Mining High Utility Itemsets in Big Data

Ying Chun Lin[1(✉)], Cheng-Wei Wu[2], and Vincent S. Tseng[2]

[1] Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan, Republic of China
yclin@idb.csie.ncku.edu.tw
[2] Department of Computer Science, National Chiao Tung University, Tainan, Taiwan, Republic of China
silvemoonfox@hotmail.com, vtseng@cs.nctu.edu.tw

Abstract. In recent years, extensive studies have been conducted on *high utility itemsets* (*HUI*) mining with wide applications. However, most of them assume that data are stored in centralized databases with a single machine performing the mining tasks. Consequently, existing algorithms cannot be applied to the big data environments, where data are often distributed and too large to be dealt with by a single machine. To address this issue, we propose a new framework for *mining high utility itemsets in big data*. A novel algorithm named *PHUI-Growth* (*Parallel mining High Utility Itemsets by pattern-Growth*) is proposed for parallel mining HUIs on Hadoop platform, which inherits several nice properties of Hadoop, including easy deployment, fault recovery, low communication overheads and high scalability. Moreover, it adopts the MapReduce architecture to partition the whole mining tasks into smaller independent subtasks and uses Hadoop distributed file system to manage distributed data so that it allows to parallel discover HUIs from distributed data across multiple commodity computers in a reliable, fault tolerance manner. Experimental results on both synthetic and real datasets show that *PHUI-Growth* has high performance on large-scale datasets and outperforms state-of-the-art non-parallel type of HUI mining algorithms.

Keywords: High utility itemset mining · Big data analytics · Hadoop platform

1 Introduction

The rapid growth of data generated and stored has led us to the new era of Big Data [3, 4, 14, 18, 19]. Nowadays, we are surrounded by different types of big data, such as enterprise data, sensor data, machine-generated data and social data. Extracting valuable information and insightful knowledge from big data has become an urgent need in many disciplines. In view of this, *big data analytics* [3, 4, 14, 18, 19] has emerged as a novel topic in recent years. This technology is particularly important to enterprises and business organizations because it can help them to increase revenues, retain customers and make more intelligent decisions. Due to its high impact in many areas, more and more systems and analytical tools have been developed for big data analytics, such as *Apache Mahout* [14], *MOA* [3], *SAMOA* [19] and *Vowpal Wabbit* [20]. However, to the best of our knowledge, no existing studies have incorporated the concept of *utility mining* [2, 6, 7, 8, 11, 12, 13] into big data analytics.

T. Cao et al. (Eds.): PAKDD 2015, Part II, LNAI 9078, pp. 649–661, 2015.
DOI: 10.1007/978-3-319-18032-8_51

Utility mining is an important research topic in data mining. The main objective of utility mining is to extract valuable and useful information from data by considering profit, quantity, cost or other user preferences. *High utility itemset* (*HUI*) *mining* is one of the most important tasks in utility mining, which can be used to discover sets of items carrying high utilities (e.g., high profits). This technology has been applied to many applications such as *market analysis*, *web mining*, *mobile computing* and even *bioinformatics*. Due to its wide range of applications, many studies [2, 6, 7, 8, 11, 12, 13] have been proposed for mining HUIs in databases. However, most of them assume that data are stored in centralized databases with a single machine performing the mining tasks. However, in big data environments, data may be originated from different sources and highly distributed. A large volume of data also makes it difficult to be moved to a centralized database. Thus, existing algorithms are not suitable for the applications of big data.□

Although mining HUIs from big data is very desirable for many applications, it is a challenging task due to the following problems posed: First, due to a large amount of transactions and varied items in big data, it would face the large search space and the combination explosion problem. This leads the mining task to suffer from very expensive computational costs in practical. Second, pruning the search space in HUI mining is more difficult than that in frequent pattern mining because the *downward closure property* [1] does not hold for the utility of itemsets. Therefore, many search space pruning techniques developed for frequent pattern mining cannot be directly transferred to the scenario of HUI mining. Third, a large amount of data cannot be efficiently processed by a single machine. A well-designed algorithm incorporated with parallel programming architecture is needed. However, implementing a parallel algorithm involves several problematic issues, such as search space decomposition, avoidance of duplicating works, minimization of synchronization and communication overheads, fault tolerance and scalability problems.

In this paper, we address all of the above challenges by proposing a new framework for *mining high utility itemsets in big data*. To our knowledge, this topic has not yet been explored. The contributions of this work are summarized as follows:

- First, we propose a novel algorithm named *PHUI-Growth* (*Parallel mining High Utility Itemsets by pattern-Growth*) for parallel mining HUIs in big data. It is implemented on a Hadoop platform [14] and thus it inherits several nice properties from Hadoop, such as easy deployment in high level language, fault tolerance, low communication overheads and high scalability on commodity hardware.
- Second, PHUI-Growth adopts the MapReduce architecture to partition the whole mining task into smaller independent subtasks and uses *HDFS* (*Hadoop Distributed File System*) to process distributed data. Thus, it can parallel mine HUIs from distributed databases across multiple commodity computers in a reliable manner.
- Third, PHUI-Growth adopts a novel strategy called *DLU-MR* (*Discarding local unpromising items in MapReduce framework*) to effectively prune the search space and unnecessary intermediate itemsets produced during the mining process, which further enhances the performance of PHUI-Growth.

- Experimental results on both synthetic and real datasets show that PHUI-Growth outperforms the state-of-the-art algorithms developed for mining HUIs on a single machine and that it has good scalability on large-scale datasets.

The remaining of this paper is organized as follows. Section 2 and Section 3 respectively introduce the basic concepts of HUI mining and related works. Section 4 presents the proposed methods. Experimental results and conclusion are presented in Section 5 and Section 6, respectively.

2 Basic Concept and Definitions

In this section, we introduce the related definitions about HUI mining. Let $I^* = \{I_1, I_2, \ldots, I_N\}$ be a finite set of distinct *items*. A *transactional database* $D = \{T_1, T_2, \ldots, T_M\}$ is a set of transactions, where each transaction $T_c \in D$ $(1 \leq c \leq M)$ is a set of items has a unique transaction identifier c, called its *TID*. Each item $I_j \in I^*$ $(1 \leq j \leq N)$ in D has a global positive real number $p(I_j, D)$, called its *external utility* (e.g., unit profit). The external utilities of items are stored in a *utility table*. Every item I_j in a transaction T_c has a local positive real number $q(I_j, T_c)$, called its *internal utility* (e.g., quantity). An *itemset* X is a set of items $\{I_1, I_2, \ldots, I_k\}$, where $k=|X|$ is called *the length of X*. An itemset of length k is called k-itemset.

Definition 1 (Utility of an itemset in a transaction). *The utility of an itemset X in a transaction $T_c \in D$* is denoted as $u(X, T_c)$ and defined as $\sum_{I_j \in X \wedge I_j \in T_c} p(I_j, D) \times q(I_j, T_c)$.

Definition 2 (Utility of an itemset in a database). *The utility of an itemset X in a database D* is the summation of the utilities of X in all the transactions containing X, which is denoted as $u(X)$ and defined as $\sum_{T_c \in D \wedge X \subseteq T_c} u(X, T_c)$.

Definition 3 (Transaction utility of a transaction). *The transaction utility* (*abbreviated as TU*) *of a transaction $T_c \in D$* is the summation of the utility of each item in T_c, which is denoted as $tu(T_c)$ and defined as $\sum_{T_c \in D} u(T_c, T_c)$.

Definition 4 (Total utility of a database). *The total utility of a database D* is the summation of the transaction utility of each transaction in the database, which is denoted as λ and defined as $\sum_{T_c \in D} tu(T_c)$.

Table 1. Transactional database

TID	Transaction
T_1	A(4), B(2), C(8), D(2)
T_2	A(4), B(2), C(8)
T_3	C(4), D(2), E(2), F(2)
T_4	E(2), F(2), G(1)

Table 2. Utility table

Item	A	B	C	D	E	F	G
External Utility	2	3	1	3	4	4	8

Definition 5 (Relative utility of an itemset in a database). *The relative utility of an itemset X in a database D* is denoted as $ru(X)$ and defined as the ratio of $u(X)$ to λ.

Definition 6 (High utility itemset). *Let* θ $(0 < \theta \leq \lambda)$ *be a user-specified minimum utility threshold. An itemset X is called a high utility itemset* (abbreviated as *HUI*) *iff* $u(X) \geq \theta$. An equivalent definition is that X is a HUI iff $ru(X) \geq \theta/\lambda$, where θ/λ is called *relative minimum utility threshold.* Otherwise, X is called *low utility itemset*.

Notice that the well-known *downward closure property* [1] does not held for the utility of itemsets. For example, {A} is low utility, but its superset {AC} is high utility. As a consequence, the search space of HUI mining cannot be effectively pruned as it is done in traditional frequent itemset mining. To effectively prune the search space, the concept of *transaction-weighted utilization model* (abbreviated as *TWU model*) [6] was proposed, which is based on the following definitions.

Definition 7 (TWU of an itemset). The *transaction-weighted utilization* (*abbreviated as TWU*) *of an itemset X* is the summation of transaction utility of each transaction containing X, which is denoted as $TWU(X)$ and defined as $\sum_{T_c \in D \wedge X \subseteq T_c} tu(T_c)$.

Definition 8 (High TWU itemset). An itemset X is called *high TWU* itemset iff $TWU(X) \geq \theta$. Otherwise, X is called *low TWU itemset*.

Definition 9 (TWU downward closure property). *The TWU downward closure property* states that any superset of a low TWU itemset is low utility. By this property, the *downward closure property* can be applied to the TWU of itemsets for effectively prune the search space. The detailed proof of this property can be found in [6].

3 Related Work

In this section, we review some studies that are related to parallel programming, HUI mining, and parallel HUI mining.

3.1 Parallel Programming

Parallel programming has become a necessity for handling big data. The parallel algorithms can be generally categorized into two types: *shared-memory algorithm* and *shared-nothing algorithm* (also called *distributed algorithm*) [14]. The main feature of shared-memory algorithms is that it allows all processing units to concurrently access a shared memory. In general, it is easier to adapt algorithms to the shared-memory architecture. However, the resulting algorithms are usually not scalable enough and still suffer from the bottleneck of huge memory requirement. On the other hand, the main feature of shared-nothing algorithms is that it allows different processors that have their own memories to communicate with each other by passing messages. Although it is not easy to adapt algorithms to the shared-nothing architecture, a well-designed distributed algorithm usually has better scalability.

The *message passing interface* (*MPI*) is one of the most well-known framework based on shared-nothing architecture, but it works efficiently only on low-level programming languages (e.g., *C* and *Fortran*). Nowadays, high-level programming languages (e.g., *Java*) have become more and more important and popular in many domains. *Apache Foundation* has developed a Jave-based open-source library named *Hadoop* [14] for parallel processing big data. It consists of two key services: *HDFS* (*Hadoop Distributed File System*) and *MapReduce software*. HDFS is a reliable distributed file system designed to efficiently access and store large-scale data. On the other hand, MapReduce software is designed to process vast amounts of data in parallel. The combination of HDFS and MapReduce software allows to parallel process large-scale datasets across multiple clusters of commodity hardware in a reliable, fault-tolerant manner. A MapReduce program consists of two stages: *map stage* and *reduce stage*. In map stage, each Mapper processes a distinct chunk of data and produces several key-value pairs. In reduce stage, these key-value pairs are aggregated and transformed. The transformed key-value pairs are fed to Reducers. Reducers further process these transformed pairs and then output the final or intermediate results.

3.2 High Utility Itemset Mining

Extensive studies have been proposed for efficiently mining HUIs in centralized databases. These algorithms can be generally categorized into two types: *two-phase* and *one-phase* algorithms. The main characteristic of two-phase algorithms is that they consist of two phases. In the first phase, they generate a set of candidates (e.g., high TWU itemsets) for HUIs. In the second phase, they calculate the utility of each candidate found in the first phase to identify HUIs. For example, *Two-Phase* [6], *IHUP* [2], *IIDS* [9] and *UP-Growth*$^+$ [12] are typical two-phase algorithms. On the contrary, the main feature of one-phase algorithms [7, 8] is that they discover HUI using only one phase. For example, *HUI-Miner* [7] is one of the state-of-the-art one-phase algorithms. Although these algorithms are very efficient for mining HUIs from a centralized database, they have not been parallelized for handling big data.

3.3 Parallel High Utility Itemset Mining

In utility mining, only few preliminary studies [11, 13] have been proposed for parallel mining HUIs in distributed databases. Vo et al. proposed the *DTWU-Mining* algorithm [13] for parallel mining HUIs from vertical partitioned distribute databases. Subramanian et al. proposed the *FUM-D* algorithm [11] to extract HUIs from distributed horizontal databases. FUM-D enumerates local HUIs in each local database and then uses local HUIs to infer all global HUIs in the global database. Although these two approaches are parallel HUI mining algorithms, they are not implemented in Hadoop platform and do not integrated with the MapReduce framework. Therefore, they do not support fault tolerance. However, fault tolerance is a very important issue in parallel mining algorithms because the probability that none of computers of cluster crashes is very small when handling big data. Therefore, DTWU-Mining and FUM-D are not reliable and practical enough for handling big data.

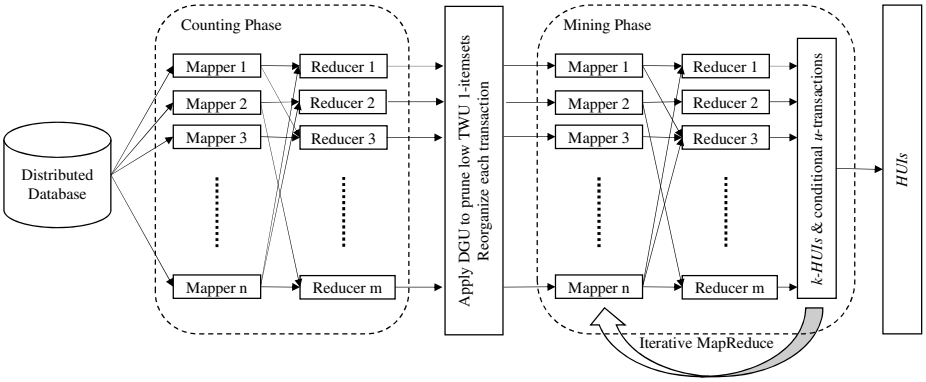

Fig. 1. The system architecture of the PHUI-Growth algorithm

4 The Proposed Method: PHUI-Growth

In this section, we propose a novel algorithm named *PHUI-Growth* (*Parallel mining High Utility Itemsets by pattern-Growth*) for efficiently parallel mining HUIs based on Hadoop MapReduce architecture. It has three input parameters: (1) a distributed database *DD*, (2) a utility table and (3) a user-specified minimum utility threshold *min_util*. After the whole mining process, the algorithm outputs the complete set of HUIs in *DD*. PHUI-Growth consists of three main phases: (1) *counting phase*, (2) *transformation phase* and (3) *mining phase*. Fig. 1 shows the system architecture of PHUI-Growth.

4.1 Counting Phase

The input database *DD* can be viewed as a set of transactions that are stored in several computers. In counting phase, the algorithm takes one MapReduce pass to parallel counts TWU of items in *DD*. The whole process in this phase can be divided into map stage and reduce stage.

Map Stage. In map stage phase, each Mapper is fed with a transaction $T_c = \{I_1, I_2,\ldots, I_L\}$ in *DD*. For each item I_j in T_c ($1 \le j \le L$), the Mapper outputs a key-value pair $<I_j, TU(T_c)>$, called *Item-TU pair*.

Reduce Stage. In reduce stage, Item-TU pairs outputted by Mappers are fed to Reducers. The Item-TU pairs having the same key are collected into the same Reducer. Let $R = \{<I, v_1>, <I, v_2>,\ldots, <I, v_n>\}$ be the set of all the Item-TU pairs collected by a Reducer. The Reducer calculates TWU of *I* by summing up each value of a pair and outputs a key-value pair $<I, TWU(I)>$, called *Item-TWU pair*.

Fig. 1 shows the process of map and reduce stages in counting phase. In Fig.2(a), the input of the Mapper 1 is T_1 = {A, B, C, D} of Table 1. After the process, the Mapper outputs four Item-TU pairs <{A}, 28>, <{B}, 28>, <{C}, 28> and <{D}, 28>. All the Item-TU pairs having the same key {A} are collected into the Reducer 1. After the process, Reducer 1 outputs an Item-TWU pair <{A}, 50>.

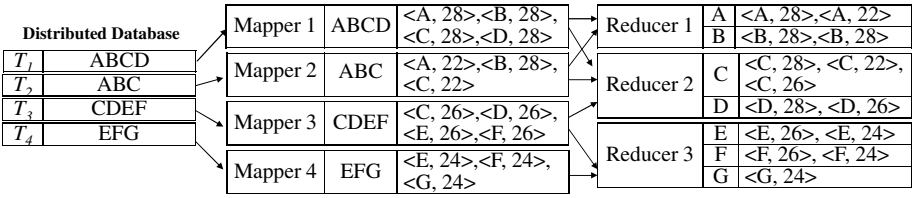

Fig. 2. Map and reduce stages in counting phase

4.2 Database Transformation Phase

In database transformation phase, the algorithm removes all low TWU 1-itemsets from *DD* (Definition 7, 8 and 9) and sorts remaining items in a TWU ascending order. A transaction after the above process is called *reorganized transaction*. Then, the algorithm transforms each reorganized transaction $T' = \{I_1, I_2, \ldots, I_L\}$ in *DD* into a special structure called *u-transaction.* A *u-transaction* is of the form $\langle I_1(u_1), I_2(u_2), \ldots, I_L(u_L)\rangle$, where u_j is called *the utility of* I_j *in* T'. Initially, u_j is set to $p(I_j, DD) \times q(I_j, T')$. The *transaction utility of* T' is denoted as $TU(T')$ and defined as the summation of utilities of all the items in T'.

4.3 Mining Phase

In mining phase, the algorithm parallel discovers HUIs through several iterations. Initially, a variable k is set to 0. Then, the algorithm starts to generate HUIs having a length greater than k. In the k-th iteration, all the HUIs of length k are discovered by performing a MapReduce pass. The process of this phase can be divided into two cases: (1) $k = 1$ and (2) $k \geq 2$. We first explain the former case and then describe the latter case.

4.3.1 Map Stage in the First Iteration

When $k = 1$, each Mapper is fed with a *u-transaction* $T' = \langle I_1(u_1), I_2(u_2), \ldots, I_L(u_L)\rangle$. For each item I_j in T' $(1 \leq j \leq L)$, the Mapper generates a special structure called *conditional u-transaction.* A conditional *u*-transaction has three fields: *Prefix*, *PrefixUtility* and *UTrans.* When $k = 1$, I_j and u_j are respectively stored in *Prefix* and *PrefixUtility* fields. For *UTrans* field, it stores the set of items appearing after I_j in T' according to TWU ascending order, that is, $\langle I_{j+1}(u_{j+1}), I_{j+2}(u_{j+2}), \ldots, I_L(u_L)\rangle$. The *prefix utility of* T' is denoted as $TU(T')$ and defined as the value in *PrefixUtility* field.

For example, in Fig. 3, the Mapper 1 is fed with the *u*-transaction {A(8), B(6), D(6), C(8)}. After the process, it generates four conditional *u*-transactions <{A}, 8, {B(6), D(6), C(8)}>, <{B}, 6, {D(6), C(8)}>, <{D}, 6, {C(8)}>, <{C}, 8, ϕ >. The PU of the first conditional *u*-transaction is 8.

Mapper 1	A(8) B(6) D(6) C(8)	<{A}, 8, {B(6)D(6)C(8)}> <{B}, 6, {D(6)C(8)}> <{D}, 6, {C(8)}> <{C}, 8, {ϕ}>
Mapper 2	A(8) B(6) C(8)	<{A}, 8, {B(6)C(8)}> <{B}, 6, {C(8)}> <{C}, 8, {ϕ} >
Mapper 3	E(8) F(8) D(6) C(4)	<{E}, 8, {F(8)D(6)C(4)}> <{F}, 8, {D(6)C(4)}> <{D}, 6, {C(4)}> <{C}, 4, {ϕ}>
Mapper 4	E(8) F(8)	<{E}, 8, {F(8)}> <{F}, 8, {ϕ}>

Reducer 1	A	<{A}, 8, {B(6)D(6)C(8)}>, <{A}, 8, {B(6)C(8)}>
	B	<{B}, 6, {D(6)C(8)}>, <{B}, 6, {C(8)}>
Reducer 2	C	<{C}, 8, {ϕ}>, <{C}, 8, {ϕ}>, <{C}, 4, {ϕ}>
	D	<{D}, 6, {C(8)}>, <{D}, 6, {C(4)}>
Reducer 3	E	<{E}, 8, {F(6)D(6)C(4)}> <{E}, 8, {F(8)}>
	F	<{F}, 8, {D(6)C(4)}>, <{F}, 8, {ϕ}>

Fig. 3. Map and reduce stages in mining phase when $k = 1$

Mapper 1	<{A},8,{B(6)D(6)C(8)}>	<{AB}, 14, {D(6)C(8)}> <{AD}, 14, {C(8)}> <{AC}, 16, {ϕ}>
	<{A},8,{B(6)C(8)}>	<{AB}, 14, {C(8)}> <{AC}, 16, {ϕ}>
Mapper 2	<{B}, 6, {D(6)C(8)}>	<{BD}, 12, {C(8)}> <{BC}, 14, {ϕ}>
	<{B}, 6, {C(8)}>	<{BC}, 14, {ϕ}>
Mapper 3	<{D}, 6, {C(8)}>	<{DC}, 14, {ϕ}>
	<{D}, 6, {C(4)}>	<{DC}, 10, {ϕ}>
Mapper 4	<{E}, 8, {F(6)D(6)C(4)}>	<{EF}, 14, {D(6)C(4)}> <{ED}, 14, {C(4)}> <{EC}, 12, {ϕ}>
	<{E}, 8, {F(8)}>	<{EF}, 16, {ϕ}>
Mapper 5	<{F}, 8, {D(6)C(4)}>	<{FD}, 14, {C(4)}> <{FC}, 12, {ϕ}>

Reducer 1	AB	<{AB}, 14, {D(6)C(8)}>, <{AB}, 14, {C(8)}>
	AC	<{AC}, 16, {ϕ}>, <{AC}, 16, {ϕ}>
	AD	<{AD}, 14, {C(8)}>
Reducer 2	BD	<{BD}, 12, {C(8)}>
	BC	<{BC}, 14, {ϕ}>, <{BC}, 14, {ϕ}>
	DC	<{DC}, 14, {ϕ}>, <{DC}, 10, {ϕ}>
Reducer 3	EF	<{EF}, 14, {D(6)C(4)}>, <{EF}, 16, {ϕ}>
	ED	<{ED}, 14, {C(4)}>, <{ED}, 14, {C(4)}>
	EC	<{EC}, 12, {ϕ}>
Reducer 4	FD	<{FD}, 14, {C(4)}>
	FC	<{FC}, 12, {ϕ}>

Fig. 4. Map and reduce stages in mining phase when $k = 2$

4.3.2 Reduce Stage in Mining Phase

In reduce stage, each Reducer is fed with a conditional *u*-transaction. The conditional *u*-transactions having the same prefix are collected into the same Reducer. Let *R*' be the set of conditional *u*-transactions collected by a Reducer and *X* be the prefix of these conditional *u*-transactions. Then, the Reducer calculates the utility of *X* by summing up *PUs* of all the *u*-transactions in *R*'. After the process, if the utility of *X* is no less than the *min_util* threshold, the Reducer outputs *X* and its utility because *X* is a HUI (Definition 6). Then, the algorithm applies a proposed strategy called *DLU-MR* (*Discarding local unpromising items in MapReduce framework*) to further reduce the search space. The main idea of this strategy is based on the following definitions.

Definition 10 (Local TWU of an item in *k*-th iteration). Let *R*' be the set of conditional *u*-transactions collected by a Reducer in *k*-th iteration, *the local TWU of an item Y* is the summation of transaction utility of each *u*-transaction containing *X*, which is denoted as $LTWU(X, k)$ and defined as $\sum_{T \in R \wedge Y \in T} [TU(T) + PU(T)]$.

Definition 11 (Local unpromising items in *k*-th iteration). Let *R*' be the set of conditional *u*-transactions collected by a Reducer in *k*-th iteration, an item is called *local unpromising items* in *k*-th iteration iff $LTWU(X, k) < \theta$.

Definition 12 (Local TWU downward closure property). *The local TWU downward closure property* (or simply called *LTWU-DC Property*) states that $\forall X$ iff $LTWU(X, k) < \theta$, all the *L*-itemsets containing *X* are low utility ($L \geq k$).

The DLU-MR strategy is performed by scanning R' twice. In the first scan of R', the Reducer calculates local TWU of items in u-transactions of R'. In the second scan of R', the Reducer removes local unpromising items in each u-transaction T' of R' and discards their utilities from $TU(T)$. When the Reducer has finished its work, each trimmed conditional u-transaction is outputted as the input of $(k+1)$-th iteration.

4.3.3 Map-Reduce Stages in the k-th Iteration ($k \geq 2$)

In the k-th iteration ($k \geq 2$), the algorithm takes a MapReduce pass to find all the HUIs of length k. In map stage, each Mapper is fed with a conditional *u-transaction* $CT = \langle X, u_x, T' \rangle$ produced from (k-1)-th iteration, where X is the prefix of CT, u_x is the PU of CT and $T' = \langle I_1(u_1), I_2(u_2),\ldots, I_L(u_L) \rangle$ is a u-transaction related to X. For each item I_j in T' ($1 \leq j \leq L$), the Mapper outputs a conditional u-transaction $\langle \{X \cup I_j\}, (u_x+u_j), \langle I_{j+1}(u_{j+1}), I_{j+2}(u_{j+2}),\ldots, I_L(u_L) \rangle \rangle$. When all the Mappers have finished their work, those conditional u-transactions having the same prefix are fed to the same Reducer. When $k \geq 2$, the process of the reduce stage is the same as that in subsection 4.3.2. Fig. 3 and Fig. 4 respectively show the running examples when k =1 and k = 2.

5 Experimental Results

In this section, we compare the performance of PHUI-Growth with HUI-Miner [7], a state-of-the-art non-parallel type of HUI mining algorithms. To evaluate the effectiveness of the DLU-MR strategy, we prepared two versions of PHUI-Growth, respectively called *PHUI-Growth*(*Baseline*) and *PHUI-Growth*(*DLU-MR*). We also evaluate the number of intermediate itemsets produced by the algorithms. For HUI-Miner, intermediate itemsets refers to the itemsets having an estimated utility no less than the *min_util* threshold. Thus, the number of intermediate itemsets produced by HUI-Miner can be regarded as that of utility-lists constructed by HUI-Miner during the mining process. For the proposed algorithms, the number of intermediate itemsets is the number of conditional u-transactions produced by Reducers during the mining process. In this section, the two kinds of intermediate itemsets are called *candidates*. All experiments were conducted on a five-node Hadoop Cluster. Each node is equipped with Intel® Celeron® CPU G1610 @ 2.60GHz CPU and 4 GB main memory. All the algorithms are implemented in Java. Both synthetic and real datasets were used in the experiments. Chainstore [10] is a real-life dataset acquired from [10], which already contain unit profits and purchase quantities. Retail dataset was obtained from FIMI Repository [15]. A synthetic dataset T10I4N10K|D|2,000K was generated from the IBM data generator [1]. The parameters of the dataset are described as follows: $|D|$ is the total number of transactions, T is the average size of transactions; N is the number of distinct items; I is the average size of potential maximal itemsets. Internal and external utilities of items are generated as the settings of [6, 12]. In Retail and T10I4N10K|D|2,000K datasets, external utilities of items are generated between 1 and 1,000 by using a log-normal distribution and internal utilities of items are generated randomly between 1 and 5, as the settings of [6, 12]. To evaluate the performance of the algorithms on a larger dataset, we duplicate each transaction in Chainstore five times to form a dataset named Chainstore×5. Table 3 shows characteristics of the datasets.

Table 3. Characteristic of Datasets

Dataset	#Trans.	# Items	Average Trans. Length	Maximum Trans. Length
Retail	88,162	16,470	10	76
Chainstore	1,112,949	46,086	7	170
T10I4N10KID|2,000K	2,000,000	10,000	10	33
Chainstore×5	5,564,745	46,086	7	170

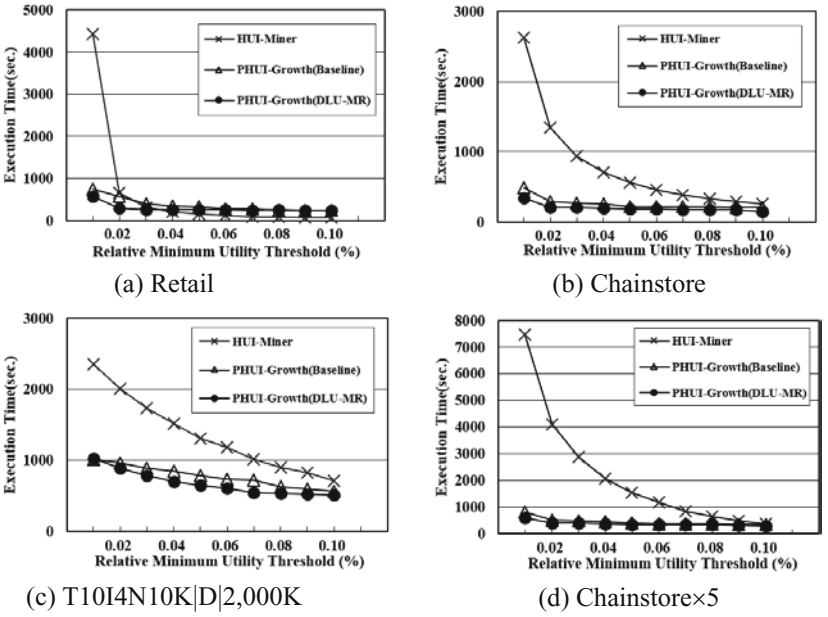

Fig. 5. Execution time of the algorithms on different datasets

5.1 Performance Evaluation on Small Dataset

In this subsection, we evaluate the performance of the algorithms on Retail dataset under varied relative *min_util* thresholds. The execution time of the algorithms and the number of candidates are respectively shown in Fig. 5(a) and Fig 6(a). In Fig. 5(a), PHUI-Growth(Baseline) and Growth(DLU-MR) generally run slightly slower than HUI-Miner when relative *min_util* thresholds are higher than 0.02%. This is because that PHUI-Growth(Baseline) and Growth(DLU-MR) use five-node Hadoop Cluster to parallel process the mining tasks and they need to pass necessary data and messages across different machines via networks, which requires additional communication overheads. Therefore, they take more time than HUI-Miner for high thresholds. However, when the threshold decreases, HUI-Miner starts to suffer from long execution time. For example, for relative *min_util* = 0.01%, HUI-Miner takes 4,429 seconds, while PHUI-Growth(DLU-MR) only takes 566 seconds. When the threshold decreases, the number of HUIs dramatically increases and HUI-Miner need to produce a large amount of utility-lists for intermediate itemsets. However, the number of candidates produced by PHUI-Growth(DLU-MR) is up to two orders of magnitude smaller than that produced by HUI-Miner.

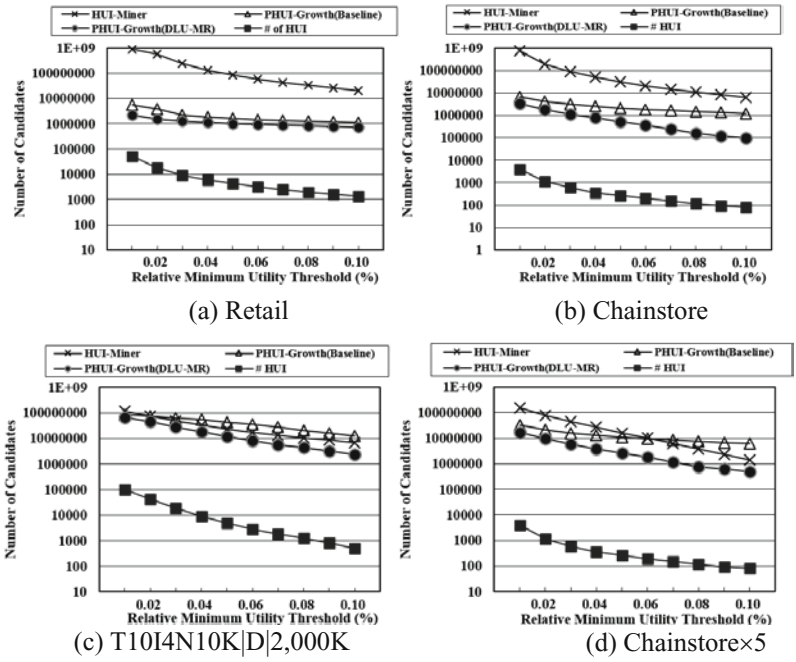

Fig. 6. Number of intermediate itemsets produced by the algorithms

5.2 Performance Evaluation on Large-Scale Datasets

In this subsection, we evaluate the performance of the algorithms on large datasets, including Chainstore, T10I4N10K|D|2000K and Chainstore×5. Execution time of the algorithms and the number of candidates are respectively shown in Fig.5 and Fig. 6. Results show that PHUI-Growth(Baseline) and Growth(DLU-MR) outperform HUI-Miner significantly. The reason why PHUI-Growth(Baseline) and Growth(DLU-MR) perform so well is that they effectively use nodes of a cluster to parallel process HUIs across multiple machines, while HUI-Miner is executed on non-parallel single machine. In Fig.5, PHUI-Growth(DLU-MR) generally runs much faster than PHUI-Growth(Baseline) on all the datasets. This is because that PHUI-Growth(DLU-MR) integrates the DLU-MR strategy for effectively prune the candidates and hence enhances its mining performance. Then, we compare the scalability of the algorithms on large datasets Chainstore×5 and T10I4N10K|D|2000K. As shown in Fig 5(c) and Fig. 5(d), PHUI-Growth(DLU-MR) has very good scalability on large datasets. On the contrary, the execution time of HUI-Miner increases dramatically on large datasets. In Fig 6(d), as the relative *min_util* is set to 0.01%, PHUI-Growth(DLU-MR) only takes about 592 seconds, while HUI-Miner takes more than 7,500 seconds.

6 Conclusion

In this paper, we propose a new framework for *mining high utility itemsets in big data*. A novel algorithm *PHUI-Growth* is proposed for efficiently parallel mining high utility itemsets from distributed data across multiple commodity computers. It is implemented on a shared-nothing Hadoop platform and thus inherits several merits of Hadoop, including easy deployment in high level language, supporting fault recovery and fault tolerance, low communication overheads and high scalability on commodity hardware. A novel strategy called *DLU-MR* is proposed to effectively prune the search space and greatly improve the performance of *PHUI-Growth*. Empirical evaluations of different types of real and synthetic datasets show that PHUI-Growth has good scalability on large datasets and outperforms the state-of-the-art algorithms.

References

1. Agrawal, R., Srikant, R.: A fast algorithms for mining association rules. In: Proceedings of VLDB, pp. 487–499 (1994)
2. Ahmed, C.F., Tanbeer, S.K., Jeong, B.-S., Lee, Y.-K.: Efficient Tree Structures for High-utility Pattern Mining in Incremental Databases. IEEE Trans. Knowl. Data Eng. **21**(12), 1708–1721 (2009)
3. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: Massive Online Analysis. JMLR (2010). http://moa.cms.waikato.ac.nz/
4. Fan, W., Bifet, A.: Mining Big Data: Current Status, and Forecast to the Future. SIGKDD Explorations **14**(2), 1–5 (2012)
5. Han, J., Pei, J., Yin., Y.: Mining frequent patterns without candidate generation. In: Proceedings of ACM SIGMOD, pp. 1–12 (2000)
6. Liu, Y., Liao, W., Choudhary, A.: Fast high-utility itemsets mining algorithm. In: Proceedings of UBDM (2005)
7. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: Proceedings of CIKM, pp. 55–64 (2012)
8. Liu, J., Wang, K., Fung, B.C.M.: Direct discovery of high utility itemsets without candidate generation. In: Proceedings of IEEE ICDM, pp. 984–989 (2012)
9. Li, Y.-C., Yeh, J.-S., Chang, C.-C.: Isolated Items Discarding Strategy for Discovering High-utility Itemsets. DKE **64**(1), 198–217 (2008)
10. Pisharath, J., Liu, Y., Ozisikyilmaz, B., Narayanan, R., Liao, W.K., Choudhary, A., Memik, G.: NU-MineBench version 2.0 dataset and technical report
11. Subramanian, K., Kandhasamy, P., Subramanian, S.: A Novel Approach to Extract High Utility Itemsets from Distributed Databases. Computing and Informatics **31**, 1597–1615 (2012)
12. Tseng, V.S., Shie, B.-E., Wu, C.-W., Yu, P.S.: Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases. IEEE Trans. Knowl. Data Eng. **25**(8), 1772–1786 (2013)
13. Vo, B., Nguyen, H., Ho, T.B., Le, B.: Parallel method for mining high utility itemsets from vertically partitioned distributed databases. In: Velásquez, J.D., Rios, S.A., Howlett, R.J., Jain, L.C. (eds.) KES 2009, Part I. LNCS, vol. 5711, pp. 251–260. Springer, Heidelberg (2009)

14. Apache Hadoop. http://hadoop.apache.org
15. Frequent itemset mining implementations repository. http://fimi.cs.helsinki.fi/
16. Microsoft Corporation: Example database FoodMart of Microsoft SQL Server Analysis Server
17. SAMOA. http://samoa-project.net
18. Vowpal Wabbit. http://hunch.net/~vw/