ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

MINING OPINIONS FROM BIG DATA OF INDONESIAN HOTEL REVIEWS

[1]VERONICA S. MOERTINI, [2]VINSENSIUS KEVIN, [3]JEANE SATYADI

Informatics Department, Parahyangan Catholic University, Bandung, Indonesia

[1]moertini@unpar.ac.id, [2]vinsensiusvey@gmail.com, [3]jsatyadi@gmail.com

ABSTRACT

Mining customer opinions from hotel reviews is useful. The results can then be used to help customers in choosing the most suitable hotel. In this research, a technique for mining opinions from big data of Indonesian hotel reviews, which is based on MapReduce, is developed. To avoid iterative computations, we adopt look-up table approach. The experiments for evaluating the performance of the technique were conducted on a Hadoop cluster with 14 clients. The results show that the proposed technique discovers useful opinion summary and is scalable.

Keywords: *mining Indonesian hotel reviews, big data, MapReduce*

1. INTRODUCTION

Along with the popularity of e-commerce transaction broker websites selling hotel rooms [17], to broaden market and reach more customers, hotels sell their rooms in few or even many websites. Each website collects its own customer reviews and provides score for every hotel. It is found that the reviews as well as the score of a specific hotel sometimes differ from a website to another, which may raise the question of which one is more trusted. To provide the summarized reviews from those websites, reviews can be crawled, collected, stored as big data and then be mined. One of the mining objectives can be obtaining customer opinions. As the opinions are mined from big data of reviews originated from many websites, it is expected that the results are more trusted or accurate. The customer opinions towards hotels can then be published by independent hotels information providers such that users are eased in choosing the most suitable hotel to stay [1].

The objective of opinion mining and summarization is to find what reviewers like and dislike towards objects being reviewed [2]. Since the number of reviews towards an object can be large, an opinion summary should be produced in such a way to support easy decision making. One known approach for mining customer reviews is features based, where the tasks include extracting object features from each review. Several techniques of extracting features from customer reviews are complex tasks that involve iterative computation (see Subsection 2.2 for more discussion).

Hadoop is an emerging platform aimed for storing and analyzing big data in distributed systems [6]. Its storage and computational capabilities scale with the addition of hosts to a Hadoop cluster and it handles volume sizes in the petabytes on clusters with thousands of hosts. Hadoop comes with master-slave architecture and consists of the Hadoop Distributed File System (HDFS) for storage and MapReduce for computational capabilities. With HDFS, large dataset is divided into blocks, distributed and replicated in the slave nodes. A MapReduce program processes data by manipulating (key/value) pairs in the following general form:

- map: (k1,v1) → list(k2,v2)
- reduce: (k2,list(v2)) → list(k3,v3).

Designed to process big data based on that key/value, MapReduce has weakness (i.e. inefficient) in iterative computations. MapReduce functions need to store the computation results into HDFS files that then be read by the functions in the next consecutive iteration (as the function inputs). With this iterative write-read, the cost of I/O is high. Given this fact, we conclude that MapReduce function is best adopted for "one-pass" computation in analyzing big data.

By studying hotel reviews written in Indonesian language (see Section 3), we find that the techniques found in literature (such as [1, 3]) can not be adopted as is, specifically for mining big data of reviews. Unlike techniques for mining English documents that have been standardized and well known, so far we only find limited research results for analyzing Indonesian documents, which are:

(1) [9] develops rules for mining public opinions of Indonesian universities, which we find too general such that it is un-applicable for mining customer reviews;
(2) [10] develops method for tagging Part-Of-Speech (POS) from Indonesian documents;
(3) [11] develops technique for extracting training corpus that can be used for sentiment analysis towards Indonesian tweets.

Hence, an enhanced method for analyzing big data of hotel reviews written in Indonesian language needs to be developed. In this research, we intend to contribute an enhanced efficient technique for mining opinions from big data of Indonesian hotel reviews, which is based on MapReduce with one-pass computation. In designing the algorithm, we avoid iterative computation (that causes high cost of I/O) and instead adopt "look up table" technique.

2. LITERATURE REVIEW

2.1 Indonesian Language

As found in other languages, Indonesian words are classified into noun, verb, adverb, adjective, prepositions and so on. These words come as root words or derived words (root words added with prefixes, suffixes or circumfixes). A word can be root word or consist of a root word with one or more affixes. In Indonesian, there are three types of affix [12, 19]:
(a) a prefix is attached before the base;
(b) a suffix comes after the base and
(c) a circumfix or confix contains two parts, one occurring before the base and one after.
However, not all of the root words can be combined with affixes.

Prefixes*:*
Prefix is an affix attached to the front of a root word that creates a new word. There are several prefixes, which are ber-, di-, me-, pe-, se- and ter. While to form new words ber-, di-, se- and ter are just attached in front of a basic word, the use of me- and pe- cause some changes to the root words.

Some example of the use of ber-, di-, se- and ter [20]:
Ber- + *bahaya* (danger) = *berbahaya* (dangerous), *ber-* + *asal* (origin) = *berasal* (originated), *di-* + *lempar* (throw) = *dilempar* (thrown by), *di-* + *nilai* (score) = *dinilai* (scored by), *se-* + *ribu* = *seribu* (one thousand), *se-* + *kelas* = *sekelas* (one class), *ter-* + *batas* (limit) = *terbatas* (limited), *ter-* + *baik* (good) = *terbaik* (the best). As a superlative ter- is always attached to an adjective while to form a passive word, it is attached to noun.

Me**-** is used to construct a verb from a noun or to indicate that the subject of a statement is the one doing the action of the verb. Me- has six variations that are used depending on the first letter of the root word which they are attached as follows:
a) Me- is used when the following word is started with l, m, n, r, w, y;
b) Meng- is used when the following word is started with g, h, k or vowels;
c) Mem- is used when the following word is started with b f p;
d) Men- is used when the following word is started with c, d, z, and t;
e) Meny- is used when the following word is started with s;
f) Menge- is used when the following word is consisted from one syllable only.

Pe**-** has variations of pe-, pem- and pen. They are used to form a noun that indicates a person or thing that do the verb or to form a noun that has the quality or attribute inherent in the adjective. Example: *pe-* + *makan* (eat) = *pemakan* (eater), *pem-* + *panas* (heat) = *pemanas* (heater), *pen-* + *dingin* (cool) = *pendingin* (cooler).

Suffixes*:*
Indonesian suffixes are -kan, -i, -an and –nya. They can form a noun, soften a command or add politeness, direct the action and derives causatives or adjectives.
Example [21]: *milik* (belong) + *-nya* = *miliknya* (belong to), *bulan* (month) + *-an* = *bulanan* (monthly).

Circumfix*:*
Indonesian circumfixes form nouns from adjectives, causatives from verbs, form nouns from verbs. They are ke-…-an, me-…-kan, pe-...-an and per-...-an.
Example: *ke-...-an* + *puas* (satisfy) = *kepuasan*, *me-...-kan* + *mandi* (to bath) = *memandikan*, *pe-...-an* + *buka* (open) = *pembukaan*.

Particles*:*
There are particles of lah, pun, kah, per. Particle '-lah' is always attached to the preceding word. It is to mark the predicate when the predicate is out of its normal position. It is never obligatory. Basically it adds polite emphasis.
Example: *tidak* + *-lah* = *tidaklah*.

Particle 'pun' can act as a focusing adjunct. It identifies the most important thing involved in what being said.
Example: *walau* + *pun* = *walaupun*. Partikel 'kah' is written as one word with the word that is followed. Example: *mau* + *kah* = *maukah*.

Indonesian Negation:
Indonesian negation turns an affirmative statement (for example, I am happy) into its opposite (I am not happy). The negation can be made by placing "no" before the main verb.
Example: *tidak, bukan*.

Stoplist:
Stoplist can be defined as a list of words where for some special reason should be ignored or bypassed by a particular data processing operation. In terms of mining sentiments, there are a number of Indonesian words that can be classified as stoplist. Some example are: *agak, agar, akan, akankah, akhir, jika, jikalau, juga, justru, kalau, sekaligus, sekalipun*, and so on.

2.2 Mining Customer Reviews

The objective of opinion mining and summarization is to find what reviewers liked and disliked towards objects being reviewed [2]. Since the number of reviews on an object can be large, an opinion summary should be produced such that this can be visualized and compared by users for making decision (such as selecting the right hotels).

One popular approach for mining reviews is features based. The tasks include [2]:
(1) Extract object features from each review;
(2) Determine whether the opinions on the features are positive, negative or neutral;
(3) Group feature synonyms and produce a summary.
There are a number of methods for extracting features, such as:
(1) Co-occurrence association-based method, which aims to extract implicit features in customer reviews [3]. The computation for finding features need matrices of words (stored in memory) which can be large size;
(2) Finding frequent item sets using association rules mining, where an item set is a set of words or a phrase that occurs together [4] and [5]. The algorithms (apriori, FP-growth, etc.) are iterative and need to access the dataset at least two times.

For identifying opinion orientation on features, [2] proposes the following technique:
For each feature, identify the sentiment or opinion orientation expressed by a reviewer. The work is performed on each sentence as a sentence may contain more than one feature.
For instance: The battery life (feature) and picture quality (feature) are great (+ or 1), but the view founder (feature) is small (- or -1). In identifying the feature (f) opinion, there are two steps:
(1) Split the sentence (into s_f) if needed based on "but" words;
(2) Find f opinion in s_f . The opinion of f is an element of {1, -1, 0}. Then the orientation of s_f are sum up accordingly.

2.3 Hadoop, HDFS and Map-Reduce

As excerpted in [18], the distributed systems framework Hadoop is useful for storing and analyzing big data [2]. Hadoop comes with master-slave architecture and consists of the Hadoop Distributed File System (HDFS) for storage and MapReduce for computational capabilities. Its storage and computational capabilities scale with the addition of hosts to a Hadoop cluster. The following is some brief overview of HDFS and MapReduce.

HDFS: HDFS is a distributed file system designed for large-scale (up to petabytes) distributed data processing under frameworks such as MapReduce and is optimized for high throughput. HDFS automatically divides the data into blocks, replicates data blocks and stored on nodes (the default is 3 replications).

MapReduce: MapReduce is a data processing model that has the advantage of easy scaling of data processing over multiple computing nodes. Map and Reduce functions run in every slave node. While Map takes input or read the local blocks, Reduce inputs are the outputs of Map functions run on nodes sent through the network in a Hadoop cluster. A MapReduce program processes data by manipulating (key/value) pairs in the general form:

map: (k1,v1) ➞ list(k2,v2)

reduce: (k2,list(v2)) ➞ list(k3,v3).

Map reads (key, value) pairs from dataset stored as HDFS blocks, then based on the functions designed by developers, it generates one or more output pairs list (k2, v2). Through a shuffle and sort phase, the output pairs are partitioned and then transferred to reducers via the network. Pairs with the same key are grouped together as (k2, list(v2)). One reducer run in a node processes one value of k2 (with its list(v2)), then it generates the final output pairs list(k3, v3) for each group based on the designed reducer algorithm.

The overall MapReduce processed is shown in Fig. 1 [2, 10]. A client submit a job to the master, which then assign and manage Map and Reduce job parts to slave nodes. Map will read and process blocks of files stored locally in the slave node. The Map output of pair key-values are sent to Reduce function for further process.

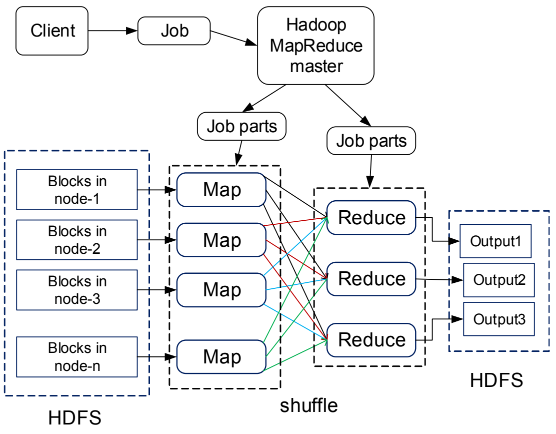

Figure 1: MapReduce processes [18].

3. INDONESIAN HOTEL REVIEWS

By collecting, observing and studying hundreds of hotel reviews, we found:
(1) Most of the reviews are expressed with "daily language", which are informal, do not follow the correct grammar, have many abbreviations and slangs (non-formal words): Many "sentences" are actually phrases that contains nouns, verbs, adjective or adverbs only; Many reviews consist of brief "key words" (only) that represent customer opinions;
(2) As lots of reviews are written using mobile devices, there are many word abbreviations;
(3) The customers care and concern with limited "things" or objects only; Those "things" are the overall hotel, room, facilities, services, access to the hotel, location, staffs, room rate/price, AC and meal;
(4) Other than stop words, reviews may also contain meaningless terms for analysis purposes.

Example of slangs or abbreviations found with their (should be) standard words: *nggak-tidak, ga-tidak, price-harga, srpan-sarapan, bkn-bukan, kmr-kamar, room-kamar, hrg-harga, cpt-cepat, view-pemandangan* and *mantab-mantap*. Some of those slangs or abbreviations are important for understanding customer opinions.

Given the rules of Indonesian grammar and those fact findings, preprocessing the reviews is an important step with the aims of:
(1) Transforming slangs and abbreviation words into the standard words;
(2) Removing stop words and meaningless terms;
(3) Stemming or obtaining "root words" as few different derived words may be originated from one root word.

The other issue that actually needs to be resolved is "grammar fixing" such that every sentence follow the correct Indonesian grammars. However, we find that this is very challenging. If a "sentence" contains only predicate-object, it is hard to define the subject. Likewise, if the "sentence" or phrase contain adverb only, for example "*sangat lama*", it is difficult to find its pair of words. For these reasons, we have not performed this task.

4. PROPOSED TECHNIQUE

We propose two techniques for mining Indonesian hotel reviews, which are:
(1) Finding customer sentiments (positive, neutral or negative) towards specific hotel features;
(2) Counting words that express customer opinions towards hotels.
The second is proposed by considering that lots of Indonesian reviews contains "key words" of customer opinions only. These techniques are enhancement of our previous works in [13] and [14].

To speed up computation, we adopt look up table method. Here we assume that the data needs to be looked up can be produced manually or using the techniques that have been developed (such as [10, 11]).

4.1 System Architecture

The system architecture is depicted in Figure 2 where each process is described as follows:
(1) Crawl reviews: Collecting hotel reviews from e-commerce websites selling hotel rooms;
(2) PreProc MapRed: Preprocessing the raw reviews by cleaning, finding root words and labeling words with their types;
(3.a) CountWordOp MapRed: Simple method of review analysis by counting opinion words;
(3.b) AnalSent MapRed: Analyzing customers opinion towards hotel features (corresponding to hotel location, rooms, food, services, etc).

Here, the one-pass computation is materialized by looking up few HDFS files that store root words, type of root words, the standard words of slangs, hotel features and the related words orientation.

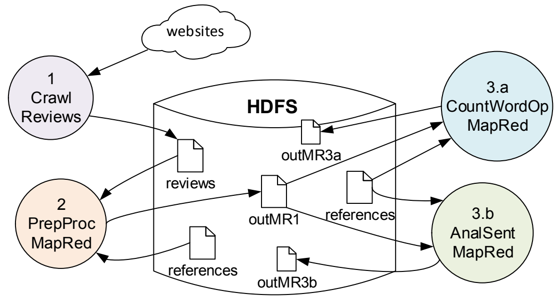

Figure. 2. The proposed system architecture.

4.2 Preprocessing MapReduce Component

The MapReduce performing data preprocessing contains two main steps, which are cleaning and labeling/tagging terms. The cleaning process involves eliminating stop words, transforming slangs and stemming (finding root words). The clean terms are then labeled with its type, which are verb, adverb, adjective, etc. (Figure 3). The references stored as HDFS (to be looked up) are:
(a) *stoplist.txt*: containing list of stop terms used to eliminate the stop words [10];
(b) *slangs.txt*: containing list of pair of slang-standard words used to transform slangs into standard words;
(c) *root_words.txt*: containing list of root words and their type (verb, adverb, adjective, etc.) used to find the root word of every non-stop word and label it with its type.

The following are the Map and Reduce algorithms.

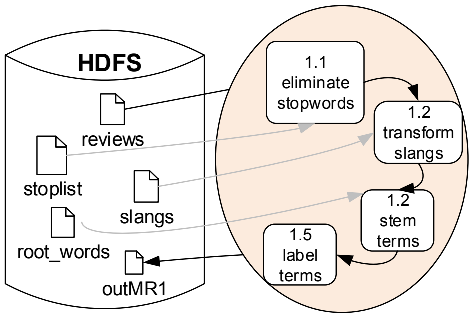

Figure 3. Preprocessing MapReduce.

Map performs:
(1) Eliminating stop words: words listed in the *stoplist.txt* are discarded;
(2) Transforming slangs: every slang listed in *slangs.txt* are replaced with its standard word;
(3) Stemming: if words are already listed in the *root_word.txt*, nothing is performed. Otherwise, the algorithm parses the words based on Indonesian rules for obtaining the root words, discard prefix(es) and/or suffix(es) to find the root word (see Table 1);
(4) Tagging or labeling root words with their types (we adopt the method in [10]).

The algorithm is as follows:

```
Method: map
Input: <key, value>, key: hotel Id, value: review texts;
Output: <key', value'>, key': hotel Id, value: cleaned review texts;
Steps:
(1) line ← value;
(2) hotelId ← get hotelId from line;
(3) 2: line ← eliminate stop words from line;
(4) line ← transform slangs from line;
(5) 4: reviewHotel ← call stemmingMethod then addTypeRootWord;
(6) 5: key' ← hotel Id; value' ← reviewHotel;
(7) 6: emit <key!, value!>
```

The stemming algorithm, which must address the issues discussed in Section 3, is as follows:

```
Method: stemmingMethod
Input: word, rootWord;
Output: word: the root word of word
Steps:
(1) if word is not root word then:
    (a) word ← delete particles (“-kah”, “-lah”, “-pun”) from word, if exists;
    (b) word ← delete possessive pronoun ("-ku", "-mu", "-nya"), if exists;
    (c) word ← delete first prefixes (if exist) from word by applying rules in Table 1;
    (d) word ← delete second prefix ("-ber-", "-bel-", "-be-", "-per-", "-pe-", "-pel-", "se-"), if exists;
    (e) word ← delete suffix ("kan", "-i", "-an") from word, if exists;
(2) return word
```

Table 1: Rules applied in deleting prefixes.

If word contains prefix	Stemming action
"meng-" followed by vocal letter of "e" or "u"	delete "meng-" and replace "e" / "u" with "k"
"meny-"	replace "meny-" with "s"
"men-"	delete "men-"
"mem-" followed by vocal letter of "a", "i", "u", "e" or "o"	delete "mem-" and replace "a", "i", "u", "e" or "o" with "p"
"me-"	delete "me-"
"peng-" followed by vocal letter of "e" or "a"	delete "peng-" and replace "e" or "a" with "k".
"peny-"	replace "peny-" with "s"
"pen-" followed by vocal letter of "a", "i", "u", "e" or "o"	delete "pen-" and replace "a", "i", "u", "e" or "o" with "t"

"pem-" followed by vocal letter of "a", "i", "u", "e" or "o"	delete "pem-" and replace "a", "i", "u", "e" or "o" with "p"
"di-"	delete "di-"
"ter-"	delete "ter-"
"ke-"	delete "ke-"

After the preprocessed reviews emitted by Map are sorted based on hotelId (through shuffle process), Reduce simply write them to the output HDFS file as follows:
Method: reduce
Input: <key, values>, key: hotelId, values: list of cleaned review texts for hotelId;
Output: <key', values'>, key': hotelId, values: cleaned review texts for hotelId;
Steps: write every key and values into outMR1.txt file

An example input and output for Preproc MapReduce:
Input: *tidak susah utk mencari makanan. kamar bersih rapi.*
Output: *tidak*/adv *susah*/adj *untuk*/pre *cari*/v *makan*/v /./ *kamar*/n *bersih*/adj *rapi*/adj /./

4.3 Analysis Sentiment (AnalSent) MapReduce Component

This MapReduce takes input of the HDFS file produced by Preprocessing MapReduce (Figure 4). It identifies hotel features, find the opinion/sentiment (positive, neutral, or negative) of every feature, then count the frequency of every sentiment for every feature. The summarized sentiment for every hotel are also computed. For identifying the customer sentiment (positive, neutral, or negative) towards every feature, we adopt the techniques discussed in [2].

There are two HDFS files that are looked up by this component, which are:
(a) *features.txt*: Containing hotel frequent features that are mostly "judged" by customers. We found 19 features, which are: location, access, parking, staff, service, price, room, bath-room, shower, amenities, ac, tv, wifi, breakfast, food, restaurant, facilities, pool and (overall) hotel.
(b) *orientation.txt*: Containing list of pair of two words, the first is the word frequently used by customers to express their opinion and the second is the sentiment (positive, neutral or negative).

We find that the first could be adjective (mostly), noun or verb. There are total of 58 pairs that we identified from the case study of reviews. Some examples are (written in Indonesian): *strategis-pos, mudah-pos, sulit-neg, mahal-neg, bersih-pos, mantap-pos, biasa-neut, jorok-neg,* and *bising-neg*.

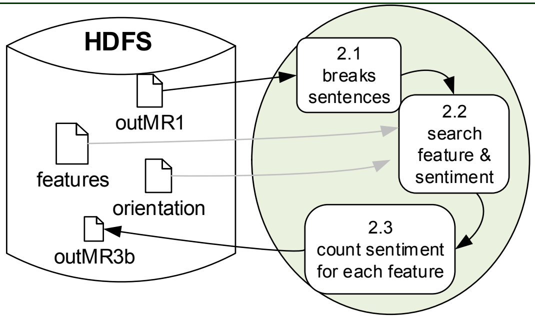
HDFS
outMR1
features
orientation
outMR3b
2.1 breaks sentences
2.2 search feature & sentiment
2.3 count sentiment for each feature

Figure 4: Analysis Sentiment MapReduce.

After examining the preprocessing results, we find that the structure of the feature and its sentiment can be:
(a) noun – adjective, for examples: *lantai/n kotor/adj, shower panas/adj, ac panas/adj, hotel/n nyaman/adj, hotel/n murah/adj, fasilitas/n baik/adj;*
(b) noun – verb, for examples: *kamarmandi/n jorok/v;*
(c) verb – adjective, for examples: *layan/v puas/adj.*
Those structures are used in this Map algorithm in finding customer sentiments toward features.
The Map and Reduce algorithms are discussed below.

By taking the input of pair of hotelId and labeled root words (such as *tidak*/adv *susah*/adj *untuk*/pre *cari*/v *makan*/v /./ *kamar*/n *bersih*/adj *rapi*/adj /./), this Map emits pair of key'-value' where key' = hotelId and value' = *feature-found-1 ctr_pos ctr_neut ctr_neg feature-found-2 ctr_pos ctr_neut ctr_neg feature-found-3 ctr_pos ctr_neut ctr_neg. .. . . . feature-found-n ctr_pos ctr_neut ctr_neg*, where *ctr_pos* denotes count of positive sentiment, *ctr_neut* denotes count of neutral sentiment, *ctr_neg* denotes count of negative sentiment.

An example of Map output is: Hbdg2290 | hotel 1 0 0 fasilitas 1 0 0 layan 1 0 0 /./

Method: map()
Input: <key,value> with key: hotelId, value: cleaned and labeled review, hotelFeatures, rootWords, adjWordSentiments;
Output: <key',value'> with key': hotelId, value': text of list of pair feature - ctr_pos ctr_neut ctr_neg;
Steps:

```
1: line ← value; hotelId ← get hotelId from line;
listFeatureCounter ← null
2: sentences[] ← break review into sentences by splitting
line using "/./"
3: foreach sentence in sentences[] do
4:     ctr_pos = 0; ctr_neut = 0; ctr_neg = 0; //counter of
positive, neutral and negative sentiment
5:     features[] ← get features from sentence // Get hotel
features (looking up hotelFeatures)
```

```
6:    if features is not empty then foreach feature in features[] do
7:        wordAdj ← find the closest adj or verb root word at the right side of feature  in sentence
8:        if wordAdj is not empty do
9:          wordAdjSent ← find sentiment of wordAdj (pos, neut or neg) by looking up to adjWordSentiments
10:           if wordAdjSent is not empty do wordAdjSent ← oppose the sentiment if negation word found in front of wordAdj
11:           increment counter ctr_pos or ctr_neut or ctr_neg accordingly
12: if feature is not in listFeatureCounter do listFeatureCounter ← add feature + ctr_pos + ctr_neut + ctr_neg
13:           else add ctr_pos + ctr_neut + ctr_neg to the same  feature in listFeatureCounter
14: resultReviewHotel  ← text of listFeatureCounter
15: emit < hotelId, resultReviewHotel >
```

Reduce is designed to compute the over all sentiments for every hotel and generate output with the format: *hotelId | feature-found-1 ctr_pos ctr_neut ctr_neg feature-found-2 ctr_pos ctr_neut ctr_neg. .. . . . feature-found-n ctr_pos ctr_neut ctr_neg TOTAL_COUNT ctr_pos ctr_neut ctr_neg.* An example Reduce output is:
h2290 | harga 2 0 0 kamarmandi 0 1 0 kamar 0 2 0 TOTAL_COUNT 2 3 0

```
Method: reduce()
Input:  key, values as list of value with key: hotelId, value: list of <feature - ctr_pos  ctr_neut ctr_neg>, feature_weights
Output: <key',value'> with  key': hotelId, value': list of <feature - ctr_pos  ctr_neut ctr_neg>
Steps:
1: featuresCounter ← HashMap<String, ArrayList<Integer>>// for storing hotel features and ctr_pos  ctr_neut ctr_neg
2: for each value in values do
3:    for each feature in value do
4:      if feature not found in featuresCounter do get ctr_pos,  ctr_neut and ctr_neg from value then add to feature and ctr_pos,  ctr_neut and ctr_neg at the bottom of featuresCounter
5:      else do get ctr_pos,  ctr_neut and ctr_neg from value then sum up ctr_pos,  ctr_neut and ctr_neg to the existing value of ctr_pos,  ctr_neut and ctr_neg in featuresCounter
6: hotel_score ← using ctr_pos,  ctr_neut and ctr_neg compute the total count of positive, neutral and negative sentiments
7: results ← text of list of feature - ctr_pos ctr_neut ctr_neg  from featuresCounter + string of  hotel_score
8: write (key', results)
```

4.4 Count Opinion Words MapReduce Component

This MapReduce reads the output of review preprocessing results and orientation words listed in orientation.txt files. For every hotelId, this simply computes the count of every orientation word found in the preprocessed review, sums up the count and write pair of *hotelId – {orientation-word count}* into a HDFS file.

The following is the detailed algorithm of Map and Reduce function.

```
Method: map()
Input:  <key,value> with key: hotelId, value: cleaned and labeled review, orientations[]
Output: <key',value'> with  key': hotelId, value': text of list of pair opinion_word - 1
Steps:
1: line ← value
2: hotelId ← hotelId from line;
3: sentences[] ← break review into sentences by splitting line using "/./"
4: outText ← "" //empty text
5: foreach sentence in sentences[] do
6:    opWords[] ← get opinion words from sentence by looking up to orientations[], including the negation word if exists
7:    if opWords is not empty then
8:       foreach word in opWords [] do
9:           outText ←  word + "1 "
10: emit < hotelId, outText >
```

```
Method: reduce()
Input:  <key,values> with key: hotelId, value: list of pairs of word - 1
Output: <key',value'> with  key': hotelId, value': text of list of pair opinion_word - count
Steps:
1: opWordCounts ← [] // initialize array of pair of word-count
2: for each value in values do
3:    if word is found in opWordCounts, increase its count by 1
4:    else add element of word – 1 at the end of opWordCounts
5: outText ← text of all pairs of  word-count found in opWordCounts
6: write (key, outText)
```

5. EXPERIMENTS

We have implemented the component of review preprocessing and sentiment analysis. To speed up the look up processes (from file references, such as basic words, features, etc.), we adopt HashMap function. Here, we conduct 2 sets of experiments, aiming to evaluate the accuracy and scalability or speed. The experiments were performed in a Hadoop cluster having 1 master and 14 slave machines. The machines specification: the processor is Quad-Core running at 3.2 GHz, the memory of 6 machines is 4 Gbyte and of 8 machines is 6 Gbyte. The HDFS block size is configured as 128 Mb and replicated 3 times.

Accuracy: As there is no labeled (with its opinions) benchmark review dataset that we can be compared, in this experiment we present the results of qualitative observations of the mining results.

We gathered some small sample dataset containing 40127 reviews given to 28 hotels from 2 well known Indonesian e-commerce websites selling hotel rooms. Examples of the analysis results are provided in Table 2. Among the 28 hotels, we find that most have positive reviews. Only one hotel (h2017) that has high percentage of negative/positive (54%), where the over all room condition and facilities are unsatisfying, while the food and service are quite acceptable.

Table 2: Example of Analysis Results.

HotelId	#Pos	#Neutral	#Neg	(Neg/Pos)%
h1001	390	34	56	14
h1002	374	24	66	18
h1003	417	16	67	16
h2017	37	3	20	54
h2026	750	38	141	19

An example of the sentiment summary of a hotel is as follows:
| h1001| akses 1 0 1 lokasi 51 0 6 posisi 3 0 0 tempat 3 1 1 hotel 95 8 8 harga 24 2 5 kamar 57 7 14 lantai 2 0 0 shower 1 0 0 amenities 1 0 0 tv 2 0 0 ruang 1 0 0 fasilitas 18 1 0 air 1 0 5 wc 0 2 0 toilet 1 1 1 kamarmandi 7 0 2 ac 1 0 0 wifi 4 1 1 suara 0 0 1 restoran 5 0 0 sarapan 14 2 3 makan 20 1 3 menu 4 4 0 suasana 2 1 1 kolamrenang 9 1 0 staff 7 0 0 resepsionis 1 0 0 layan 53 1 2 lobby 2 1 1 check-out 0 0 1 TOTAL_COUNT 390 34 56.
The interpretation: *The hotel location is good, the price is right, the room, food and services are also satisfying*. The negative opinions (14%) are directed towards hotel location, overall hotel condition, price, room, food and services.

We compare the sentiment summary of many hotels with the quantitative reviews provided in the 2 e-commerce websites and find that among 28 hotel summaries, 22 of these are highly inline and 6 are somewhat inline with those presented in the websites. Hence, we conclude that the proposed methods succeed in mining customer opinions with high accuracy.

Speed and Scalability: We use review dataset with the size from 1.3 to 20.23 gigabytes. The execution time is shown in Figure 4 and Figure 5. The plots are linear, which indicates that the scalability of the proposed technique is acceptable. The total time for preprocessing 20.23 gigabytes of review is 2161 seconds (36.02 min), while analyzing the sentiment of 18.47 gigabytes (the output of preprocessing) is 619 seconds (10.32 min). Hence, the algorithms are fast. The time needed for analyzing is approximately one quarter of preprocessing. Our analysis: The elimination of stop words and finding root words consume more I/O and look up process (as it reads 2 files with larger size) compared to the analysis component.

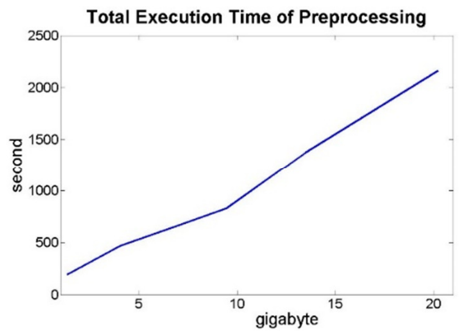

Figure 4: The Execution Time of Review Preprocessing.

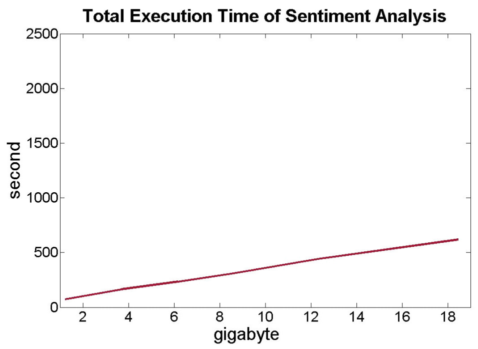

Figure 5: The Execution Time of Sentiment Analysis.

6. CONCLUSION AND FURTHER WORKS

The proposed technique have succeeded in mining opinions from big data of Indonesian hotel reviews. By performing qualitative evaluation, we find that the accuracy is high. With adopting look up tables, the algorithms execution times in the Hadoop cluster proves to be fast and guarantees scalability.

There are issues that have not been addressed, such as fixing grammar in sentences, developing techniques for handling slangs and finding hotel features automatically. Integrating the proposed technique with other data analysis technique (such as [15, 16]) in the e-commerce CRM system [17] is also necessary such that the analysis results can be used to support customer decision makings.

7. ACKNOWLEDGMENT

We like to thank to the Directorate General of Higher Education of Ministry of Research, Technology and Higher Education of the Republic of Indonesia who is funding this research in 2016-2017 through Hibah Bersaing scheme with contract number of III/LPPM/2017-04/130-P.

REFERENCES

[1]. E. Bjørkelund, T. H. Burnett, K. Nørvåg, "A study of opinion mining and visualization of hotel reviews", Proc. iiWAS2012, 3-5 Dec., 2012, Bali, Indonesia.

[2]. B. Liu, Opinion Mining & Summarization - Sentiment Analysis, Tutorial, April 21, 2008 in Beijing. http://www.cs.uic.edu/~liub

[3]. Y. Zhang and W. Zhu, "Extracting implicit features in online customer reviews for opinion mining", Proc. WWW2013 Rio de Janeiro, Brazil, 2013.

[4]. J. Wang and H. Ren, Feature-based Customer Review Mining, Dept. of Computer Science, Stanford University, 2006.

[5]. S. H. Ghorashi, R. Ibrahim, S. Noekhah and N. S. Dastjerdi, "A frequent pattern mining algorithm for feature extraction of customer reviews", International Journal of Computer Science Issues, Vol. 9, Issue 4, No 1, July 2012.

[6]. A. Holmes, Hadoop in Practice, USA: Manning Publ., 2012.

[7]. C. Lam, Hadoop in Action, USA: Manning Publ., 2010.

[8]. E. Sammer, Hadoop Operations, USA: O'Reilly Media, Inc., 2012.

[9]. I. F. Rozi, S. H. Pramono, and E. A. Dahlan, "Implementasi opinion mining (analisis sentimen) untuk ekstraksi data opini publik pada perguruan tinggi," Jurnal EECCIS, vol. 6, no. 1, pp. 37-43, 2012.

[10]. A. Wicaksono and A. Purwarianti, "HMM based pos tagger for bahasa indonesia," in Proc. of 4th Intl. MALINDO (Malay-Indonesian Language) Workshop, Jakarta, Indonesia, 2nd August, 2010.

[11]. A. F. Wicaksono, C. Vania, B. Distiawan T., M. Adriani, "Automatically building a corpus for sentiment analysis on Indonesian tweets", Proc. 28th Pacific Asia Conf. on Language, Information and Computation, Phuket, Thailand, 2014, pp. 185–194.

[12]. D. N. Djenar, A Student's Guide to Indonesian Grammar, Oxford University Press, Australia, 2005.

[13]. J. Satyadi, Pengembangan Aplikasi untuk Menambang dan Meringkas Data Revuew Pelanggan pada Website E-Commerce Penyedia Kamar Hotel, Informatics Dept., Parahyangan Catholic Univ., Indonesia, 2015.

[14]. V. Kevin, Peringkasan dan Penambangan Data Review Pelanggan pada Sistem Terdistribusi Hadoop, Informatics Dept., Parahyangan Catholic Univ., Indonesia, 2016.

[15]. V. S. Moertini and L. Venica, "Enhancing parallel k-means using map reduce for discovering knowledge from big data", Proc. 2016 Intl. Conf. on Cloud Computing and Big Data Analysis (ICCCBDA 2016), pp. 81- 87, Chengdu China, 5-7 July.

[16]. V. S. Moertini, N. Ibrahim and Lionov, "Efficient techniques for predicting suppliers churn tendency in e-commerce based on website access data", Journal of Theoretical and Applied Information Technology, vol. 74, no. 3. pp. 300-309, 2015.

[17]. N. Ibrahim, V. S. Moertini and Verliyantina, "Supplier relationship management model for SME's e-commerce transaction broker case study: hotel rooms provider", Journal of Theoretical and Applied Information Technology, vol. 71, no. 1, pp. 61-70, 2015.

[18]. V. S. Moertini, L. Venica, "Parallel K-Means for Big Data: On Enhancing Its Cluster Metrics and Patterns", Journal of Theoretical and Applied Information Technology, vol. 95, no. 8. pp. 1844-1857, 2017.

[19]. http://www.bahasakita.com/about/grammar/suffix/ (accessed: 12 December 2016)

[20]. http://mylanguages.org/indonesian_prefixes.php (accessed: 12 December 2016)

[21]. http://langhub.com/en-id/indonesian-grammar/147-suffixes (accessed: 12 December 2016)