is calculated as: $u([(a{:}1)(e{:}3)],s_2) = 1 \times 5 + 3 \times 1 = 8$

Based on the above definitions, we can then calculate the utility of a q-sequence s in the database by the following definition.

Definition 3. *$u(s)$ is used to denote as the utility of a q-sequence in a quantitative sequential database D, and is defined as follows:*

$$u(s) = \sum_{s \in D \wedge X \in s} u(X, s) \tag{3}$$

Example 3. The utility of the q-sequence s_2 in Table 1 is calculated as: $u(s_2) = u([(a{:}1)(e{:}3)],s_2) + u([(a{:}5)(b{:}3)(d{:}2)],s_2) + u([(b{:}2)(c{:}1)(d{:}4)(e{:}3)],s_2) = 8 + 38 + 21 = 67$.

To calculate the utility of a quantitative sequential database D, the following definition and its example are given below.

Definition 4. *$u(D)$ is used to denote the utility of a quantitative sequential database D which is the sum of the utility of each q-sequence, and is defined as follows:*

$$u(D) = \sum_{s \in D} u(s) \tag{4}$$

Example 4. The utility of the quantitative sequential database D in Table 1 is calculated as: $u(D) = u(s_1) + u(s_2) + u(s_3) + u(4) + u(s_5) = 94 + 67 + 56 + 67 + 76 = 360$.

To show all the elements (or item/sets) of an itemset, the formal definition and the relevant example are given below.

Definition 5. *Given two itemsets, $x_a = [i_{a_1}, i_{a_2}, \ldots, i_{a_m}]$ and $x_b = [i_{b_1}, i_{b_2}, \ldots, i_{b_n}]$, where $i_{a_k} \in I(i \leq k \leq m)$ and $i_{b_{k'}} \in I(i \leq k' \leq n)$. If there exists positive integers $1 \leq j_1 \leq j_2 \leq \cdots \leq j_m \leq n$ such that $i_{a_1} = i_{b_{j_1}}$, $i_{a_2} = i_{b_{j_2}}$, $\ldots$, $i_{a_m} = i_{b_{j_m}}$, then x_b is said to contain x_a, which is denoted as $x_a \subseteq x_b$.*

Example 5. The itemset $[a, c]$ contains the itemsets $[a]$, $[c]$ and $[a, c]$.

To show whether an itemset is contained in a sequence, the formal definition and the example are given below to clearly show their relationships.

Definition 6. *Given two q-itemsets such as X_a and X_b, then q-itemset $X_a = [(i_{a_1}, q_{a_1})(i_{a_2}, q_{a_2}) \ldots (i_{a_m}, q_{a_m})]$ and a q-itemset $X_b = [(i_{b_1}, q_{b_1})(i_{b_2}, q_{b_2}) \ldots (i_{b_n}, q_{b_n})]$, where $i_{a_k} \in I(i \leq k \leq m)$ and $i_{b_{k'}} \in I(i \leq k' \leq n)$. If there exists positive integers $1 \leq j_1 \leq j_2 \leq \cdots \leq j_m \leq n$ such that $i_{a_1} = i_{b_{j_1}} \wedge q_{a_1} = q_{b_{j_1}}$, $i_{a_2} = i_{b_{j_2}} \wedge q_{a_2} = q_{b_{j_2}}$,$\ldots$,$i_{a_m} = i_{b_{j_m}} \wedge q_{a_m} = q_{b_{j_m}}$, then X_b is said to contain X_a, which is denoted as $X_a \subseteq X_b$.*

Example 6. The q-itemset $[(a{:}3)(c{:}2)]$ in q-sequence s_1 in Table 1 contains the q-itemset $[(a{:}3)]$, q-itemset $[(c{:}2)]$ and q-itemset $[(a{:}3)(c{:}2)]$.

To elaborate the relationships of a sequence to a sequence, the following definition with a simple example is given below.

Definition 7. *Given two sequences $s = \langle x_1, x_2, \ldots, x_m \rangle$ and $t = \langle y_1, y_2, \ldots, y_n \rangle$, where $x_i \subseteq I$ and $y_j \subseteq I$ are both itemsets, if there exists positive integers $1 \leq j_1 \leq j_2 \leq \cdots \leq j_m \leq n$ such that $x_1 \subseteq y_{j_1}$, $x_2 \subseteq y_{j_2}$, $\ldots$, $x_m \subseteq y_{j_m}$, then s is the subsequence of t, which is denoted as $s \subseteq t$.*

Example 7. A sequence $\langle [a,b]\ [a,c], [b,c] \rangle$ is the subsequence of the sequence $\langle [a,b], [a,b,c], [a,b], [b,c] \rangle$.

To handle the quantitative number of the items in the sequential database, the definition is then given below to show the relationship of a sequence and its sub-sequences.

Definition 8. *Given two q-sequences $s = \langle X_1, X_2, \ldots, X_m \rangle$ and $t = \langle Y_1, Y_2, \ldots, Y_n \rangle$, where X_i and Y_j are both q-itemsets, if there exist positive integers $1 \leq j_1 \leq j_2 \leq \cdots \leq j_m \leq n$ such that $X_1 \subseteq Y_{j_1}$, $X_2 \subseteq Y_{j_2}$, $\ldots$, $X_m \subseteq Y_{j_m}$, then s is the q-subsequence of t, which is denoted as $s \subseteq t$.*

Example 8. The q-sequences $\langle [(a{:}2)], [(b{:}1)(c{:}2)] \rangle$ and $\langle [(a{:}3)(c{:}2)], [(a{:}4)(d{:}4)], [(e{:}3)] \rangle$ are two q-subsequences of the q-sequence s_1 in Table 1.

To show the number of matches regarding the sub-sequences within a sequence, the following definition and its example are then given below.

Definition 9. *Given a q-sequence $s = \langle X_1, X_2, \ldots, X_n \rangle$ and a sequence $t = \langle x_1, x_2, \ldots, x_m \rangle$, if $n = m$ and the items in X_i are same as the items in x_i, where $1 \leq i \leq n$, then s is said to match t, which is denoted as $t \sim s$.*

Example 9. A sequence $\langle [a][a,b][a,d] \rangle$ matches the s_1 in Table 1. Note that two q-itemsets may be considered as different although they contain the same itemset because of the quantities and the position of a q-sequence. Therefore, it is possible that more than one q-subsequence of a q-sequence match a given sequence. The sequence $\langle [a] \rangle$ has three matches in s_1:$\langle [(a{:}2)] \rangle$, $\langle [(a{:}3)] \rangle$, and $\langle [(a{:}4)] \rangle$.

Definition 10. *A q-itemset containing k items is called k-q-itemset. A q-sequence containing k items is called k-q-sequence.*

Example 10. The q-sequence s_1 is a 9-q-sequence.

Definition 11. *$u(t, s)$ is used to denote as the utility of a sequence t in a q-sequence s, and is defined as follows:*

$$u(t, s) = max\{u(s_k) | t \sim s_k \wedge s_k \subseteq s\}, \tag{5}$$

where $\sim$ denotes the match relationship and $t \sim s_k$ represents that s_k the match of t.

Example 11. The utility of a sequence <[a],[b]> in the q-sequence s_1 of Table 1 is calculated as: u(<[a],[b]>,s_1) = max{u([a:2],[b:1],s_1),u([a:2],[b:5],s_1),u([a:3],[b:5],s_1)} = max{13,25,30}=30.

This example shows that a target sequence in HUSPM may have multiple utility values in a transaction, which is quite different from generic HUIM and ARM. Different evaluation criteria choose different utility values, and here the *maximum* value is used as the utility value of the target sequence in HUSPM.

Definition 12. *$u(t)$ is used to denote the utility of a sequence t in a quantitative sequence database D, and is defined as follows:*

$$u(t) = u(t) = \sum_{s \in D} \{u(t,s) | t \sim s_k \wedge s_k \subseteq s\} \tag{6}$$

Example 12. The utility of a sequence <[a],[b]> in Table 1 is calculated as: u(<[a],[b]>) = u(<[a],[b]>,s_1) + u(<[a],[b]>,s_2) + u(<[a],[b]>,s_3) + u(<[a],[b]>,s_4) + u(<[a],[b]>,s_5) = 30 + 31 + 27 + 37 + 35 = 160.

To handle multiple databases in the distributed and parallel environment, let D be a quantitative sequence database and D_1, D_2, ..., D_m are the partitions of D satisfied by D ={$D_1 \cup D_2 \cup \ldots \cup D_m$} and $\forall\{D_i, D_j\} \in D$, $D_i \cap D_j = \emptyset$. For example, the database D in Table 1 can be split into two partitions, D_1 and D_2, as Tables 3 and 4 show. Table 2 is also the profit table of these two quantitative databases.

Table 3. Quantitative sequence database D_1

s_{id}	**sequence**
s_1	<[(a:2)(c:3)],[(a:3)(b:1)(c:2)],[(a:4)(b:5)(d:4)],[(e:3)]>
s_2	<[(a:1)(e:3)],[(a:5)(b:3)(d:2)],[(b:2)(c:1)(d:4)(e:3)]>

Table 4. Quantitative sequence database D_2

s_{id}	**sequence**
s_3	<[(e:2)],[(c:2)(d:3)],[(a:3)(e:3)],[(b:4)(d:5)]>
s_4	<[(b:2)(c:3)],[(a:5)(e:1)],[(b:4)(d:3)(e:5)]>
s_5	<[(a:4)(c:3)],[(a:2)(b:5)(c:2)(d:4)(e:3)]>

Definition 13. *$u_L(t, D_i)$ is used to denote the utility of a sequence t in the partition D_i, called the local utility of a sequence in a partition, and is defined as follows:*

$$u_L(t, D_i) = \sum_{s_j \in D_i} u(t, s_j) \tag{7}$$

Example 13. The utility of the sequence <[a],[b]> in partition D_1 of Table 3 is calculated as: u(<[a],[b]>,D_1) = u(<[a],[b]>,s_1) + u(<[a],[b]>,s_2) = 30 + 31 = 61.

To find the utility of a sequence t in the partitions, the definition is given as follows.

Definition 14. *$u_G(t, D)$ is used to denote the utility of a sequence t in the partitions, called the global utility of a sequence in the sequence database D, and is defined as follows:*

$$u_G(t, D) = \sum_{D_i \in D} u_L(t, D_i) \tag{8}$$

Example 14. The utility of a sequence <[a],[b]> in the sequence database D of Table 1 is calculated as: u(<[a],[b]>,D) = u(<[a],[b]>,D_1) + u(<[a],[b]>,D_2) = 160.

Definition 15. *If the utility of the sequence t in the partition quantitative database D_i is not less than the user-defined minimum threshold, then it is called a local high-utility sequential pattern, and is defined as follows:*

$$u_L(t, D_i) \geq \delta \times u(D_i), \tag{9}$$

where δ is the minimum utility threshold given in percentage and $u(D_i)$ is the total utility of the partition D_i.

Example 15. The utility of the sequence <[a],[b]> in partition D_1 is u_L(<[a],[b]>,D_1) = 61, and the utility of partition D_1 is $u(D_i)$ = 161. If the minimum utility threshold is set to 0.3, then the sequence <[a],[b]> is a local high-utility sequential pattern in partition D_1 because $61 \geq 0.3 \times 161 = 48.3$.

Definition 16. *If the summed up utility values of a sequence t in the quantitative database D is not lower than the user-defined minimum threshold, then it is called a global high-utility sequential pattern, and is defined as follows:*

$$u_G(t, D) \geq \delta \times u(D), \tag{10}$$

where δ is a utility threshold given in percentage and $u(D)$ is the total utility of the all partitions.

Example 16. The utility of the sequence <[a],[b]> in the sequence database D is u_G(<[a],[b]>,D) = 160, and the utility of the sequence database D is $u(D)$ = 360. If the minimum utility threshold is set to 0.3, then the sequence <[a],[b]> is a global high-utility sequential pattern in the sequence database D because $160 \geq 0.3 \times 360 = 108$.

Problem Statement. Given a large-scale quantitative database D and a minimum utility threshold δ, the task of HUSPM using a distributed and parallel method for handling the large-scale dataset is to discover the complete set of sequences whose global utility is not less than $\delta \times u(D)$ by efficiently parallel mining the partition D_i of D.

4 DESIGNED MAPREDUCE MODELS AND ALGORITHMS

In this paper, we first develop a three-stage MapReduce framework for discovering HUSPs from large-scale databases, which is the first work to adopt a MapReduce model in HUSPM. Furthermore, two data structures respectively called `sidset` and `Utility-Linked List` are utilized here to keep the necessary information for the mining progress. Fig. 1 first shows an overview of the designed framework. The framework is divided into three phases which are **Identification**, **Local Mining**, **Integration**. Each MapReduce is then performed for each phase in the designed framework. The three MapReduce operations used in the designed framework are respectively representing the three phases and are described below.

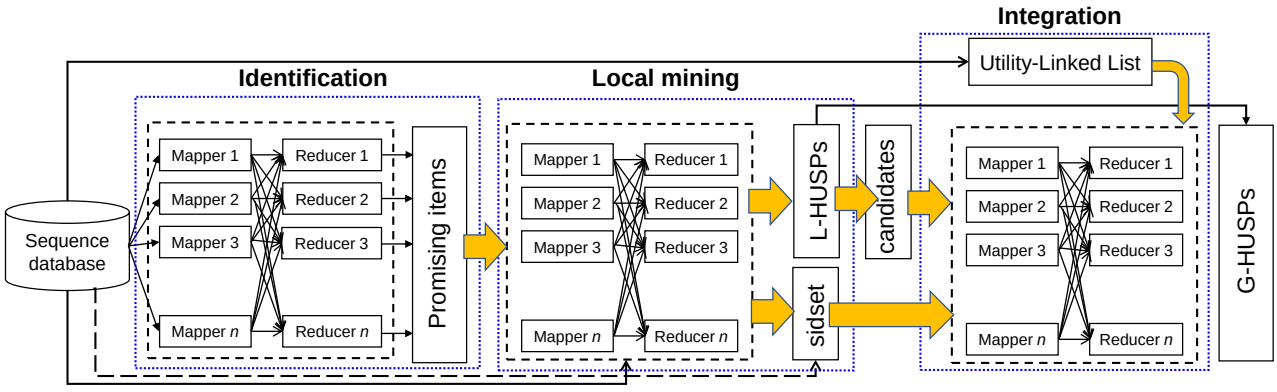

Fig. 1. An overview of the framework.

4.1 Identification

The first phase uses MapReduce to identify promising items that may be HUSPs along with their super-sequences. The unpromising items and their super-sequences are discarded and do not need to be considered according to the first designed property in the first phase. Details of Property 1 are described next.

Property 1. Considering that a quantitative sequential database is divided into multiple parts, if a pattern p is a HUSP, then it is a HUSP in at least one part.

Proof. Suppose a database D is divided into n parts $\{D_1, D_2, \ldots, D_n\}$, the total utilities of each part are $\{u(D_1), u(D_2), \ldots, u(D_n)\}$, the minimum utility threshold is δ, and the sequence p is the global high-utility sequential pattern (*GHUSP*) over the entire database. Let p be a global HUSP, the following formula can be established as:

$$u(p) \geq \delta \times \sum_{i=1}^{n} u(D_i) \tag{11}$$

The counter-evidence, $\{u_1, u_2, \ldots, u_n\}$ is used to denote the utility of the pattern p in each part, and the pattern p is not the HUSP in all parts, which means that $u_1 < \delta \times u(D_1)$, $u_2 < \delta \times u(D_2)$, $\ldots$, $u_n < \delta \times u(D_n)$. Then, $u(p) = \sum_{i=1}^{n} u_i < \delta \times \sum_{i=1}^{n} u(D_i)$ conflicts with the above formula. Therefore, it is proven that *Property 1* is correct. □

Based on this property of the designed three-stage MapReduce framework, we can say that Property 1 ensures the integrity of the mined results. Additionally, the search space is reduced significantly compared with the original search space used in the first phase. To handle the parallel and distributed system in the designed framework, the Local Sequence Weighted Utility (*LSWU*) and Global Sequence Weighted Utility (*GSWU*) of a sequence are respectively defined. Unlike generic ARM or SPM, HUSPM does not hold the downward closure property. The search space for HUSPM algorithms is thus very large without the downward closure property. Then, the sequence weighted utility (*SWU*) [42] is utilized to keep the downward closure property of the designed LSWU and GSWU. Details are given next.

Definition 17. *The $LSWU(t, D_i)$ is used to denote the local sequence weighted utility of a sequence t in partition D_i, and is defined as follows:*

$$LSWU(t, D_i) = \sum_{s \in D_i} \{u(s) | t \subseteq s\}. \tag{12}$$

Example 17. The local sequence weighted utility of a sequence <[a]> in partition D_1 is calculated as: $LSWU(<[a]>, D_1) = u(s_1) + u(s_2) = 94 + 67 = 141$.

Definition 18. *The $GSWU(t, D)$ is used to denote the global sequence weighted utility of a sequence t in database D, and is defined as follows:*

$$GSWU(t, D) = \sum_{D_i \in D} LSWU(t, D_i) \tag{13}$$

Example 18. The global sequence weighted utility of a sequence <[a]> in database D is calculated as: $GSWU(t, D) = LSWU(<[a]>, D_1) + LSWU(<[a]>, D_2) = 160$.

Based on the GSWU, the high global sequence weighted utility sequence (H-GSWUS) is defined below.

Definition 19. *A sequence t in a sequence database D is a high global sequence weighted utility sequence (H-GSWUS) if its GSWU value is no less than the minimum utility value, denoted as follows:*

$$H\text{-}GSWUS \leftarrow \{t | GSWU(t, D) \geq \delta \times u(D)\}, \tag{14}$$

where δ is the minimum utility threshold given in percentage and $u(D)$ is the total utility of the database.

According to the downward closure property used in the designed *LSWU* and *GSWU*, the second property (Property 2) as described next is used to extend the downward closure property for supersets of satisfied sequences.

Property 2. There is a sequence database D and two sequences t and t' that are satisfied with $t \subseteq t'$, and then $GSWU(t, D) \geq GSWU(t', D)$.

Proof. Since the *GSWU* is the summed up value of *LSWU* for all partitions in the database, and *LSWU* is based on the sequence-weighted utilization (*SWU*) model, thus the *LSWU* of a sequence is, to sum up all the *SWU* values in all partitions if a sequence s appears in the sequences. In general, if two sequences $t \subseteq t'$ hold, that is the length of t' is larger than or equal to t. Based on definitions 17 and 18, the $SWU(t) \geq SWU(t')$ holds; $LSWU(t) \geq LSWU(t')$ holds. Since *GSWU* is the summed up value of *LSWU* for all partitions in the database, we then can conclude that $GSWU(t, D) \geq GSWU(t', D)$ holds based on the downward closure property of *SWU* and *LSWU*. □

According to *Property 2*, if the *GSWU* value of a sequence t is not a *H-GSWUS*, then the sequence t and its super sequences cannot be HUSPs. We can safely prune the sequences whose *GSWU* value is less than $\delta \times u(D)$ without affecting the complete set of HUSPs from the database D. Thus, the designed algorithms for the first MapReduce in the identification stage are described below.

ALGORITHM 1: Designed Mapper of MapReduce-1

Input: A set of key-value pairs; the key is the sequence s_{id} and the value is the q-sequence information.
Output: A set of key-value pairs; the key is the item and the value is the sequence utility containing this item.

```
1 for each q-sequence s do
2     calculate u(s);
3     for each item i in s do
4         write a pair (i, u(s));
5     end
6 end
```

In Algorithm 1, each Mapper obtains a partition of the sequence database (Algorithm 1, line 1). Then, the key-value pair <*key*, *value*> for the item and sequence utility of a certain sequence which contains this item is output to the Reducer (Algorithm 1, lines 2-4). Based on this pair set, it is easy to measure the utility of an item i in a sequence s. Please note that the size of a given q-sequence should not be larger than a maximum size of the partition to be processed. The Mappers of the first MapReduce are designed and shown in Algorithm 2.

ALGORITHM 2: First Mappers

Input: A set of key-value pairs; the key is the sequence s_{id} and the value is the list of sequence utility for a certain sequences containing this item.
Output: A set of key-value pairs; the key is the item and the value is *LSWU* of the item in the partition.

```
1 for each key-value pair (i, l_u) do
2     set LSWU = 0;
3     for each u ∈ l_u do
4         LSWU := LSWU + u;
5     end
6     write a pair (i, LSWU);
7 end
```

In Algorithm 2, the Mapper nodes accumulate the value with the same item before it outputs the key-value pair list to Reducers (Algorithm 2, lines 1-5). Furthermore, the output value of Algorithm 2 is the *LSWU* value of the item in the partition (Algorithm 2, line 6). It can reduce the requirement of the communication cost and the time of transportation. Simultaneously, it can also reduce the workload of the Reducers. The reason is that before the Reducers are processed, the key-value pairs of the same item are assigned to the same Reducer, thus the communication between different Reducers for calculating the same item can be greatly reduced. Then, the Reducers calculate the *GSWU* value for each item (Algorithm 3, lines 1-5), and output the items and their *GSWU* whose *GSWU* values are no less than $\delta \times u(D)$ while the unpromising items are discarded (Algorithm 3, lines 6-8). The promising items are used in a later MapReduce process to build the search space for each HUSPM task happening in each working node. The Reducer is then shown in Algorithm 3.

ALGORITHM 3: First Reducers

Input: A set of key-value pairs; the key is the item and the value is a list of the *LSWU* for the item.
Output: A set of key-value pairs; the key is the item and the value is *GSWU* of the item in the entire database.

```
1 for each key-value pair (i, LSWUs) do
2     set GSWU := 0;
3     for each LSWU in LSWUs do
4         GSWU := GSWU + LSWU;
5     end
6     if GSWU ≥ δ × u(D) then
7         write a pair (i, GSWU);
8     end
9 end
```

Generally speaking, in the first stage, the input database is split into several partitions and each Mapper is fed with a partition. All the items within *H-GSWU-sequence* are found to ensure the completeness and correctness for the later mining progress of varied k-itemsets ($k \geq 2$); the unpromising items are pruned to efficiently reduce the search space for later progresses. The second local mining stage is then described below.

4.2 Local Mining

The second phase uses an existing HUSPM (i.e., HUS-Span [42]) algorithm to mine HUSPs in each partition, called the local HUSPs. Note here that the HUSP-Span can be replaced by other efficient memory-based HUSPM algorithms. Because the overall task of mining HUSPs on the entire database is fairly large, it is divided into small, partial, and multiple sets, and the same tasks are executed in parallel in each node. Due to the smaller amount of memory required, what was impossible for a single machine to perform is now possible, and the set of candidates containing all the HUSPs can be produced. At the same time, the candidate patterns have been calculated the utility in each node mining. In this progress, we then developed the `sidset` structure to speed up the checking process in the further third phase. The `sidset` is a compressed data structure that keeps the necessary information for the later progress. The definition of the designed `sidset` structure is then described below.

Definition 20. *The* `sidset` *is the horizontal structure, and it is composed by the form <s_{id} ,($pattern_1$, $utility_1$), ($pattern_2$, $utility_2$), . . . , ($pattern_n$, $utility_n$)>, where s_{id} represents a certain quantitative sequence, and {$patttern_1$,. . . , $pattern_n$} are contained by this quantitative sequence.*

Before the second MapReduce starts, a simple load balancing method on each node is utilized that assists to split the sequence data regarding their sizes into MapReduce tasks (Algorithm 4, lines 2-11). This task can

speed up the entire MapReduce process since the minimal workload for each MapReduce can be found and balanced. The idea of load balancing is that the HUS-Span [42] uses the matching and comparison mechanism to generate the promising sequences. In this step, the number of the generated task files should match with the number of mappers in the second MapReduce. The workload is calculated by measuring the number of promising items within a sequence (Algorithm 4, lines 2-8), and then assigning this sequence to the task file with minimal workload (Algorithm 4, lines 9-11). This process helps to equally distribute the computations to each node, thus the processing time can be reduced compared to the serialization progress. Details are described in Algorithm 4.

ALGORITHM 4: Generate tasks

Input: k, the number of data partitions; *items*, the promising items whose *GSWU* values are no less than $\delta \times u(D)$; D, the input sequence database.

Output: k task files

```
1  initialize the work load WL_i to 0 of each task files i ;
2  for each quantitative sequence q in D do
3      Num = 0 ;
4      for each item in q do
5          if item in items then
6              Num = Num + 1 ;
7          end
8      end
9      find task file i with the minimum work load ;
10     WL_i = WL_i + Num ;
11     assign the quantitative sequence q to task i ;
12 end
13 output the k task files;
```

For the Mapper progress of the second MapReduce in the second stage, the HUS-Span algorithm [42] is used to find a set of local high-utility sequential patterns in partition D_i whose utility is no less than $\delta \times u(D_i)$ (Algorithm 5, lines 2-11). Each Mapper outputs the local HUSPs as a pair of $(pattern, (s_{id}, utility))$ (Algorithm 5, lines 9-11). We also build the *Utility-chain* [23] for each promising items to speed up the search complexity (Algorithm 5, lines 3-6). This chain structure has better performance than the generic HUSPM algorithms thus the computational cost can be greatly reduced. Details are shown in Algorithm 5.

For the Reducer progress of the second MapReduce, the mapper task executes HUS-Span algorithm [42] to mine the HUSPs in this partition. The local HUSPs that have the same key are assigned to the same Reducer. Then, the partial total utility of a pattern can be summed (Algorithm 6, lines 2-4). Through this method, the global HUSPs whose partial utility sum is more than $\delta \times u(D)$ can be identified because the complete total utility of a pattern is no less than the value of the partial utility sum (Algorithm 6, line 5). Then, the global HUSPs are saved to the result file (Algorithm 6, line 6); otherwise, the Reducers change the form of the key-value pair and output the key-value pairs as $(s_{id}, (pattern, utility))$ for later use in generating the candidate set and the `sidset` (Algorithm 6, lines 8-10). Next, all the candidate patterns and `sidset` need to be generated after the second MapReduce stage is completed. This process is shown in Algorithm 6.

We note here that the utility values are their utilities in the q-sequence that is calculated in the second stage. By reducing repeated computation, the `sidset` structure can accelerate the calculation of the total utility of the candidates. For instance, if the quantitative sequence contains a candidate, then its utility can be obtained directly, and there is no need to calculate its utility again. The reason is to iteratively calculate the utility of the same item is costly; based on the `sidset`, this utility can be directly traced without further calculation.

ALGORITHM 5: Second Mappers

Input: A set of key-value pairs, the value is task file T;*items*, the promising items whose *GSWU* values are no less than $\delta \times u(D)$;D, the input sequence database.

Output: A set of key-value pair $((pattern, (s_{id}, utility)))$, the key *pattern* is the *LHUSP*, the value $(s_{id}, utility)$ is utility of this *pattern* in a transaction s_{id}.

```
initialize Utility-chain of each item in items ;
for each quantitative sequence q in D and q in task file do
    for each item in items and each item in q do
        calculate the utility and remaining utility of each matching that item is in q ;
        build Utility List of item in q ;
        add the Utility List of item to the Utility-chain item ;
    end
end
for each item in items do
    {pattern,{s_id,utility}} ← HUS-Span(item, Utility-chain of item) ;
    write pairs (pattern, (s_id, utility));
end
```

ALGORITHM 6: Second Reducers

Input: A set of key-value pairs, the key denoted *pattern* is a *LHUSP* and value is the list denoted L with pairs $(s_{id}, utility)$;$u(D)$, the total utility; δ, the minimum utility threshold

Output: A set of key-value pair $(s_{id}, (pattern, utility))$

```
set U=0 ;
for each pair (s_id, utility) do
    U += utility ;
end
if U ≥ δ × u(D) then
    save pattern to the result file ;
end
else
    write pairs (s_id, (pattern, utility))
end
```

Please note the current framework does not deal with the case when partitions size exceeds the memory size. An alternative solution is to use approximate solutions where only small parts of each partition may be handled. This considerably reduces the number of frequent patterns discovered.

4.3 Integration

In the third phase, by computing the global utility of each local HUSP using MapReduce, the candidates produced by each partition are checked to see if they are a high-utility sequential pattern. In this phase, the data structure `sidset` produced by the second phase is used to reduce the utility calculation of the patterns that have been calculated in each node mining during the second phase. Simultaneously, the `Utility-Linked List`, which is transformed and expanded by the sequence in the original database and records information about the original database and common information that needs to be calculated, is also used to accelerate the computation of the utility. The definition of the `Utility-Linked List` is then described below.

Definition 21. *The* `Utility-Linked List` *is a data structure based on the idea of "space-for-time" which is formed by the transformation and expansion of the q-sequence in the original database. It consists of two arrays of* `UP Information` *and* `Header_Table`. *A* `Header_Table` *is a collection of non-repeating items in a transaction including the item name and the location of each item that first appeared in the transaction.*

The developed `Utility-Linked List` records information about the original database and common information that needs to be calculated. Due to this complete structure, the complete information is then kept in the main memory. It increases the computational speed for calculating the utility of a sequence. As mentioned earlier, the target sequence may have multiple matches in a single transaction. Therefore, calculating the utility value of a sequence in a transaction requires finding all matches and then taking the maximum utility value. The `Utility-Linked List` records the next location of the project in the transaction; therefore, the algorithm does not need to scan the transaction multiple times. The maximum utility value of the sequence in the transaction can be calculated as long as the next position of the item is continuously searched. Table 5 is a `Utility-Linked List` converted from the *q*-sequence *s1* of Table 1.

Table 5. The Utility-Linked List of s_1

UP Information	<[(*a*, 10, 3) (*c*, 12, 5)], [(*a*, 15, 6) (*b*, 3, 7) (*c*, 8, -)], [(*a*, 20, -) (*b*, 15, -) (*d*, 8, -)], [*e*, 3, -]>
Header Table	(*a*, 1) (*b*, 4) (*c*, 2) (*d*, 8) (*e*, 9)

As an example taken from Table 5, the non-repeating items in the *q*-sequence s_1 have *a*, *b*, *c*, *d*, and *e*, and their first occurrences in the *q*-sequence s_1 are 1, 4, 2, 8, and 9, respectively. *UP Information* is an extension of a *q*-sequence in which each element consists of three parts: the item name, the project utility value, and the next occurrence of the item in the *q*-sequence. By taking the first element of the UP Information of Table 5 as an example, the utility value of *a* is 10, and the position where *a* appears next in the *q*-sequence s_1 is 3.

In the third MapReduce stage, given the set of candidate patterns and the data structure `sidset`, this phase calculates the global HUSP in the candidate set and checks whether it is a global HUSP or not. In this stage, the core and time-cost operations are used to calculate the utility of a candidate pattern in this *q*-sequence. There are two situations:

(1) when the utility of this candidate pattern has been calculated, and the `sidset` of the *q*-sequence can be queried and its utility value can be obtained directly without the need to be computed again.
(2) when the utility of the candidate pattern has not been calculated. In this case, it needs to be checked if it appears in a certain *q*-sequence. If it appears, the utility of this candidate needs to be calculated in the *q*-sequence.

We note here that the calculation of this operation is time-consuming because it needs to scan the *q*-sequence, and the pattern may have multiple matches in a *q*-sequence; therefore, the algorithm needs to scan multiple times to find the largest match as the utility value of the candidate pattern in this *q*-sequence. Thus to complete the mining task, the entire sequential database must be scanned several times. We designed this framework and also the developed `Utility-Linked List` to handle this limitation for the large-scale databases Thus, the three parts, Mapper, Combiner and Reducer of the third MapReduce are respectively shown in Algorithms 7, 8, and 9. In the Mapper stage, each Mapper first projects the *q*-sequence information into `Utility-Linked List` (Algorithm 7, line 1) and then calculates the local utility of all patterns in the candidate set (Algorithm 7, lines 2-10). If the pattern can be queried by the sequence id in the `sidset` (Algorithm 7, lines 3-4), then this means that the utility of the pattern in this sequence has been calculated in the second MapReduce phase and the Mapper outputs the pair $(pattern, utility)$ for the Reduce stage; if not, the utility of the pattern in this sequence needs to be calculated using the `Utility-Linked List` and then output it (Algorithm 7, lines 6-9). Using the data structure `sidset`

and the `Utility-Linked List` can save much time and accelerate the process of calculating the global utility in the sequence database. In Combiner and Reducer stage (Algorithms 8 and 9), these two stages are used to sum the utility of a pattern. Algorithm 8 first calculates the local utility of each partition, and Algorithm 9 sums up the global utility of the sequences from all partitions. If the global utility of a pattern is no less than $\geq \delta \times u(D)$ in the Reduce stage (Algorithm 9, lines 5-7), then it is the needed global high-utility sequential pattern and is output as the final results.

ALGORITHM 7: Third Mappers

Input: A set of key-value pairs; the key is the sequence id and the value is the q-sequence information.
Output: A set of key-value pairs; the key is the item and the value is the sequence utility for a contain sequences containing this item.

```
1  project the q-sequence into Utility-Link List L;
2  for each pattern in candidate set C do
3      if pattern p appears in sidset where s_id == key then
4          write a pair (pattern, utility);
5      end
6      else
7          calculate the utility of a pattern using Utility-Linked List L;
8          write a pair (pattern, utility);
9      end
10 end
```

ALGORITHM 8: Combiner of Third Mappers

Input: A set of key-value pairs; the key is a pattern and the value is te list of the utilities of the pattern denoted as l_u.
Output: A set of key-value pairs; the key is the pattern and the value is utility value of the pattern.

```
1  set local_utility := 0;
2  for each u in l_u do
3      local_utility := local_utility + u;
4  end
5  write a par (key, local_utility);
```

ALGORITHM 9: Third Reducers

Input: A set of key-value pairs; the key is a pattern and the value is the list of the utilities of the pattern denoted as l_u.
Output: A set of key-value pairs; the key is the pattern and the value is global utility value in the sequential database.

```
1  set global_utility := 0;
2  for each u in l_u do
3      global_utility := global_utility + u;
4  end
5  if global_utility ≥ δ × u(D) then
6      write a pair (key, global_utility);
7  end
```

5 EXPERIMENTAL RESULTS

Several experiments were conducted to evaluate the performance of the presented MapReduce framework in the Spark model. The designed three-stage MapReduce framework without any structure (i.e., `sidset` and `Utility-Linked List`) is named **M-HUSPM** in the experiments, and the designed three-stage MapReduce framework with both the `sidset` and the *Utility-Linked List* structures for the implementation is named **ML-HUSPM** in the experiments. The state-of-art algorithm HUS-Span [42] is also used as the serial algorithm for comparisons and evaluation. Each algorithm is then performed ten times for the evaluation. Experiments were performed with a local Spark cluster on a workstation having Intel Xenon CPU 2.10 GHz with 8 cores, 16 threads, 16 GB RAM, and 1.5 TB of disk storage. Spark-2.1.1 is installed over Ubuntu 20.04, 64 bit running on the workstation. Note that the data structure is stored using HDFS (Hadoop Distributed File System) storage system. To save the shared structure, we used the Hadoop sequence file, which is a binary file format containing all of the data in the shared structures presented by <*key*, *value*> pairs in a serialized form. Four real-life datasets [34] were used in the experiments. The characteristics of the four original datasets are shown in Table 6. The parameters of the datasets are indicated using the following four attributes: $|\boldsymbol{D}|$ states the total number of sequences; $|I|$ is the number of distinct items; $\boldsymbol{C}$ is the average number of itemsets per sequence, and ***MaxLen*** states the maximum number of items per sequence. In real cases, there is no large-scale datasets for performing the designed model for efficiency evaluation, thus, the original datasets in Table 6 are then enlarged that is the original size is multiplied by various numbers (i.e., 1, 20, 50, 100, 200, 500, 1, 000, 2, 000, 5, 000, 10, 000). The sizes of the conducted large-scale datasets are also illustrated in Table 7.

Table 6. Characteristics of experimental datasets

Dataset	$\|\boldsymbol{D}\|$	$\|\boldsymbol{I}\|$	$\boldsymbol{C}$	***MaxLen***
SIGN	730	267	52.0	94
Leviathan	5,834	9,025	33.8	100
MSNBC	31,790	17	13.3	100
BMS	59,601	497	2.5	267

Table 7. Data size in GB

Dataset	**1**	**20**	**50**	**100**	**200**	**500**	**1,000**	**2,000**	**5,000**	**10,000**
SIGN	0.0002	0.004	0.0110	0.02	0.04	0.10	0.20	0.40	1.12	2.25
Leviathan	0.001	0.02	0.05	0.10	0.20	0.50	1.00	2.00	5.00	10.00
MSNBC	0.002	0.04	0.10	0.20	0.40	1.00	2.00	4.00	10.00	20.00
BMS	0.001	0.02	0.05	0.10	0.20	0.50	1.00	2.00	5.00	10.00

5.1 Runtime Performance

The designed three-stage MapReduce framework was proposed to handle the problem of large-scale datasets. This section describes the runtime performance of the state-of-the-art serially HUS-Span, M-HUSPM, and ML-HUSPM on several large-scale datasets. Fig. 2 shows the execution time of the three algorithms on the four datasets. The results of runtime regarding maximum (Max.), minimum (Min.), and average (Avg.) are then illustrated in Table 8.

From the results, it can be seen that the HUS-Span while running on a single machine cannot handle much data. For example, in the Sign and Leviathan datasets, the HUS-Span obtains lower runtime than that of the M-HUSPM and ML-HUSPM when the database size is less than 100 times the original ones. However, when the database size increases to 200 times the original dataset, the generic and serial HUS-Span cannot obtain any of

Table 8. Comparisons of Max., Min., and Avg. in terms of runtime (sec.)

		M-HUSPM			ML-HUSPM			HUS-Span		
Dataset	**Times (Size)**	**Max.**	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**
(a) SIGN (δ=0.05)	20 (0.004GB)	8	5	7	5	3	4	8	5	7
	500 (0.01GB)	21	16	18	11	14	13	-	-	-
	10,000 (2.25GB)	4,275	4,047	4,118	2,982	2,895	2,934	-	-	-
(b) Leviathan (δ=0.13)	20 (0.02GB)	14	12	13	9	8	7	19	15	17
	500 (0.5GB)	340	321	335	224	205	211	-	-	-
	10,000 (10GB)	6,879	6,674	6,709	4,875	4,679	4,708	-	-	-
(c) MSNBC (δ=0.08)	20 (0.04GB)	248	201	224	174	142	167	341	328	335
	500 (1GB)	5,475	4,975	5,214	4,124	3,905	4,074	-	-	-
	10,000 (20GB)	11,475	10,248	11,005	8,475	8,005	8,248	-	-	-
(d) BMS (δ=0.04)	20 (0.02GB)	25	18	20	11	5	8	36	44	40
	500 (0.5GB)	457	419	443	214	201	207	-	-	-
	10,000 (10GB)	9,421	9,142	9,415	4,452	4,005	4,214	-	-	-

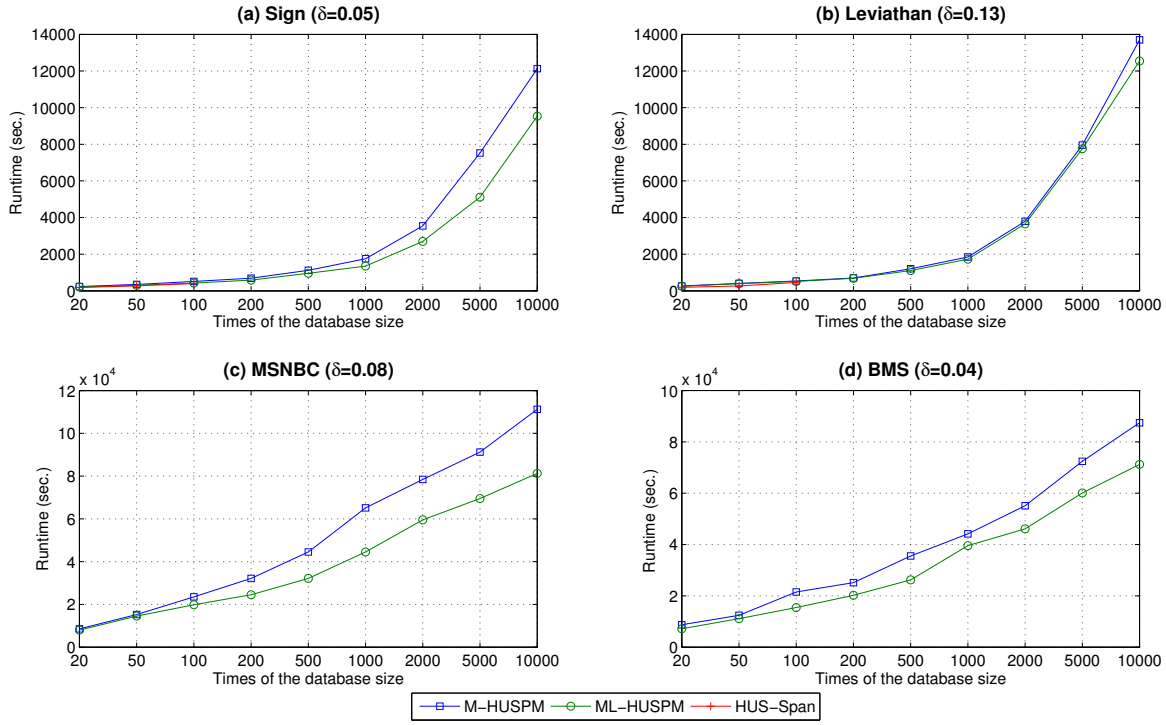

Fig. 2. The runtime on varied big datasets.

the results due to the memory leakage. This is reasonable since the serial HUS-Span can only be performed on a small dataset, which is not able to handle a very large-scale dataset. However, the designed two algorithms can be performed on four datasets in terms of varied database size from 20 to 10,000 times of the original ones.

Table 9. Comparisons of Max., Min., and Avg. in terms of memory usage (MB)

		M-HUSPM			ML-HUSPM			HUS-Span		
Dataset	**Times (Size)**	**Max.**	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**
(a) SIGN (δ=0.05)	20 (0.004GB)	2	5	3	2	2	2	471	512	492
	500 (0.01GB)	41	38	40	38	33	35	-	-	-
	10,000 (2.25GB)	915	821	854	884	801	842	-	-	-
(b) Leviathan (δ=0.13)	20 (0.02GB)	4	2	3	3	1	2	514	485	506
	500 (0.5GB)	83	80	82	66	61	64	-	-	-
	10,000 (10GB)	1,348	1,249	1,278	1,107	1,067	1,085	-	-	-
(c) MSNBC (δ=0.08)	20 (0.04GB)	7	5	6	5	5	5	854	757	804
	500 (1GB)	75	66	72	42	48	45	-	-	-
	10,000 (20GB)	1,970	1,824	1,970	1,523	1,329	1,482	-	-	-
(d) BMS (δ=0.04)	20 (0.02GB)	12	8	10	8	5	7	751	706	733
	500 (0.5GB)	52	48	50	32	27	29	-	-	-
	10,000 (10GB)	1,005	904	957	804	777	792	-	-	-

It is also obvious to see that the designed ML-HUSPM obtains better performance than that of the M-HUSPM, which can be seen from Fig. 2(a), Fig. 2(c), and Fig. 2(d). Thanks to the developed `sidset` and `Utility-Linked List` structures, both of them can be used to greatly reduce the computational cost while mining the required HUSPs from a large-scale dataset. The next section will provide the evaluation in terms of memory usage of three compared algorithms.

5.2 Memory Usage

This section examines the maximum memory usage of each working node on the Spark cluster compared to the maximum memory usage of a single machine. Fig. 3 shows the result of the maximum memory usage of these three algorithms. The results of memory usage regarding maximum (Max.), minimum (Min.), and average (Avg.) are then illustrated in Table 9. In addition, Table 10 presents the memory usage of the `Utility-Linked List` of the proposed framework.

As shown in Fig. 3, the memory usage of the HUS-Span increased as the size of the dataset increased because the HUS-Span is memory-based and needs to load all data into memory before mining. For example in Fig. 3(a), the required memory of HUS-Span is about 1, 200 MB when the size of the database is 20 times the original one. As the database size increases to 50 times of the original one, the HUS-Span needs about 2, 300 MB, and when the size increases to 100 times of the original one, the HUS-Span requires about 3, 000 MB. This situation also applies to Fig. 3(b). However, as the dataset increased in size, the HUS-Span algorithm may result in an out of memory error especially when the size of the conducted datasets is over than 200 times of the original ones that can be observed both in Fig. 3(a) and Fig. 3(b). In addition, the HUS-Span can only be performed while the size of the original database is under 20 times of the original databases (i.e., MSNBC and BMS), which can be discovered from Fig. 3(c) and Fig. 3(d). When the size increases more than 50 times of the original databases, the HUS-Span cannot be well performed and causes the memory leakage issue. The designed M-HUSPM and ML-HUSPM obtain stable results in terms of memory usage, and even the ML-HUSPM requires the extra `sidset` and `Utility-Linked List` structures to keep more information for speeding up the computational progress, but those two structures can also be helpful to reduce the multiple database scans (the generated candidates required memory for further processing). Furthermore, Table 10 shows that the percentage of the memory usage of the utility linked list

Table 10. Percentage of memory usage of the Utility-Linked lists of the proposed framework.

Dataset	Times (Size)	Max.	Min.	Avg.
(a) SIGN (δ=0.05)	20 (0.004GB)	0.65	0.32	0.45
	500 (0.01GB)	1.12	1.06	1.08
	10,000 (2.25GB)	25.67	22.45	23.57
(b) Leviathan (δ=0.13)	20 (0.02GB)	1.06	0.98	1.03
	500 (0.5GB)	3.27	3.06	3.15
	10,000 (10GB)	25.98	22.36	23.45
(c) MSNBC (δ=0.08)	20 (0.04GB)	1.59	1.45	1.51
	500 (1GB)	7.12	6.93	7.05
	10,000 (20GB)	42.56	40.65	41.20
(d) BMS (δ=0.04)	20 (0.02GB)	2.04	1.57	1.86
	500 (0.5GB)	6.71	6.33	6.53
	10,000 (10GB)	33.27	30.91	31.95

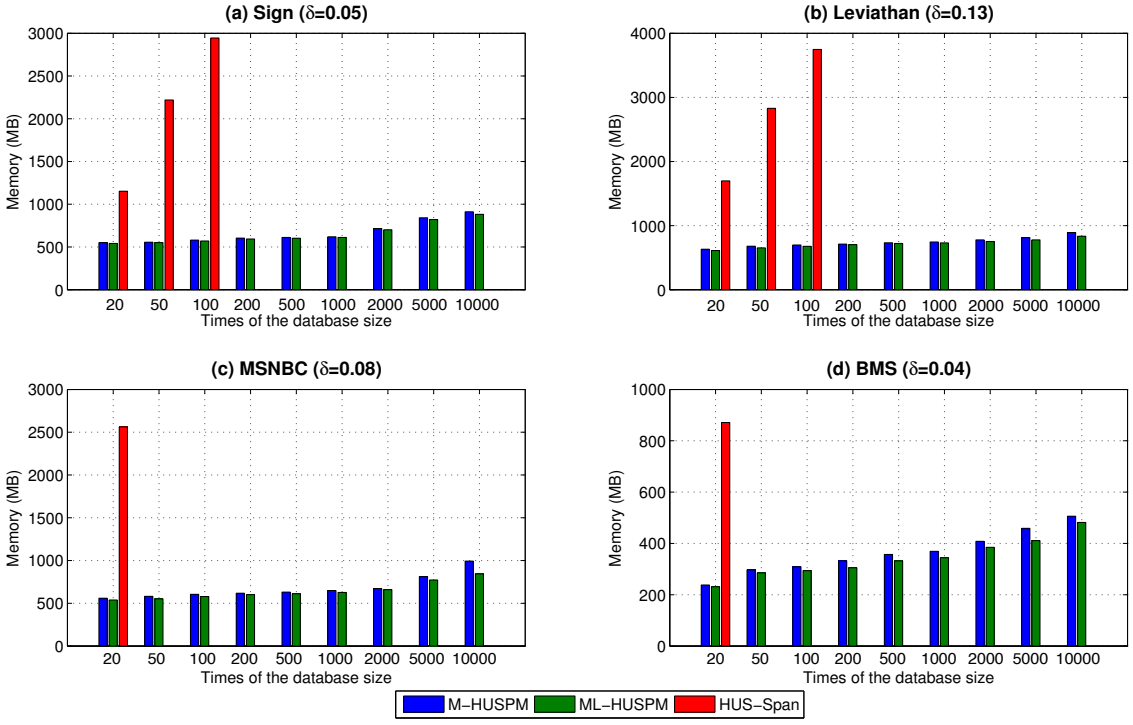

Fig. 3. The memory usage of compared algorithms.

structures does not exceed 43% even for big databases. Thus, the memory usage of the ML-HUSPM can still be minimized compared to the M-HUSPM. Moreover, it can also be observed that the parameters $|I|$, C, and $MaxLen$ does not seriously affect the results of the compared algorithms but the database size $|D|$ since the HUSP-Span cannot be performed for the MSNBC and BMS datasets while the database size is over than 50 times of the

original ones. This observation also showed that the designed MapReduce models have good capability to handle the large-scale datasets, and it does not matter about varied parameters of the datasets.

5.3 Speedup performance

In this section, the Spark cluster was run on one server with multiple virtual machines. These virtual machines shared the CPU, IO, and main memory. Note that the main memory is limited to one server. We ran the designed algorithms using 2, 4, 8, 16, and 32 nodes. The work nodes increased we increase the number of virtual machines. The results are shown in Fig. 4.

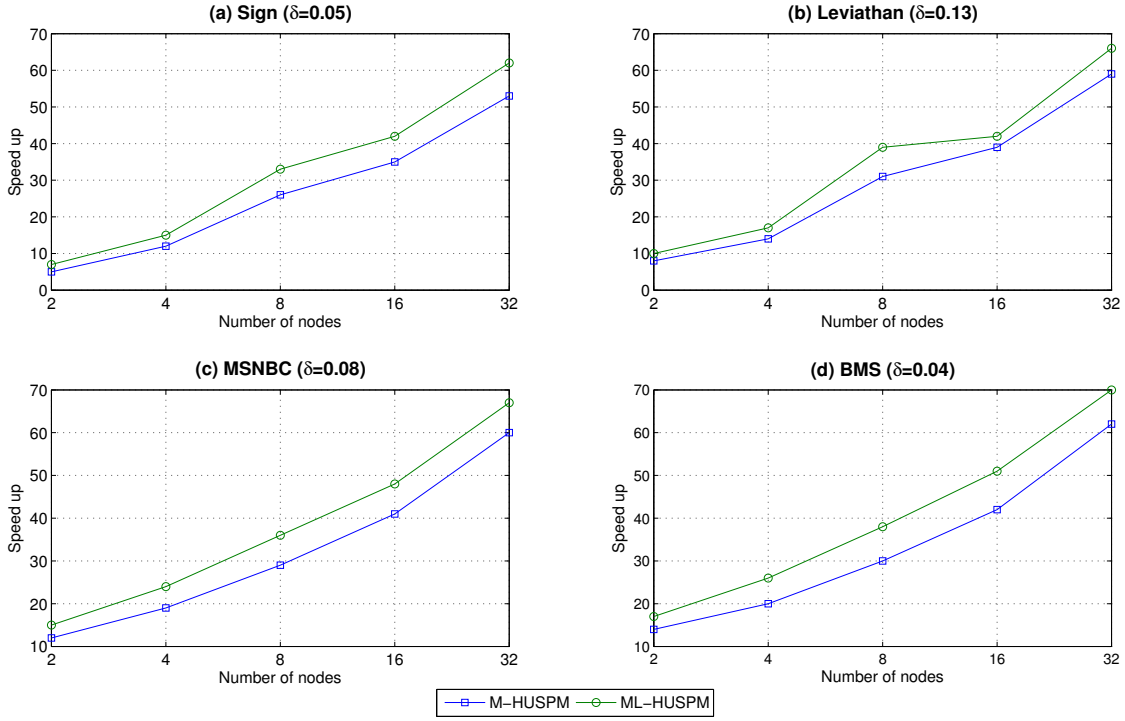

Fig. 4. The runtime on varied number of nodes.

From the results shown in Fig. 4, it is obvious to see that when the nodes were increased, the acceleration effect was very obvious. The runtime of these two distributed models has linearly sped up along with the increases in the number of nodes in the distributed system. Thus, with the increasing number of nodes in the distributed system, the performance can be increased. Thanks to the developed two structures, the ML-HUSPM always obtains better performance than that of the M-HUSPM.

5.4 Scalability

The last experiments aim to test the scalability of the proposed framework on large-scale databases regarding the number of distributed nodes in the MapReduce system. Several tests have been carried out by varying the number of nodes, and data size in GB. Fig. 5 presents the runtime in seconds, Fig. 6 shows the memory consumption, and Fig. 7 discusses the speedup of both M-HUSPM, and ML-HUSPM using 40GB of the duplicated BMS data. Note that each result is the standard deviation of 10 samples.

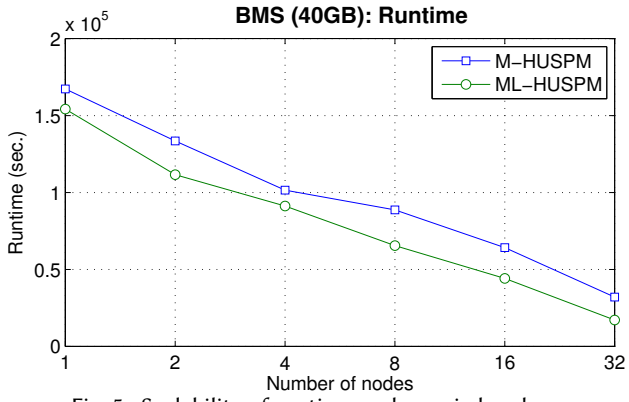

Fig. 5. Scalability of runtime under varied nodes.

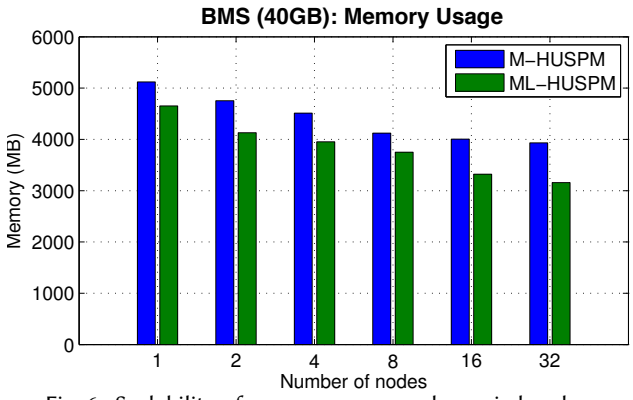

Fig. 6. Scalability of memory usage under varied nodes.

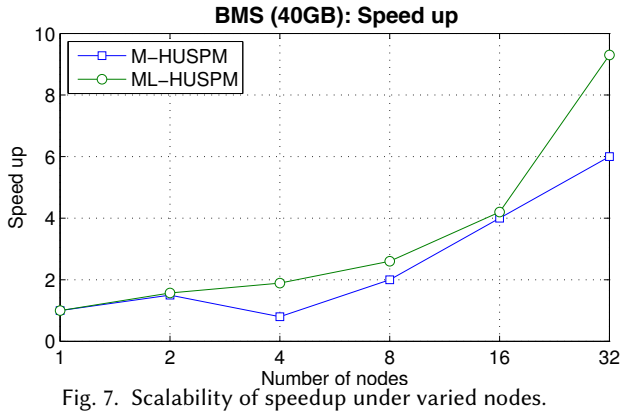

Fig. 7. Scalability of speedup under varied nodes.

With varying the number of nodes from 1, 2, 4, 8, 18 and 32, the scalability of both approaches increased. Since the serial HUS-Span cannot handle the large-scale datasets, thus it could not be compared with the designed algorithms. Generally, the runtime of these two distributed MapReduce frameworks decreases as the number of

the work nodes increases. For example, the runtime is decreased from more than 15, 000 seconds to less than 4, 000 seconds, the memory consumption is decreased from more than 5, 000 MB to less than 3, 500 MB, and the speedup is increased from less than 2 to more than 8. Thus, the runtime and speed-up performances are greatly improved and the memory usage stably decreases along with the number of distributed nodes. In addition, the ML-HUSPM outperforms the M-HUSPM, whatever the scenario used in the experiment. In summary, the designed models obtained good performance in the large-scale dataset and the as the increasing number of distributed nodes, the scalability of the designed algorithms can thus be efficiently achieved.

6 CONCLUSION AND FUTURE WORK

A three-stage MapReduce framework is designed in this paper to handle the high-utility sequential pattern mining in large-scale databases. To speed up mining performance, two data structures called `sidset` and `Utility-Linked List` are applied in the designed model. Moreover, two properties are then developed to hold the correctness and completeness of the discovered patterns. From the conducted results in the experiments, the designed model showed better performance compared to the traditional HUSPM models in terms of runtime, memory usage, and scalability, particularly in large-scale databases. In future works, the designed model can be extended to the other constraint-based approaches, i.e., top-k, maximal or closed high-utility sequential pattern mining. Moreover, the evolutionary computation models can also be discussed and utilized in the designed model to improve the effectiveness and efficiency of the mining progress.

ACKNOWLEDGMENT

This work is partially supported by Western Norway University of Applied Sciences, Bergen, Norway, and by NSF under grants III-1763325, III-1909323, III-2106758, and SaTC-1930941.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. N. Swami, Database mining:A performance perspective, *IEEE Transactions on Knowledge and Data Engineering*, vol. 5(6), pp. 914–925, 1993

[2] R. Agrawal and R. Srikant, Fast algorithms for mining association rules in large databases, *The International Conference on Very Large Data Bases*, pp. 487–499, 1994.

[3] R. Agrawal and R. Srikant, Mining sequential patterns, *The International Conference on Data Engineering*, pp. 3–14, 1995.

[4] C. F. Ahmed, S. K. Tanbeer, and B. S. Jeong, Mining high utility web access sequences in dynamic web log data, *ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, pp. 76–81, 2010.

[5] C. F. Ahmed, S. K. Tanbeer, and B. S. Jeong, A novel approach for mining high-utility sequential patterns in sequence databases, *Electronics and Telecommunications Research Institute*, vol. 32(5), pp. 676–686, 2010.

[6] O. K. Alkan and P. Karagoz, CRoM and HuspExt: Improving efficiency of high utility sequential pattern Extraction, *IEEE Transactions on Knowledge and Data Engineering*, vol. 27(10), pp. 2645–2657, 2015.

[7] U. Ahmed, J. C. W. Lin, G. Srivastava, R. Yasin, and Y. Djenouri, An evolutionary model to mine high expected utility patterns from uncertain databases, *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2020.

[8] M. Chen, J. Han, and P. S. Yu, Data mining: An overview from a database perspective, *IEEE Transactions Knowledge and Data Engineering*, vol. 8(6), pp. 866–883, 1996.

[9] R. Chan, Q. Yang, and Y.-D. Shen, Mining high utility itemsets, *IEEE International Conference on Data Mining*, pp. 19–26, 2003.

[10] Y. Chen and A. An, Approximate parallel high utility itemset mining, *Big Data Research*, vol. 6, pp. 26–42, 2016.

[11] J. Dean and S. Ghemawat, MapReduce: simplified data processing on large clusters, *Communications of the ACM*, vol. 51(1), pp. 107–113, 2008.

[12] K. C. Duong, M. Bamha, A. Giacometti, D. Li, A. Soulet, and C. Vrain, MapFIM+: memory aware parallelized frequent itemset mining in very large datasets, *Transactions on Large-Scale Data and Knowledge-Centered Systems*, vol. 39, pp. 200–225, 2018.

[13] J. Ge, Y. Xia, and J. Wang, Mining uncertain sequential patterns in iterative mapReduce, *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 243–254, 2015.

[14] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, and P. S. Yu, A survey of parallel sequential pattern mining, *ACM Transactions on Knowledge Discovery from Data*, vol. 13(3), pp. 1–34, 2019.

[15] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, V. Tseng, and P. S. Yu, A survey of utility-oriented pattern mining, *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2019.

[16] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu, Freespan: frequent pattern-projected sequential pattern mining, *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 355–359, 2000.

[17] J. Han, J. Pei, Y. Yin, and R. Mao, Mining frequent patterns without candidate generation: A frequent-pattern tree approach, *Data Mining and Knowledge Discovery*, vol. 8(1), pp. 53–87, 2004.

[18] H. Kim, U. Yun, Y. Baek, H. Kim, H. Nam, J. C. W. Lin, and P. Fournier-Viger, Damped sliding based utility oriented pattern mining over stream data, *Knowledge-Based Systems*, vol. 213, pp. 106653, 2021.

[19] H. Li, Y. Wang, D. Zhang, M. Zhang and, E. Y. Chang, Pfp: parallel fp-growth for query recommendation, *ACM conference on Recommender systems*, pp. 107–114, 2008.

[20] Y. Liu, W. Liao, and A. N. Choudhary, A two-phase algorithm for fast discovery of high utility itemsets, *Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pp. 689–695, 2005.

[21] J. C. W. Lin, T. Hong, and W. Lu, An effective tree structure for mining high utility itemsets, *Expert Systems with Applications*, vol. 38(6), pp. 7419-–7424, 2011.

[22] M. Liu and J. Qu, Mining high utility itemsets without candidate generation, *ACM International Conference on Information and Knowledge Management*, pp. 55–64, 2012.

[23] J. Liu, K. Wang, and B. C. M. Fung, Direct discovery of high utility itemsets without candidate generation, *IEEE International Conference on Data Mining*, pp. 984–989, 2012.

[24] M. Y. Lin, P. Y. Lee and, S. C. Hsueh, Apriori-based frequent itemset mining algorithms on MapReduce, *The International Conference on Ubiquitous Information Management and Communication*, pp. 1–8, 2012.

[25] G. C. Lan, T. P. Hong, H. C. Huang, and S. T. Pan, Mining high fuzzy utility sequential patterns, *The International Conference on Fuzzy Theory and Its Applications*, pp. 420–424, 2013.

[26] Y. C. Lin, C. W. Wu, and V. S. Tseng, Mining high utility itemsets in big data, *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 649–661, 2015

[27] J. Liu, K. Wang, and B. C. M. Fung, Mining high utility patterns in one phase without generating candidates, *IEEE Transactions on Knowledge and Data Engineering*, vol. 28(5), pp. 1245–1257, 2016.

[28] J. C. W. Lin, W. Gan, P. Fournier-Viger, T. P. Hong, and V. S. Tseng, Efficient algorithms for mining high-utility itemsets in uncertain databases, *Knowledge-Based Systems*, vol. 96, pp. 171–187, 2016.

[29] J. C. W. Lin, L. Yang, P. Fournier-Viger, and T. P. Hong, Mining of skyline patterns by considering both frequent and utility constraints, *Engineering Applications of Artificial Intelligence*, vol. 77, pp. 229–238, 2019.

[30] S. Moens, E. Aksehirli, and B. Goethals, Frequent itemset mining for big data, *IEEE International Conference on Big Data*, pp. 111–118, 2013.

[31] T. Mai, L. T. T. Nguyen, B. Vo, U. Yun, and T. P. Hong, Efficient algorithm for mining non-redundant high-utility association rules, *Sensors*, vol. 20(4), 1078, 2020.

[32] H. Nam, U. Yun, E. Yoon, and J. C. W. Lin, Efficient approach of recent high utility stream pattern mining with indexed list structure and pruning strategy considering arrival times of transactions, *Information Sciences*, vol. 529, pp. 1–27, 2020.

[33] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, Prefixspan: Mining sequential patterns by prefix projected growth, *The International Conference on Data Engineering*, pp. 215–224, 2001.

[34] P. Fournier-Viger, J. C. W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, The SPMF open-source data mining library version 2, *The European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 36–40, 2016.

[35] R. Srikant and R. Agrawal, Mining sequential patterns: Generalizations and performance improvements, *The International Conference on Extending Database Technology*, pp. 3–17, 1996.

[36] B. E. Shie, J. F. Hsiao, V. S. Tseng, and P. Yu, Mining high utility mobile sequential patterns in mobile commerce environments, *The International Conference on Database Systems for Advanced Applications*, pp. 224–238, 2011.

[37] B. E. Shie, J. H. Cheng, K. T. Chuang and V. S. Tseng, A one-phase method for mining high utility mobile sequential patterns in mobile commerce environments, *The International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems*, pp. 616–626, 2012.

[38] G. Srivastava, J. C. W. Lin, M. Pirouz, Y. Li, and U. Yun, A pre-large weighted-fusion system of sensed high-utility patterns, *IEEE Sensors Journal*, 2020.

[39] S. Sumalatha and R. B. V. Subramanyam, Distributed mining of high utility time interval sequential patterns using mapreduce approach, *Expert System with applications*, vol. 141, pp. 112967, 2020.

[40] V. S. Tseng, B. Shie, C. Wu, and P. S. Yu, Efficient algorithms for mining high utility itemsets from transactional databases, *IEEE Transactions Knowledge and Data Engineering*, vol. 25(8), pp. 1772-–1786, 2013.

[41] B. Vo, L. T. T. Nguyen, T. D. D. Nguyen, P. Fournier-Viger, and U. Yun, A multi-core approach to efficiently mining high-utility itemsets in dynamic profit databases, *IEEE Access*, vol. 8, pp. 85890–85899, 2020.

[42] J. Wang, J. Huang, and Y. Chen, On efficiently mining high utility sequential patterns, *Knowledge and Information Systems*, vol. 49(2), pp. 597-–627, 2016

[43] J. M. T. Wu, J. C. W. Lin, and A. Tamrakar, High-utility itemset mining with effective pruning strategies, *ACM Transactions on Knowledge Discovery from Data*, vol. 13(6), pp. 1–22, 2019.

[44] J. M. T. Wu, G. Srivastava, M. Wei, U. Yun, and J. C. W. Lin, Fuzzy high-utility pattern mining in parallel and distributed Hadoop framework, *Information Sciences*, vol. 553, pp. 31–48, 2021.

[45] H. Yao, H. J. Hamilton, and C. J. Butz, A foundational approach to mining itemset utilities from databases, *SIAM International Conference on Data Mining*, pp. 482-–486, 2004.

[46] J. Yin, Z. Zheng, and L. Cao, USpan: an efficient algorithm for mining high utility sequential patterns, *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 660–668, 2012.

[47] U. Yun, H. Nam, J. Kim, H. Kim, Y. Baek, J. Lee, E. Yoon, T. Truong, B. Vo, and W. Pedrycz, Efficient transaction deleting approach of pre-large based high utility pattern mining in dynamic databases, *Future Generation Computer Systems*, vol. 103, pp. 58–78, 2020.

[48] J. Yin, Z. Zheng, L. Cao, Y. Song, and W. Wei, Efficiently mining top-*k* high utility sequential patterns, *IEEE International Conference on Data Mining*, pp. 1259-–1264, 2013.

[49] L. Zhou, Y. Liu, J. Wang, and Y. Shi, Utility-based web path traversal pattern mining, *IEEE International Conference on Data Mining*, pp. 373–380, 2007.

[50] S. Zida, P. Fournier-Viger, J. C. W. Lin, C. W. Wu, and V. S. Tseng, EFIM: a fast and memory efficient algorithm for high-utility itemset mining *Knowledge and Information Systems*, vol. 51(2), pp. 595–625, 2017.

[51] M. Zihayat, H. Davoudi, and A. An, Mining significant high utility gene regulation sequential patterns, *BMC Systems Biology 2017, 11(Suppl 6):109*, vol. 11(109), pp. 1–14, 2017.