Distributed and Parallel High Utility Sequential Pattern Mining

Morteza Zihayat†, Zane Zhenhua Hu‡, Aijun An† and Yonggang Hu‡
†*Department of Electrical Engineering and Computer Science, York University, Toronto, Canada*
‡*Platform Computing, IBM, Toronto, Canada*
zihayatm@cse.yorku.ca, zanezhenhuahu@gmail.com, aan@cse.yorku.ca, yhu@ca.ibm.com

***Abstract*—The problem of mining high utility sequential patterns (HUSP) has been studied recently. Existing solutions are mostly memory-based, which assume that data can fit into the main memory of a computer. However, with advent of big data, such an assumption does not hold any longer. Hence, existing algorithms are not applicable to the big data environments, where data are often distributed and too large to be dealt with by a single machine. In this paper, we propose a new framework for mining HUSPs in big data. A distributed and parallel algorithm called *BigHUSP* is proposed to discover HUSPs efficiently. At its heart, BigHUSP uses multiple MapReduce-like steps to process data in parallel. We also propose a number of pruning strategies to minimize search space in a distributed environment, and thus decrease computational and communication costs, while still maintaining correctness. Our experiments with real life and large synthetic datasets validate the effectiveness of BigHUSP for mining HUSPs from large sequence datasets.**

***Keywords*-High Utility Sequential Pattern Mining, Distributed Algorithms, Big Data**

I. Introduction

Nowadays, huge volumes of data are produced in the form of sequences [1]. *Frequent sequential pattern mining* is an important task in data mining to discover sequences of itemsets that frequently appear in a dataset of sequences [2]. In [3], the authors showed that such approaches may not be sufficiently practical for industrial needs. This is due to the fact that many patterns returned by sequential pattern mining are not particularly related to a business need so that business people do not know which patterns are truly interesting and actionable for their business. In view of this, *high utility sequential pattern (HUSP) mining* was introduced [4] to extract valuable and useful sequential patterns from data by considering a business objective such as *profit*, *user's interest*, *cost*, etc. A sequence is a high utility sequential pattern (HUSP) if its utility, defined based on the business objective (e.g., *profit*), in a dataset is no less than a *minimum utility threshold*.

Although a few preliminary works have been conducted on HUSP mining, the existing studies (e.g., [5], [4]) are memory-based, which assume that data can fit in main memory of a single machine and do not consider the real-life applications that involve big data. Below are three examples drawn from *retail business*, *news portal* and *bioinformatics*, where data are too huge to fit into the main memory of a single machine and mining HUSPs from such big data can play an important role in business. *1) Mining profitable shopping behaviour in retail business:* In 2012 Walmart generated more than 2.5 petabytes of data relating to more than 1 million customer transactions every hour [6]. From such a huge dataset, finding the most profitable shopping behaviour is a critical business need. Such patterns are important in making business decisions for maximizing revenue or minimizing marketing or inventory costs. These patterns may not be discovered by traditional approaches since they discover patterns based on frequency than the profit of a pattern. *2) Mining user reading behaviour in a news portal:* In January 2015, the total number of news portal visitors in top-10 news portals varies from 130 million to 50 million [7]. From such big datasets, modeling users' reading behavior is a major way to obtain a deep insight into the users. Reading behaviour patterns can be used to build a news recommendation and can also improve the portal design and e-business strategies. However, traditional frequency-based approaches may not discover such patterns since they do not take the user's interest into account. *3) Mining disease-related gene sequences in biosequence datasets:* The size of a single sequenced human genome is approximately 200 gigabytes [8]. Identifying potential gene regulations that occur in a period of time with respect to a disease is important for biologists. Such patterns provide important information on the relative levels of expression of thousands of genes among samples. However, the frequency alone may not be informative enough to discover such sequences. For example, some genes are more important than others in causing a particular disease and some genes are more effective than others in fighting diseases.

While much work has been conducted on big data analytics [9], *mining high utility sequential patterns* from big data has received little attention. When dealing with a considerably large number of sequences (e.g., the above examples) whose information may not be entirely loaded into main memory, most existing HUSP mining algorithms cannot efficiently complete the mining process. Thus, existing algorithms are not suitable for handling big data.

Mining HUSPs from big data is not an easy task due to the following challenges. First, with the exponential growth of data in different domains, it is impossible or prohibitively costly to execute HUSP mining algorithms

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

on a single machine. Developing a parallel and distributed algorithm is the key to solving the problem. However, in order to implement a parallel algorithm, we need to address several issues such as how to decompose the search space, how to minimize communication overhead and how to deal with scalability problems. Second, a HUSP mining method needs to compute the utility of a candidate pattern over the entire set of input sequences in a sequence database. In a distributed platform, if the input sequences are distributed over various worker nodes, the local utility of a sequence in the partition at a worker node is not much useful for deciding whether the given pattern is a HUSP or not. Hence, we need to design a mechanism to aggregate the local utility of a pattern in various nodes into a global value so that we can calculate the utility of a pattern efficiently. Third, high utility sequential pattern analysis in big data faces the critical combinatorial explosion of search space caused by sequencing among sequence elements. Thus, pruning search space without losing HUSPs is critical for efficient mining of HUSPs. However, pruning search space in HUSP mining is more difficult than that in frequent sequential pattern mining because the *downward closure property* does not hold for the utility of sequences. That is, the utility of a sequence may be higher than, equal to, or lower than its super-sequences and sub-sequences [5], [4]. Thus, many search space pruning techniques that rely on the downward closure property cannot be directly used for mining HUSPs.

Motivated by the above challenges, we propose a parallel and distributed high utility sequential pattern mining algorithm called *BigHUSP* to mine HUSPs from big data. To the best of our knowledge, this topic has not been addressed so far. At a high-level, BigHUSP is designed and developed based on the *Apache Spark* [10] platform and takes advantage of several merit properties of Spark such as distributed in-memory processing, fault recovery and high scalability. We also propose a number of novel strategies to effectively prune the search space and unnecessary intermediate patterns in a distributed manner, which reduce computational and communication costs drastically. We prove that the proposed pruning strategies do not miss any HUSPs. Moreover, we conduct extensive experiments to evaluate the performance of the proposed algorithm. Experimental results verify that BigHUSP significantly outperforms baseline methods and efficiently mines HUSPs from big data.

The rest of the paper is organized as follows. Section II summarizes the related work. Section III presents preliminaries and the problem statement. The propose framework is discussed in Section IV. We report our experimental results in Section V. Finally, Section VI concludes the paper.

II. Related Work

High utility pattern mining has been studied recently [5], [4], [11]. The concept of HUSP mining was first proposed by Ahmed et al [5]. They proposed two algorithms, called UL and US, for mining HUSPs. UL is a level-wise candidate generation-and-testing algorithm and US is a pattern growth method. Shie et al. [12] proposed a framework for mining HUSPs in a mobile environment. The framework can only handle sequences with a single item in each sequence element. Ahmed et al. proposed efficient algorithms for mining *high utility access sequences* from web log data [13], which also only considered single-item sequences. Yin et al. [4] proposed the USpan algorithm for mining HUSPs. In this study, a lexicographic tree was used to extract the complete set of high utility sequential patterns and designed mechanisms for expanding the tree with two pruning strategies. Most recently, Alkan et al [14] proposed a tighter upper bound called *Cumulated Rest of Match (CRoM)* to prune search space before candidate pattern generation. They also proposed an algorithm called *HuspExt* to find HUSP by calculating the utilities of the child patterns based on that of parents. Although these algorithms are efficient for mining HUSPs from a centralized database using a single machine, they have not been parallelized for handling big data and also their proposed pruning strategies are not applicable to a distributed environment. Lin et al [15] proposed a method to mine high utility itemsets in big data, where the sequential ordering of itemsets is not considered. The addition of ordering information makes the pattern mining problem fundamentally different and much more challenging than mining high utility itemsets.

Many distributed algorithms have been proposed to mine frequent sequential patterns. Most of these approaches partition data into many small batches and mine the patterns in parallel with multiple machines [16], [17], [18]. Although these algorithms work in a distributed manner, each machine has to share data through communication with one another, thus suffering from a high mining overhead. On the other hand, some researchers designed algorithms on the cloud computing environment [19], [20]. However, most of the existing methods require numerous rounds of MapReduces jobs, thus cause the load unbalancing problem and suffer from high cost of MapReduce initialization. In addition, all these methods are for finding frequent sequential patterns and do not consider a business objective to discover patterns. Hence, some useful infrequent patterns with high utility may be missed.

To the best of our knowledge, existing methods for mining HUSPs do not address the issues of mining HUSPs in big data, and previous methods in big data do not find high utility sequential patterns. So far, no study has been conducted to learn high utility sequential patterns in big data, which is more challenging than finding frequent sequences due to the fact that the sequence utility does not satisfy the downward closure property.

III. Preliminaries and Problem Statement

A. Distributed Platform

Apache Spark was proposed as a framework to support iterative algorithms efficiently [10]. The Spark engine runs in a variety of environments like *Hadoop*[1], *Mesos clusters*[2] and *IBM Platform Conductor for Spark*[3] and it has been used in a wide range of data processing applications. The main key concept in Spark is the *resilient distributed dataset (RDD)*. RDD enables us to save great efforts for fitting into the MapReduce framework and also improves the processing performance. In this paper, we use Spark on top of IBM Platform Conductor that is an enterprise-grade, multi-tenant resource manager. IBM Platform Conductor allows organizations to run multiple instances of Spark frameworks simultaneously on a shared infrastructure for the best time to results and resource utilization through its efficient resource scheduling.

B. Problem Statement

Let $I^* = \{I_1, I_2, \cdots, I_N\}$ be a set of items. A sequence S is an ordered list of itemsets $\langle X_1, X_2, \cdots, X_Z\rangle$, where Z is the size of S. The length of S is defined as $\sum_{i=1}^{Z} |X_i|$. An *L-sequence* is a sequence of length L. A **sequence database** D consists of a set of sequences $\{S_1, S_2, ..., S_K\}$, in which each sequence S_r has a unique sequence identifier r and consists of an ordered list of itemsets $\langle IS_{d_1}, IS_{d_2}, ..., IS_{d_n}\rangle$, where each itemset IS_{d_i} has a unique global identifier d_i. An itemset IS_d in the sequence S_r is also denoted as S_r^d.

Definition 1: (**Super-sequence and sub-sequence**) Sequence $\alpha = \langle X_1, X_2, ..., X_i\rangle$ is a *sub-sequence* of sequence $\beta = \langle X_1^{'}, X_2^{'}, ..., X_j^{'}\rangle$ $(i \leq j)$ or equivalently β is a *super-sequence* of α if there exist integers $1 \leq e_1 < e_2 < ...e_i \leq j$ such that $X_1 \subseteq X_{e_1}^{'}, X_2 \subseteq X_{e_2}^{'}, ..., X_i \subseteq X_{e_i}^{'}$ (denoted as $\alpha \preceq \beta$).

Definition 2: (**External utility and internal utility**) Each item $I \in I^*$ is associated with a positive number $p(I)$, called the *external utility* (e.g., price/unit profit). In addition, each item I in itemset X_d of sequence S_r (i.e., S_r^d) has a positive number $q(I, S_r^d)$, called its internal utility (e.g., quantity) of I in X_d or S_r^d.

Definition 3: (**Utility of an item in an itemset of a sequence** S_r) The utility of an item I in an itemset X_d of a sequence S_r is defined as $u(I, S_r^d) = f_u(p(I), q(I, S_r^d))$, where f_u is the function for calculating utility of item I based on internal and external utilities. For simplicity, without loss of generality, we define the utility function as $f_u(p(I), q(I, S_r^d)) = p(I) \cdot q(I, S_r^d)$.

Definition 4: (**Utility of an itemset in an itemset of a sequence** S_r) Given itemset X, the utility of X in the itemset X_d of the sequence S_r where $X \subseteq X_d$, is defined as $u(X, S_r^d) = \sum_{I \in X} u(I, S_r^d)$.

Definition 5: (**Occurrence of a sequence** α **in a sequence** S_r) Given a sequence $S_r = \langle S_r^1, S_r^2, ..., S_r^n\rangle$ and a sequence $\alpha = \langle X_1, X_2, ..., X_Z\rangle$ where S_r^i and X_i are itemsets, α occurs in S_r iff there exist integers $1 \leq e_1 < e_2 < ... < e_Z \leq n$ such that $X_1 \subseteq S_r^{e_1}, X_2 \subseteq S_r^{e_2}, ..., X_Z \subseteq S_r^{e_Z}$. The ordered list of itemsets $\langle S_r^{e_1}, S_r^{e_2}, ..., S_r^{e_Z}\rangle$ is called an *occurrence of* α *in* S_r. The set of all occurrences of α in S_r is denoted as $OccSet(\alpha, S_r)$.

Definition 6: (**Utility of a sequence** α **in a sequence** S_r) Let $\tilde{o} = \langle S_r^{e_1}, S_r^{e_2}, ..., S_r^{e_Z}\rangle$ be an occurrence of $\alpha = \langle X_1, X_2, ..., X_Z\rangle$ in the sequence S_r. The utility of α w.r.t. $\tilde{o}$ is defined as $su(\alpha, \tilde{o}) = \sum_{i=1}^{Z} u(X_i, S_r^{e_i})$. The utility of α in S_r is defined as $su(\alpha, S_r) = \max\{su(\alpha, \tilde{o}) \mid \tilde{o} \in OccSet(\alpha, S_r)\}$.

Figure 1(a) shows a sequence database with five sequences. The utility of itemset $\{ac\}$ in S_2^2 is $u(\{ac\}, S_2^2) = u(a, S_2^2) + u(c, S_2^2) = 4 \times 2 + 1 \times 1 = 9$. Given $\alpha = \{a\}\{c\}$, the set of all occurrences of the sequence α in S_1 is $OccSet(\alpha, S_1) = \{\tilde{o_1} : \langle S_1^1, S_1^2\rangle, \tilde{o_2} : \langle S_1^1, S_1^3\rangle\}$, hence the utility of α in S_1 is $su(\alpha, S_1) = \max\{su(\alpha, \tilde{o_1}), su(\alpha, \tilde{o_2})\} = \{5, 7\} = 7$.

Let D be a **sequence database** and $D_1, D_2, ..., D_m$ are partitions of D such that $D = \{D_1 \cup D_2 \cup ... \cup D_m\}$ and $\forall\{D_i, D_j\} \in D, D_i \cap D_j = \emptyset$. We have the following definitions:

Definition 7: (**Local utility of a sequence** α **in a partition** D_i) The local utility of a sequence α in the partition D_i is defined as $su_L(\alpha, D_i) = \sum_{S_r \in D_i} su(\alpha, S_r)$.

Definition 8: (**Global utility of a sequence** α **in a sequence database** D) The global utility of a sequence α in D is defined and denoted as $su_G(\alpha, D) = \sum_{D_i \subseteq D} su_L(\alpha, D_i)$.

Accordingly, the **total utility of a partition** D_i is defined as $U_{D_i} = \sum_{S_r \in D_i} su(S_r, S_r)$. The **total utility of a sequence database** D is defined as $U_D = \sum_{D_i \subseteq D} U_{D_i}$.

Definition 9: (**Local High Utility Sequential Pattern (L-HUSP)**) Given a utility threshold δ in percentage, a sequence α is a *local high utility sequential pattern* in the partition D_i, iff $su_L(\alpha, D_i) \geq \delta \cdot U_{D_i}$.

Definition 10: (**Global High Utility Sequential Pattern (G-HUSP)**) Given a utility threshold δ in percentage, a sequence α is a *global high utility sequential pattern* in sequence database D, iff $su_G(\alpha, D) \geq \delta \cdot U_D$.

Problem Statement. Given a minimum utility threshold δ (in percentage) and a sequence database D, our problem of distributed and parallel high utility sequential pattern mining from big data D is to discover the complete set of subsequences of itemsets whose global utility in D is no less than $\delta \cdot U_D$ by parallel mining of partitions of D over a

[1] http://wiki.apache.org/hadoop

[2] http://mesos.apache.org

[3] https://www.ibm.com/developerworks/servicemanagement/tc/pcs/index.html

PID	SID	Sequence Data
D_1	S_1	S_1^1:{(a,2)(b,3)(c,2)}; S_1^2:{(b,1)(c,1)(d,1)}; S_1^3:{(c,3)(d,1)}
	S_2	S_2^1:{(b,4)}; S_2^2:{(a,4)(b,5)(c,1)}
D_2	S_3	S_3^1:{(b,3)(d,1)}; S_3^2:{(a,4)(b,5)(c,1)}; S_3^3:{(a,2)(c,3)}
D_3	S_4	S_4^1:{(a,2)(b,5)(e,2)}
	S_5	S_5^1:{(c,4)}

Items	S_3^1	S_3^2	S_3^3
a	⟨0,44⟩	⟨8,23⟩	⟨4,3⟩
b	⟨9,38⟩	⟨15,8⟩	⟨0,3⟩
c	⟨0,35⟩	⟨1,7⟩	⟨3,0⟩
d	⟨4,34⟩	⟨0,7⟩	⟨0,0⟩

Item	a	b	c	d	e
Profit	2	3	1	4	3

(a) **(b)**

Figure 1. (a) An example of sequence database, (b) Utility Matrix (UM) of S_3

cluster of computers.

C. Global Sequence-Weighted Utility

It has been proved that the utility of a sequence does not have the *downward closure property* [5], [4]. Thus, the search space for HUSP mining cannot be pruned as it is done in the frequent sequential pattern mining framework. Inspired by the proposed overestimate utility model in [21], we first propose an overestimate utility model, called $GSWU$, to identify and filter out items that cannot be part of a HUSP. Then, we prove that $GSWU$ has the downward closure property. Eventually, we propose our first pruning strategy used in our method using $GSWU$.

Definition 11: Given a partition of sequences D_i, the Local Sequence-Weighted Utility (**LSWU**) of a sequence α in D_i, denoted as $LSWU(\alpha, D_i)$, is defined as the sum of the utilities of all the sequences containing α in D_i: $LSWU(\alpha, D_i) = \sum_{S_r \in D_i \wedge \alpha \preceq S_r} su(S_r, S_r)$ where $\alpha \preceq S$ means α is a subsequence of S.

Accordingly, the *Global Sequence-Weighted Utility* (**GSWU**) of a sequence α in database D is defined as: $GSWU(\alpha, D) = \sum_{D_i \subseteq D} LSWU(\alpha, D_i)$.

Definition 12: (**High GSWU Sequence**). Given a minimum utility threshold δ and sequence database D, a sequence α is called high GSWU sequence iff $GSWU(\alpha, D) \geq \delta \cdot U_D$.

Below we prove that the maximum utility of any sequence containing α will be no more than $GSWU(\alpha, D)$.

Lemma 1: Given a sequence database D and two sequences α and β such that $\alpha \preceq \beta$, $GSWU(\alpha, D) \geq GSWU(\beta, D)$.

Proof: Given D as set of partitions $D_1, D_2, ..., D_m$, we prove that $LSWU(\alpha, D_i) \geq LSWU(\beta, D_i)$ where $D_i \subseteq D$. The proof can be easily extended to sequence database D. Let D_i^α be the set of sequences containing α in D_i and D_i^β be the set of sequences containing β in D_i. Since $\alpha \preceq \beta$, β cannot be presented in any sequence where α does not exist. Consequently, $D_i^\beta \subseteq D_i^\alpha$. Based on Definition 11, $LSWU(\alpha, D_i) \geq LSWU(\beta, D_i)$. ■

According to Lemma 1 if α is not a high GSWU, there will be no HUSP containing α. For example in Figure 1, given pattern $\alpha = \langle\{b\}\{a\}\rangle$ and $minUtil = 90$, according to Definition 11, $GSWU(\alpha, D) = 36 + 44 + 0 = 80$. According to the above lemma, since $GSWU(\alpha) < 90$, hence $su(\alpha, D) \leq GSWU(\alpha, D) < 90$, α is not a HUSP. Moreover, all of its super sequences such as $\beta = \{b\}\{a\}\{a\}$, are not HUSPs. Because, $su(\beta, D) \leq GSWU(\beta, D)$ and $GSWU(\beta, D) = 44 < GSWU(\alpha) < 90$.

IV. Mining High Utility Sequential Patterns from Big Data

In this section, we propose an efficient algorithm called *BigHUSP* for discovering HUSPs in big data. BigHUSP takes a sequence database D and a minimum utility threshold δ as inputs and outputs the complete set of G-HUSPs in D. Figure 2 shows an overview of BigHUSP. In the *initialization phase*, BigHUSP uses a MapReduce step to compute the GSWU value for each item and identify *unpromising items*, that is, items whose GSWU is less than the minimum utility threshold (which cannot form a HUSP). By pruning the unpromising items from the matrices, the search space becomes significantly smaller. In the *L-HUSP mining phase*, BigHUSP employs an existing HUSP mining algorithm on each partition of data to mine local HUSPs. Later, G-HUSPs can be found by calculating the utility of each L-HUSP over all the partitions. Since the number of L-HUSPs can be large, before calculating the utility of each L-HUSP, in *the PG-HUSP (potential G-HUSP) generation phase*, L-HUSPs which cannot become a G-HUSP are pruned using a novel overestimate utility model and the rest of them are considered as potential G-HUSPs. Finally, the global utility of each PG-HUSP is calculated and all the G-HUSPs are returned in the *G-HUSP mining phase*.

A. Initialization

In this phase, the input sequence database is split into several partitions and each mapper is fed with a partition. A mapper converts a sequence into an efficient data structure called *utility matrix* to maintain some utility information

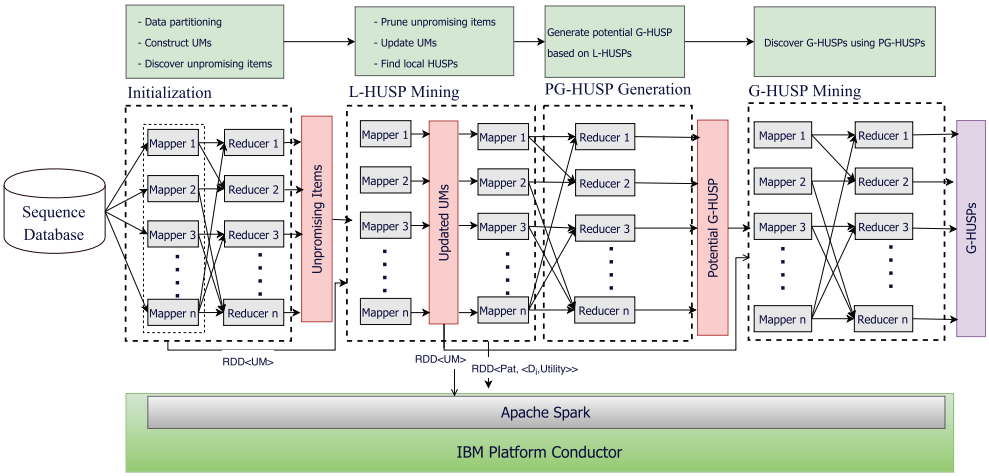

Figure 2. Overview of BigHUSP

so that BigHUSP can fast retrieve the utility values in later phases and does not need to process the original data anymore. In this phase, we also find items which cannot form a HUSP (i.e., *unpromising items*). Later, BigHUSP prunes these items to reduce the search space efficiently. This phase contains two main stages:

1) Map stage: given a partition, each mapper constructs a **utility matrix (UM)** for each input sequence in the partition. *UM* is an efficient data structure to keep the required information to mine L-HUSP from the partition. This representation makes the mining process faster since the utility values can be calculated more efficiently. Each element in the matrix consists of an item and a few tuples, one per itemset in the sequence where a tuple contains two values: (1) the utility of item in the itemset, and (2) the *remaining utility* of the rest of items in the sequence w.r.t the item. The *remaining utility* values are used in L-HUSP mining phase to prune the search space. Figure 1(b) shows the utility matrix of sequence S_3 in the partition D_2 presented in Figure 1(a). For example, given item b in the second itemset of S_3, $u(b, S_3^2) = 5 \times 3 = 15$ and its remaining utility is $\{u(c, S_3^2) + u(a, S_3^3) + u(c, S_3^3)\} = 8$. Once the UMs are constructed, they are maintained in RDD for later use.

Not every item in the database can form a HUSP. Hence, we use $LSWU$ and $GSWU$ to find items which cannot form a HUSP (i.e., *unpromising items*). Each mapper calculates $LSWU$ value of each item in a partition and outputs a key-value pair $\langle$ *item*, LSWU (*item*,D_i)$\rangle$, where the value is the $LSWU$ of $item$ in partition D_i.

2) Reduce stage: the output with the same key (i.e., item) is sent to the same reducer. A reducer calculates $GSWU$ of each item by summing up the *LSWU* values of the same item. After *GSWU* values are calculated, each reducer returns the items whose *GSWU* value is less than the minimum utility threshold as *unpromising items*. The results of reducers are collected and maintained in RDD to update UMs in the next phase.

B. L-HUSP Mining

In this phase, all the unpromising items are pruned from the matrices in each partition to reduce the search space. Then, since it is not possible to build the search space over the entire data to find G-HUSPs, this phase builds the local search spaces and finds local HUSPs. Later G-HUSPs are discovered from the L-HUSPs found in this phase. It consists of two consecutive map transformations as follows.

Map transformation 1: given the original UMs and the set of unpromising items obtained from the initialization phase, each mapper prunes the unpromising items from each UM. The updated UMs are output by the mappers and stored in RDD.

Given set of updated UMs, there are two general approaches to mine local HUSPs. The first approach is to discover L-HUSPs by iteratively executing MapReduce rounds as follows. Initially, a variable k is set to zero. In the k-th iteration, all the L-HUSPs of length k are discovered by performing a MapReduce pass. In the map task, the candidates with length *(k + 1)* are generated using the k-sequence obtained from the previous MapReduce iteration, and in the reducer, the true utility of generated candidates are calculated and *(k+1)-sequences* are discovered. However, this approach suffers from excessive communications overheads during the mining phase between MapReduce tasks. The most challenging problem to mine G-HUSPs in a distributed environment is how to avoid the excessive communication overheads among nodes and yet discover the complete set of G-HUSPs. The second approach is to find all patterns in a partition that has a non-zero utility value in the map phase of the mining, and then in the reduce phase, it decides whether

a pattern is a G-HUSP by aggregating its utility computed in all partitions from different computing nodes. However, due to the combinatorial number of possible sequences, this is an infeasible approach especially for big data. Instead, we design the second map transformation to find only L-HUSPs in each partition.

Map transformation 2: given a minimum utility threshold δ, a partition D_i as a set of updated *UMs* and total utility U_{D_i}, BigHUSP applies *USpan* [4] to find a set of L-HUSPs whose utility is no less than $\delta \times U_{D_i}$.

Each mapper outputs the local HUSPs as a pair of $\langle Pat, \langle D_i, utility \rangle \rangle$ where D_i is the partition id and $utility$ is the utility of pattern Pat in D_i. The pairs are stored in RDD for later use.

Below we first prove that, given a non-zero minimum utility threshold, if a pattern is not an L-HUSP in any of the partitions, it will not be a G-HUSP.

Lemma 2: Given a sequence database D and m non-overlapped partitions $\{D_1, D_2, ..., D_m\}$ and the minimum utility threshold δ, a sequence pattern α **is not** a G-HUSP, if $\forall D_i \subseteq D, su_L(\alpha, D_i) < \delta \cdot U_{D_i}$.

Proof: We prove the lemma by contradiction. Assume that α is not an L-HUSP, but it is a G-HUSP.

According to Definition 9, we have, $\forall D_i, su_L(\alpha, D_i) \leq \delta \times U_{D_i}$. Consequently,

$$\sum_{D_i \subseteq D} su_L(\alpha, D_i) \leq \delta \times \sum_{D_i \subseteq D} U_{D_i} \quad (1)$$

On the other hand, based on Definition 10, α is G-HUSP iff $su(\alpha, D) \geq \delta \cdot U_D$. Since we divide D into m partitions $D_1, D_2, ..., D_m$ so that $\forall D_i, D_j \in D, D_i \cap D_j = \emptyset$, we have: $\sum_{D_i \subseteq D} su_L(\alpha, D_) \geq \delta \times \sum_{D_i \subseteq D} U_{D_i}$.

Hence it is a contradiction with equation 1. ■

According to this lemma, by mining L-HUSPs, we do not miss any G-HUSPs.

C. PG-HUSP Generation

In order to find G-HUSPs, we need to calculate the global utility of each L-HUSP found in the previous phase. Since the number of L-HUSP can be large, we first define potential G-HUSP (i.e., PG-HUSP) and prune all L-HUSPs which are not PG-HUSPs.

Definition 13: **(Maximum utility of a sequence α in a partition D_i)** Given a minimum utility threshold δ and the partition D_i, the *maximum utility of* α in D_i is defined as follows:

$$MAS(\alpha, D_i) = \begin{cases} su_L(\alpha, D_i), & \text{if } su_L(\alpha, D_i) \geq \delta \cdot U_{D_i} \\ \delta \cdot U_{D_i}, & \text{otherwise} \end{cases}$$

Definition 14: **Maximum utility of a sequence α in a sequence database** D is defined as follows: $MAS(\alpha, D) = \sum_{D_i \subseteq D} MAS(\alpha, D_i)$

Algorithm 1 *Utility Calculation*

Input: $curNode$, $UMSet_{D_i}$, α, idx, $CType$
Output: $\langle \alpha, su(\alpha, D_i) \rangle$

```
1: if α is the pattern presented by curNode then
2:    return ⟨α, curNode.utility⟩
3: end if
4: Create node N as a child of curNode
5: if CType[idx] = 'I' then
6:    N.Pattern ← curNode.Pattern ⊕ α[idx]
7:    N.Utility ← Call I-Step using curNode.Pattern, α[idx]
      and UMSet_{D_i}
8: else if CType[idx] = 'S' then
9:    N.Pattern ← curNode.Pattern ⊗ α[idx]
10:   N.Utility ← Call I-Step using curNode.Pattern, α[idx]
      and UMSet_{D_i}
11: end if
12: return Algorithm 1 (N, UMSet_{D_i}, α, idx + 1, CType)
```

Lemma 3: The maximum utility of sequence α in a sequence database D is an upper bound of the true utility of α in D.

Proof: According to Definition 9, if α is not a L-HUSP in a partition D_i, $su_L(\alpha, D_i) < \delta \cdot U_{D_i}$.

Let D^1 be the set of partitions in D where α is an L-HUSP and D^2 be the set of partitions in D where α is not an L-HUSP. Considering Definition 13 and Definition 14:

$$su_G(\alpha, D) = \sum_D su_L(\alpha, D_i) = \sum_{D_i \in D^1} su_L(\alpha, D_i) + \sum_{D_i \in D^2} su_L(\alpha, D_i) \leq \sum_{D_i \subseteq D^1} su_L(\alpha, D_i) + \sum_{D_i \subseteq D^2} \delta \cdot U_{D_i} = MAS(\alpha, D)$$

Hence $MAS(\alpha, D)$ is an upper bound of the true utility of α in D. ■

Definition 15: **(Potential Global High Utility Sequential Pattern (PG-HUSP))** Given a minimum utility threshold δ and a sequence database D, α is called *PG-HUSP* iff: $MAS(\alpha, D) \geq \alpha \cdot U_D$.

Given set of L-HUSPs, the PG-HUSP generation phase finds all PG-HUSPs in one reduce stage.

Reduce stage: the L-HUSPs having the same key (i.e., pattern) are collected into the same reducer. Let α be an L-HUSP in reducer R. If the pair $\langle \alpha, \langle D_i, utility \rangle \rangle$ exists, then $MAS(\alpha, D)$ is increased by *utility* value. Otherwise, it adds $\delta \cdot U_{D_i}$ as the maximum utility of α in D_i. All the patterns whose MAS value is no less than the threshold are returned as PG-HUSPs.

D. G-HUSP Mining

Given the set of PG-HUSPs (i.e., *PG-Set*), the G-HUSP mining phase calculates the global utility of each pattern in *PG-Set* and discovers G-HUSPs.

Map stage: each mapper calculates the local utility of all patterns in *PG-Set* as follows. If a pattern $\alpha \in$ PG-Set is an L-HUSP in partition D_i, then its utility has already been calculated in the *L-HUSP mining* phase and the mapper

returns the pair $\langle \alpha, \langle D_i, utility \rangle \rangle$. Otherwise, the mapper calculates α's utility. We design a pattern-growth algorithm that traverses the minimum search space to calculate the utility of α in a partition. Below we first provide some definitions and then describe the proposed algorithm to calculate the utility of α.

Similar to the other pattern-growth approaches [4], the search space is a lexical sequence tree, where each non-root node represents a sequence of itemsets. Each node at the first level under the root is a sequence of length 1, a node on the second level represents a 2-sequence and so on. Each non-root node of the tree has two fields: (1) *Pattern*: the pattern presented by the node, and (2) *Utility*: the utility of the pattern for all the sequences in the database. There are two types of patterns presented by nodes in the tree:

Definition 16: **(I-concatenate Sequence)** Given a sequence pattern α, an *I-concatenate* pattern β represents a sequence generated by adding an item I into the last itemset of α (denoted as $\alpha \oplus I$).

Definition 17: **(S-concatenate Sequence)** Given a sequence α, an *S-concatenate* pattern β represents a sequence generated by adding a 1-Itemset $\{I\}$ after the last itemset of α (denoted as $\alpha \otimes I$).

I-concatenate and *S-concatenate* sequences are generated using *sequence-extension step (S-step)* and *itemset-extension step (I-step)* respectively. We demonstrate *I-step* and *S-step* procedures of pattern $\alpha = \{a\}$ with sequence S_3 in Figure 1 (b). We start from the *I-Step*. Given the pattern α and item $I = b$, in order to form $\beta = \{ab\}$ and calculate its utility, USpan applies I-step as follows. According to Figure 1 (b), only itemset S_3^2 has b which is $\langle 15, 8 \rangle$ can be used to form sequence β. The utility of β is the utility of $su(\alpha, S_3)$ plus the newly added item's utilities $su(b, S_3^2)$. Therefore, $su(\beta, S_3) = \{23\}$. Given pattern $\alpha = \{ab\}$ and $I = c$, to construct pattern $\beta = \{ab\}\{c\}$ and calculate its utility, S-step works as follows. Since itemset $\{c\}$ must occur in any itemset after α occurs, the only case for itemset $\{c\}$ is in S_3^3. Hence, $su(\beta, S_3) = \{23 + 3\} = 26$.

A mapper calculates α's utility by calling Algorithm 1. Given partition D_i, Algorithm 1 is designed such that a minimum required search space is traversed to calculate utility of a pattern in D_i. Algorithm 1 takes the following parameters as inputs: (1) $curNode$: the current node in the search space. The initial value is *root* node which is an empty node. (2) $UMSet_{D_i}$ is the set of UMs in D_i. (3) α is a PG-HUSP and presented as a list of items. (4) idx: is an index pointing at the current item in α and its initial value is zero. (5) *CType* is an array representing the types of concatenation in the sequence α. Each element value is either I for I-concatenate pattern or S for S-concatenate pattern.

Figure 3 shows how BigHUSP calculates the local utility of pattern $\alpha = \{b\}\{ac\}$ in D_2 in Figure 1. It starts by an empty sequence (e.g., β) and the utility value equals *zero*. Since the first item in α is b, β is extended by the itemset $\{b\}$ to form S-concatenate pattern $\beta = \{b\}$. Iteratively, β is extended by items in α until all the items in α are added to β. In each iteration, the utility of the extended sequence is calculated using UMs in the partition. For example, the utility of $\beta = \{b\} \otimes \{a\}$ in D_2 is calculated as follows. According to Figure 1(b), $OccSet(\{b\}, S_3) = \{S_3^1, S_3^2\}$ and $OccSet(\{a\}, S_3) = \{S_3^2, S_3^3\}$. Since itemset $\{a\}$ must occur in any itemset after $\{b\}$ occurs, the utility of β is: $su(\beta, S_3) = \max(\{u(\{b\}, S_3^1) + u(\{a\}, S_3^2)\}, \{u(\{b\}, S_3^1) + u(\{a\}, S_3^3)\}, \{u(\{b\}, S_3^2) + u(\{a\}, S_3^3)\}) = \max(\{9 + 8\}, \{9 + 4\}, \{15 + 4\}) = 19$.

Reduce stage: Given a set of PG-HUSPs and their utility values, the pairs with same pattern (which is the key) will be sent to the same reducer. The input is in the form of $\langle pattern, utility \rangle$ where the *utility* is the local utility generated by mappers. After all PG-HUSPs are read, the reducer sums up the utility of each pattern. All the patterns whose total utility is no less than the threshold are returned as G-HUSPs.

Theorem 1: Given a sequence database D, if a pattern α is among the global high utility sequential patterns, it is returned by BigHUSP.

Proof: We prove the theorem by showing that the proposed pruning strategies in BigHUSP never miss a global high utility sequential pattern.

1) *Pruning unpromising items*: unpromising items are found using the proposed overestimate utility model (i.e., GSWU). According to Lemma 1, GSWU has the *downward closure property*. That is, if the GSWU value of an item is less than the threshold, not only the utility of the item but also the utility of all sequences contain the item is less than the threshold and they are not G-HUSP. Hence, we do not miss any G-HUSP by pruning such items.
2) *L-HUSP mining*: Instead of finding all patterns in a partition that has a non-zero utility value, we discover local high utility sequential patterns. According to Lemma 1, pruning local low utility sequential patterns will not cause missing any global high utility sequential patterns. Hence, no G-HUSP will be missed during L-HUSP mining.
3) *PG-HUSP discovery*: In the third phase of BigHUSP, the patterns whose maximum utility value (i.e., MAS) is less than the threshold will be pruned. Since the maximum utility value of a pattern is an upper bound of the true utility of the pattern, if its MAS value is less than the threshold, its utility will be certainly less than the threshold (See Lemma 3). Consequently, pruning the pattens whose MAS value is less than the threshold will not miss any G-HUSP.

Since USpan is an exact method to find all true HUSPs in a single node [4] and the proposed pruning strategies do not miss any G-HUSP, if α is among the global high utility sequential patterns, it will be returned by BigHUSP. ■

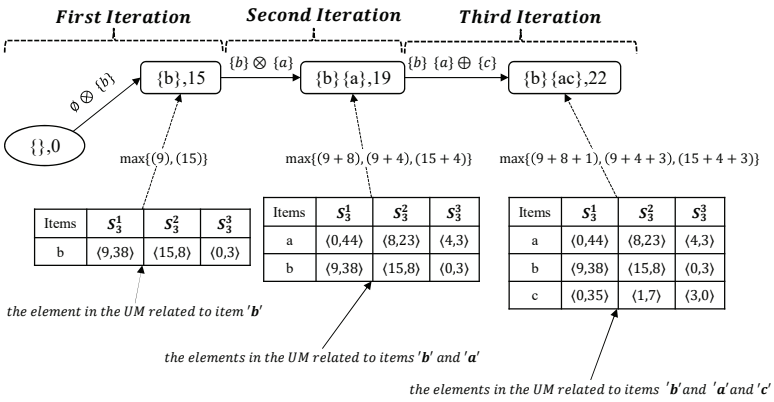

Figure 3. The utility of $\alpha = \{b\}\{ac\}$ in D_2 in Figure 1

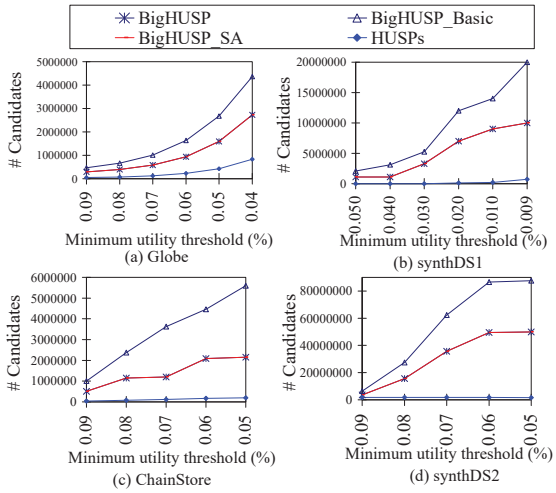

Figure 4. Number of candidates produced by the algorithms

V. Experimental Results

The experimental environment contains one master node and six worker nodes. Each node is equipped with Intel Xeon 2.6 Ghz (each 12 core) and 128 GB main memory and the Spark 1.6.0 is used with IBM Platform Conductor for Spark.

We evaluate our algorithms on two real datasets: *Globe* and *Chainstore*. The *Globe* dataset was obtained from a Canadian news agency (*The Globe and Mail* [4]), which is a web clickstream dataset and contains 600K sequences and 24770 distinct articles. The dataset was created based on a random sample of users visiting *The Globe and Mail* during a six months period in 2014. Given a news article nw and a user usr, the *internal utility* of nw with respect to usr is defined as the browsing time (in seconds) that usr spent on nw[5]. In addition, the *external utility* of nw is defined as:

$$p(nw) = \frac{1}{accessDate(nw) - releasedDate(nw) + 1},$$

where $accessDate$ is the date that usr clicks on nw and $releasedDate$ is the released date of nw. Note that 1 in the denominator is added to avoid zero division. The *ChainStore* dataset consists of 3000K customer buying sequences and 46086 distinct items which already contains internal and external utilities[6]. We also evaluate our algorithm on synthetic datasets. Two synthetic datasets *synthDS1:D2000K-C10-T3-S4-I3-N10K* and *synthDS2:DB-D4000KC15T2.5S4I2.25N10K* are generated by the IBM data generator [22]. The number of sequences in synthDS1 (synthDS2) is 2000K(4000K), the average number of transactions in a sequence is 10(15), the average number of items in a transaction is 3(2.5), the average length of a maximal pattern consists of 4(4) itemsets and each itemset is composed of 3(2.25) items average. The number of items in the dataset is 10k(10k). We follow previous studies [5], [23] to generate internal and external utilities of items in the synthetic datasets. The external utility of each item is generated between 1 and 100 by using a log-normal distribution and the internal utilities of items in a transaction are randomly generated between 1 and 100.

In the experiments, we use the following performance measures: (1) *Run time*: the total execution time of a method to find HUSPs in big data, and (2) *Number of Candidates*: the number of intermediate candidates produced by the algorithms.

[4]http://www.theglobeandmail.com/

[5]We consider the time interval between two consecutive visited news articles as time spent of the former article. The last visited news article is removed from the sequence since we cannot calculate its time spent. We also consider the maximum time spent 15 minutes for news articles whose time spent is more than 15 minutes.

[6]The original dataset contains 1000K transactions. We grouped transactions in different sizes so that each group represents a sequence of transactions. We duplicated each sequence in the dataset five times.

Table I
EXECUTION TIME ON THE DIFFERENT DATASETS

m = Minutes, h = Hours				
Dataset	δ (%)	*BigHUSP*	$BigHUSP_{Basic}$	$BigHUSP_{SA}$
Globe	0.09	1.6 m	3.6 m	0.99 h
	0.08	2.3 m	4.4 m	1.4 h
	0.07	3.1m	6.6 m	2.2 h
	0.06	5.0 m	11.0 m	3.3 h
	0.05	9.2 m	20.7 m	4.5 h
synthDS1	0.05	3.0 m	10.0 m	1.1 h
	0.04	4.26 m	14.4 m	1.2 h
	0.03	6.23 m	17.9 m	1.6 h
	0.02	9.9 m	27.2 m	1.9 h
	0.01	14.3 m	29.4 m	3.2 h
	0.009	37.6 m	76.5 m	7.8 h
ChainStore	0.09	15.0 m	33.4 m	6.4 h
	0.08	19.9 m	56.2 m	12.0 h
	0.07	25.0 m	77.0 m	13.4 h
	0.06	34.8 m	107.7 m	14.6 h
	0.05	38.7 m	159.8 m	17.4 h
synthDS2	0.09	13.1 m	26.3 m	7.7 h
	0.08	16.3 m	34.5 m	9.3 h
	0.07	20.6 m	47.2 m	15.7 h
	0.06	23.8 m	51.8 m	17.8 h
	0.05	32.2 m	85.3 m	21.4 h

To the best of our knowledge, no method was proposed to mine HUSPs in big data. Our preliminary experiment showed that the existing methods (e.g., *USpan*) which were inherently designed for running on a single machine were not able to handle the above 4 datasets due to the out-of-memory problem and very long processing time. Therefore, we implemented two versions of BigHUSP as baseline methods: (1) a basic version of BigHUSP, called $BigHUSP_{Basic}$, which does not apply the proposed pruning strategy to prune unpromising items and also L-HUSPs in the PG-HUSP generation phase, and (2) a stand alone version, called $BigHUSP_{SA}$, which runs *BigHUSP* on a single node of the cluster and does not have the inter-node communication cost.

A. Number of Candidates

Figure 4 shows the results in terms of the number of generated intermediate candidates under different utility thresholds. In this figure, *HUSPs* presents the number of HUSPs found in the datasets for different minimum utility threshold values. As shown in Figure 4, BigHUSP produces much fewer candidates than $BigHUSP_{Basic}$. On the larger datasets, i.e., synthDS1, ChainStore and synthDS2, the number of candidates grows quickly when the threshold decreases. The main reason why *BigHUSP* produces much fewer candidates is that it applies the proposed pruning strategies which avoid generating a large number of intermediate candidates during the mining process.

B. Time Efficiency of BigHUSP

Table I shows the execution time of the algorithms on each of the four datasets with different minimum utility thresholds. As it is shown in the Table I, *BigHUSP* is much faster than $BigHUSP_{SA}$. For example, BigHUSP runs 25 times faster on the *ChainStore* dataset and more than 40 times faster than $BigHUSP_{SA}$ on synthDS2. The average execution time of BigHUSP on *Globe* is 4 minutes, while that of $BigHUSP_{SA}$ on the same dataset is close to 2 hours.

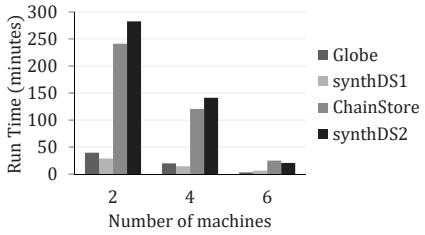

Figure 5. Scalability of BigHUSP on different datasets

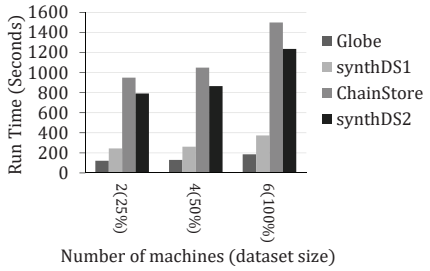

Figure 6. Scalability of BigHUSP on different datasets with different size

Besides, it can be observed that BigHUSP runs faster than $BigHUSP_{Basic}$ as well. This performance asserts that the proposed pruning strategies to reduce the search space and communication costs are effective.

C. Scalability

In this section, we investigate the scalability of BigHUSP. To evaluate scalability, we first run BigHUSP on a fixed dataset using 2, 4, and 6 worker nodes. The results are shown in 5. In this experiment, the threshold values for Globe, synthDS1, ChainStore and synthDS2 datasets are set to 0.06%, 0.03%, 0.07% and 0.07% respectively. BigHUSP showed linear scalability as the number of available machines is increased, meaning equally to decrease the run time for the mining task. The ability of BigHUSP to scale up can be related to efficiency of the proposed pruning strategies and the large number of partitions that can processed and mined independently with limited number of MapReduce jobs.

To further evaluate the scalability of BigHUSP, we also perform a weak scalability experiment for BigHUSP, in which we increased the number of available worker nodes (2, 4, 6) and the size of the sequence database (25%, 50%, 100% sequences of the dataset) simultaneously. Ideally, we expect the total run time remains constant as we increase the number of nodes and the size of the dataset simultaneously. As Figure 6 shows, the rune time is almost constant. In this figure, an increase in run time is observed when the number of machines is 6. This is due to the fact that when the size of the input sequences is doubled, the number of output sequences increases more than twice.

VI. Conclusions

In this paper, we proposed a novel algorithm called *BigHUSP* for parallel and distributed mining of high utility sequential patterns from big data. BigHUSP uses multiple MapReduce-like steps to process the input data in parallel. It divides the input sequences into several partitions and processes each partition of the input sequence database independently. Two novel and distributed strategies are proposed to effectively prune the search space and greatly improve the performance of BigHUSP. Our experiments suggest that BigHUSP is orders of magnitudes more efficient and scalable than baseline algorithms for high utility sequential pattern mining. For example, in our experiments, BigHUSP mined 4 million input sequences in less than half an hour on seven machines, while the baseline approaches took around 20 hours to return the results.

VII. Acknowledgment

This work is funded in part by Natural Sciences and Engineering Research Council of Canada (NSERC), and the Big Data Research, Analytics, and Information Network (BRAIN) Alliance established by the Ontario Research Fund - Research Excellence Program (ORF-RE).

References

[1] N. R. Mabroukeh and C. I. Ezeife, "A taxonomy of sequential pattern mining algorithms," *ACM Comput. Surv.*, vol. 43, no. 1, pp. 3:1–3:41, 2010.

[2] C. H. Mooney and J. F. Roddick, "Sequential pattern mining approaches and algorithms," *ACM Comput. Surv.*, vol. 45, no. 2, pp. 19:1–19:39, 2013.

[3] L. Cao, Y. Zhao, H. Zhang, D. Luo, C. Zhang, and E. Park, "Flexible frameworks for actionable knowledge discovery," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 9, pp. 1299–1312, 2010.

[4] J. Yin, Z. Zheng, and L. Cao, "Uspan: An efficient algorithm for mining high utility sequential patterns," in *In Proc. of ACM SIGKDD*, 2012, pp. 660–668.

[5] C. F. Ahmed, S. K. Tanbeer, and B. Jeong, "A novel approach for mining high-utility sequential patterns in sequence databases," *In ETRI Journal*, vol. 32, pp. 676–686, 2010.

[6] R. Kitchin, *Big Data*. John Wiley and Sons, Ltd, 2016. [Online]. Available: http://dx.doi.org/10.1002/9781118786352.wbieg0145

[7] A. Mitchell and D. Page, "State of the news media 2015," *Pew Research Journalism Project 2015*, 2015. [Online]. Available: http://www.journalism.org/files/2015/04/FINAL-STATE-OF-THE-NEWS-MEDIA1.pdf

[8] H. Kashyap, H. A. Ahmed, N. Hoque, S. Roy, and D. K. Bhattacharyya, "Big data analytics in bioinformatics: A machine learning perspective," *CoRR*, vol. abs/1506.05101, 2015. [Online]. Available: http://arxiv.org/abs/1506.05101

[9] W. Fan and A. Bifet, "Mining big data: Current status, and forecast to the future," *SIGKDD Explor. Newsl.*, vol. 14, no. 2, pp. 1–5, 2013.

[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, pp. 10–10, 2010.

[11] M. Zihayat and A. A. Mining, "top-k high utility patterns over data streams," *In Information Sciences, Available Online*, 2014.

[12] B. Shie, H. Hsiao, and V. S. Tseng, "Efficient algorithms for discovering high utility user behavior patterns in mobile commerce environments," *In KAIS journal*, vol. 37, 2013.

[13] C. F. Ahmed, S. Tanbeer, and B. Jeong, "A framework for mining high utility web access sequences," *In IETE Journal*, vol. 28, pp. 3–16, 2011.

[14] 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, no. 10, pp. 2645–2657, Oct 2015.

[15] Y. C. Lin, C.-W. Wu, and V. S. Tseng, *Mining High Utility Itemsets in Big Data*. Cham: Springer International Publishing, 2015, pp. 649–661.

[16] V. Guralnik and G. Karypis, "Parallel tree-projection-based sequence mining algorithms," *Parallel Comput.*, vol. 30, no. 4, pp. 443–472, Apr. 2004. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2004.03.003

[17] M. J. Zaki, "Parallel sequence mining on shared-memory machines," *J. Parallel Distrib. Comput.*, vol. 61, no. 3, pp. 401–426, Mar. 2001. [Online]. Available: http://dx.doi.org/10.1006/jpdc.2000.1695

[18] W. Fang, M. Lu, X. Xiao, B. He, and Q. Luo, "Frequent itemset mining on graphics processors," in *Proceedings of the Fifth International Workshop on Data Management on New Hardware*, ser. DaMoN '09. New York, NY, USA: ACM, 2009, pp. 34–42. [Online]. Available: http://doi.acm.org/10.1145/1565694.1565702

[19] D. Yu, W. Wu, S. Zheng, and Z. Zhu, *BIDE-Based Parallel Mining of Frequent Closed Sequences with MapReduce*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 177–186.

[20] J.-W. Huang, S.-C. Lin, and M.-S. Chen, *DPSP: Distributed Progressive Sequential Pattern Mining on the Cloud*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 27–34.

[21] C. F. Ahmed, S. K. Tanbeer, and B. Jeong, "A framework for mining high utility web access sequences," *In IETE Journal*, vol. 28, pp. 3–16, 2011.

[22] R. Agrawal and R. Srikant, "Mining sequential patterns," in *ICDE*, 1995, pp. 3–14.

[23] M. Zihayat, C.-W. Wu, A. An, and V. S. Tseng, "Mining high utility sequential patterns from evolving data streams," in *ASE BD&SI '15*, 2015, pp. 52:1–52:6.