Parallel Methods for Evidence and Trust based Selection and Recommendation of Software Apps from Online Marketplaces

Lahiru S. Gallege and Rajeev R. Raje
Department of Computer and Information Science
Indiana University-Purdue University Indianapolis, Indianapolis, IN USA.
Email: {lspileth, rraje}@iupui.edu

ABSTRACT

With the popularity of various online software marketplaces, third-party vendors are creating many instances of software applications ('apps') for mobile and desktop devices targeting the same set of requirements. This abundance makes the task of selecting and recommending (S&R) apps, with a high degree of assurance, for a specific scenario a significant challenge. The S&R process is a precursor for composing any trusted system made out of such individually selected apps. In addition to feature-based information, about these apps, these marketplaces contain large volumes of user reviews. These reviews contain unstructured user sentiments about app features and the onus of using these reviews in the S&R process is put on the user. This approach is ad-hoc, laborious and typically leads to a superficial incorporation of the reviews in the S&R process by the users. However, due to the large volumes of such reviews and associated computing, these two techniques are not able to provide expected results in real-time or near real-time. Therefore, in this paper, we present two parallel versions (i.e., batch processing and stream processing) of these algorithms and empirically validate their performance using publically available datasets from the Amazon and Android marketplaces. The results of our study show that these parallel versions achieve near real-time performance, when measured as the end-to-end response time, while selecting and recommending apps for specific queries.

CCS CONCEPTS

• **Information systems ~ Trust** • *Information systems ~ Recommender systems* • Social and professional topics ~ Software selection and adaptation

KEYWORDS

Software Selection and Recommendation; Algorithm Parallelization; Sentiment Analysis; Subjective Logic.

1. Introduction: TruSStReMark

The S&R process typically is ad-hoc, manual and most often uses the average star-rating system [1,2]. In our previous works [3,4,5,6,7,8], we intoduced a framework, named as TruSStReMark Trust-based Service Selection and Recommendation for Marketplaces, which contains techniques to model, quantify, specify, and monitor the trust of software apps. We have also provided methods to analyze, and aggregate, external reviews of software apps and used them to perform trust-based service selection, and recommendation (S&R). The TruSStReMark enhances the two prevalent approaches (i.e., CBF [14] and CLF [15]) by incorporating insights from user reviews. Our previous trust-based S&R algorithm (EbRank*n*Rec) [6,7,8] has two variations: (1) Eb-CBF-Rank (which integrates evidence-based techniques into the CBF for selection) and (2) Eb-CLF-Rec (which integrates evidence-based techniques into the CLF for recommendation). In this paper, we describe two parallel versions of these two previous algorithms – these versions are: i) batch processing versions (i.e., bt-pEb-CBF-Rank and bt-pEb-CLF-Rec, which are based on MapReduce ecosystem) and (2) stream processing versions (i.e., st-pEb-CBF-Rank and st-pEb-CLF-Rec, which are based on Sparks and Sparks Streaming ecosystems). As these parallel techniques use the principles of Subjective Logic (SL) [10], Sentiment Analysis (SA) [11], MapReduce Echo-System (MR) [16], and Spark [17] and Spark Streaming (ST) technologies [18].

1.1 pEbRanknRec-Batch Processing with Hadoop

The improved TruSStReMark framework proposed in this paper gathers and analyses external user reviews generated by online marketplaces – these reviews due to their sheer size are now considered as a big data challenge. The improved TruSStReMark framework parallelizes and enhances the two prevalent approaches (based on CBF and CLF) as shown in Figures 1 and 2. The goal of these two enhanced techniques is to improve the performance and confidence while searching and recommending software services by parallelizing the previous algorithms [8] of aggregating external evidences available as textual reviews. First, the parallel algorithms maintain a list of named entities for each user (i.e., a profile of important service QoS feature keywords). It performs the improved evidence-based search algorithm (Figures 1 to 4) in parallel as follows: Each user query for items (or services or apps) is tagged with associated evidences of these feature vectors calculated from users' reviews about other items. Given a review dataset from a marketplace, which includes service descriptions, user-service

This is the author's manuscript of the article published in final edited form as:

Gallege, L. S., & Raje, R. R. (2017). Parallel Methods for Evidence and Trust Based Selection and Recommendation of Software Apps from Online Marketplaces. In Proceedings of the 12th Annual Conference on Cyber and Information Security Research (p. 4:1–4:8). New York, NY, USA: ACM. https://doi.org/10.1145/3064814.3064819

relations, and set of reviews by the users, the pEbRank*n*Rec performs dataset transformations to partition the dataset randomly (possible due to the review independence property) such that HDFS could handle the correct service spaces, user spaces and user-service spaces for each mapper. For each service inside each mapper, the basic content-based algorithm calculates a vector (of size k keywords) by parsing the service descriptions using a NLP technique such as TF-IDF (all data structures are cached and reused if no updates detected). Additionally, the pEbRanknRec augments the process by parsing textual reviews from users to calculate the sentiment expressed by each user's about the targeted QoS properties of services. These QoS values for services can be calculated from the keywords, from service descriptions and from user reviews. Using the TextBlob library [19] for each sentence, it calculates the sentiments of user reviews and caches them. SL provides the necessary operators to aggregate different opinions about a service QoS values from different users depending on the situation. The conjunction, consensus and ordering operators (from the subjective logic) [10] are used in pEbRanknRec algorithms. The positive polarity of SA is a measure of belief in the textual evidences and similarly, the negative polarity is a measure of disbelief. Then naturally, subjectivity is the indication of certainty when high and uncertainty when low. Therefore, without the loss of generality we adapt the following linear conversion from SA to SL, described below in subsection 2.2.1, for all the calculations done inside the pEbRanknRec.

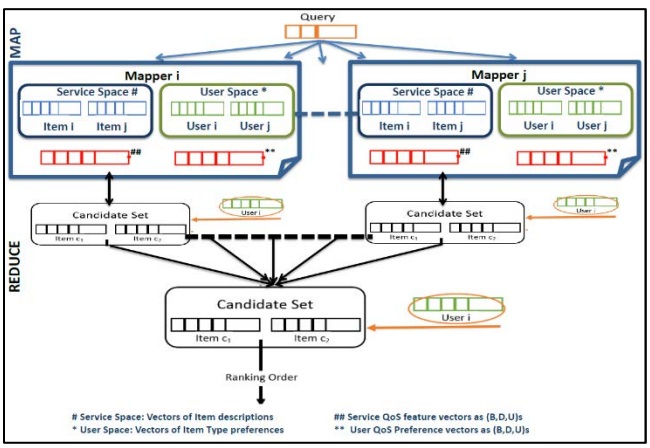

Figure 1. Parallel evidence-based CBF ranking (pEb-CBF-Rank)

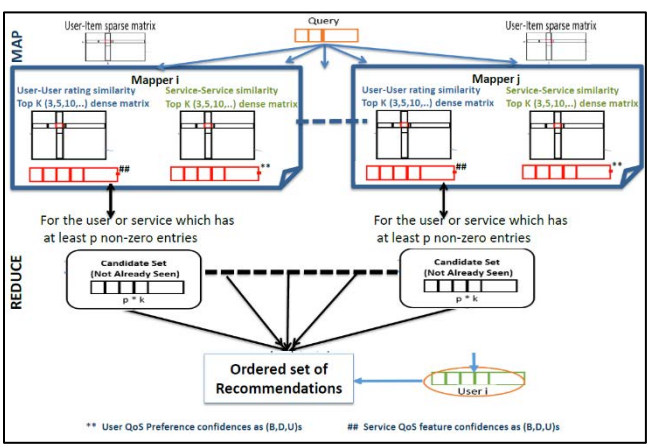

Figure 2. Parallel evidence-based Collaborative Filtering (CLF) recommendations (pEb-CLF-Rec)

1.2 Conversion of SA to SL

Table 1 presents four sample sentiments calculated from randomly selected but associated review sentences. The TruSStReMark identifies a set of named entities for users from storing their queries (i.e., which QoS features are important to them). These named entities for apps can be calculated from the keywords, their description, and from user reviews. Given that, and using TextBlob library, the TruSStReMark calculates the sentiments of user reviews. We map the values in Table 1 to the corresponding SL tuples using the process described below. The boundary cases in this conversion process are computed as listed in Table 2. The in-between values are equally weighted and divided among the relevant entries.

Table 1. General and QoS sentiments and scores from reviews [7,8]

Sample Review Sentences	Named Entity	Sentiment Analysis	
		Polarity	Subjectivity
Nice, Good Work.	General	+0.700	0.875
Very bad.	General	-0.350	0.600
I love this apps' UI with large text and figures.	User Interface	+0.357	0.514
I hate this app, it drains the battery so fast and slows down my device.	Resources Usage	-0.252	0.596

Table 2. Conversion from Sentiments (Subjectivity, Polarity) tuples to < Belief, Disbelief, Uncertainty > tuples

(P,S)	<B,D,U>		(P,S)	<B,D,U>
(+1,1)	<1,0,0>		(0,0)	<undefined>
(+0.75,0.25)	<0.75,0,0.25>		(-0.25,0.25)	<0,0.25,0.75>
(+0.5,0.5)	<0.5,0,0.5>		(-0.5,0.5)	<0,0.5,0.5>
(+0.25,0.25)	<0.25,0,0.75>		(-0.75,0.25)	<0,0.75,0.25>
(0,0)	<undefined>		(-1,1)	<0,1,0>

Table 3. Matrix representation of the SSA to SL conversion

A = [1, 1 0.75, 0.25 0.5, 0.5 0.25, 0.25 -0.25, 0.25 -0.5, 0.5 -0.75, 0.25 -1, 1]	Y = [1, 0, 0 0.75, 0, 0.25 0.5, 0, 0.5 0.25, 0, 0.75 0, 0.25, 0.75 0, 0.5, 0.5 0, 0.75, 0.25 0, 1, 0]

Let A be a matrix in which each row is a case of P, S combination, and the first column is P, and the second column is S. For example, from the table, A should be represented in matrix format as indicated in Table 3. Let Y be a matrix in which each row is the corresponding case of B, D, U combination, and the 1st column is B, 2nd column is D and the 3rd column is U. From Table 3, Y should be derived from direct conversion of A (we just need B and D in Y since B, D, U are linearly dependent such that B+D=1-U). Then, the SA to SL mapping problem is defined as to convert from A to B. Assume the conversion follows the following linear form:

$$\mathbf{A * X = Y,}$$

…where X is the conversion matrix that the solution is looking for. We formulate the solution as a multivariate linear regression [20,21]. Next, the calculation of X is obtained by solving the following optimization problem:

$$\mathbf{min\_X \ \| A * X - Y \|^2\_F}$$
$$\mathbf{s.t., \quad A * X >= 0}$$

Once X is calculated, then we can use it to convert a new (P, S) combination (i.e., a), via the following equation:

BDU values = a * X (that is, y = a * X is the converted BDU values)

This result is not necessarily non-negative (as b,d,u values are probabilities), hence, a normalization method needed to be applied and the TruSStReMark framework uses the following normalization:

y = y / sum(y) (i.e., normalize y by dividing y by the sum of its values)

This is chosen to normalize the dominance of any single value and get the ranges within the value ranges of b, d, and u [21]. Therefore, the results should be the normalized y representing the BDU probabilities.

Starting from the identified boundary samples, which are listed in Table 3, all the other sentiment values computed by the framework are converted to subjective logic tuples using the model described above.

2. pEbRank*n*Rec-Batch Processing with Hadoop

This section discusses the parallelization of these approaches in details using the map-reduce paradigm.

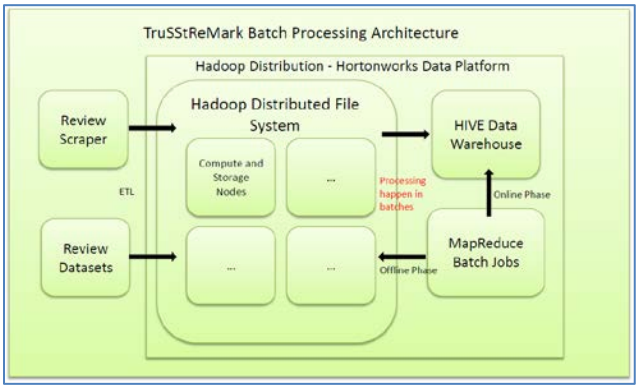

Figure 3. TruSStReMark Batch Processing Architecture

TruSStReMark is categorized into two phases called as online and offline phases. The offline phase performs scraping of evidences from marketplace, organize and cleansed them into a data warehouse (i.e., ETL -Extract, Transform and Load, categorization and organization of evidences). The online phase uses the data warehouse to generate and update metadata related to trust contracts and to perform analysis (i.e., execution of the algorithms to analyze evidences in real-time). Real-time analysis of evidences for one service is fast enough to perform in a single computing node, however, the next step was to improve our solution to perform parallel quarries from different users and handle the load of a real marketplace. As the number of apps and number of parallel queries (roughly indicated by number of downloads) increases, both online and offline phases needed modifications such that they can perform analysis and produce results in real-time. When loaded with parallel user quarries to simulate an online service marketplace, the performance of these evidence-based operations in both phases are affected as a result of these modifications.

First, as an experimental motivation for parallelizing our previous algorithms, we performed and compared the runtime variation of the serialized versions of these algorithms (i.e., Eb-CBF-Rank and Eb-CLF-Rec) with the increasing number of reviews in the datasets. As the number of reviews increases, both the number of users and number of apps also increases [8]. Based on this exercise, it was clear that both prevalent and our trust-based algorithms take more time to perform their operations. However, the trust-based algorithms increase at a much larger rate, due to the sentiment analysis operations on the reviews, the conversions of aggregated sentiments to subjective logic based tuples, and the subjective logic-based calculations of the selected set of QoS-related sentences [8].

Therefore, we propose that this increase in time can be reduced significantly by parallelizing and running the algorithms (named pEbRanknRec) using a framework such as MapReduce based Hadoop environment using an Amazon EMR (Elastic MapReduce) cluster. Additionally, the pEbRanknRec augments the process (i.e., relevant CBF and CLF) by parsing textual reviews from users to calculate the sentiment expressed about each user's targeted QoS properties of apps. These QoS values, as indicated earlier, for apps can be calculated from the keywords, from service description and from user reviews. Using TextBlob library inside each sentence, the algorithm calculates the sentiments of user reviews and cached them. The conjunction, consensus and ordering operators from subjective logic are used in pEbRanknRec algorithms [8].

We use the Hortonworks Data Platform (Figure 3), which is an open-source Hadoop distribution with pre-configured packages and tools such as, YARN (i.e., the resource and job management engine), HDFS (i.e., Hadoop Distributed File System) and HIVE (i.e., the data warehouse infrastructure). Periodically, during the offline phase, the custom review scrapper loads the most recent batch of the reviews from the marketplace to the HDFS. This ETL processing (i.e., Extract, Transform and Load) part of the offline phase can be configured to run either hourly, daily or off-peak load times. For example, a MapReduce job runs offline periodically to calculate vectors (of size k keywords –the outline of this algorithms are presented in Figure 4 and Figure 5) by parsing a new service description (i.e., identified using unique item id from the reviews) using a NLP technique such as TF-IDF (all data structures are cached and reused if no updates detected).

Given a review dataset from a marketplace which includes service descriptions, user-service relations, and a set of reviews by the users, the batch processing version of the algorithm pEbRanknRec performs dataset transformation to partition (i.e., possible due to the review independence property) such that HDFS could handle the correct service spaces, user spaces and user-service spaces for each mapper. Also, during the offline phase, MapReduce batch jobs are submitted to extract sentiments and load to a structured format in the Hive warehouse, and then convert those sentiments to subjective logic based tuples to the appropriate location in the Hive warehouse. Finally, when the new data is available in the Hive warehouse, other batch jobs use subjective logic operators to update trust vectors in the user space and the service space (i.e., Service Vector Space S[] and User Vector Space in U[] in Figure 4 and Figure 5) with a fresh set of subjective logic tuples, which are organized by their QoS values.

The goal of the online phase of the parallelized TruSStReMark framework (i.e., pEbRanknRec approach as indicated in Figure 3 and Figure 4) is to improve performance and confidence while

searching and recommending software apps by parallelizing the previous algorithms of aggregating external evidences available as textual reviews. First, it has a list of named entities for each user (i.e., a profile of important service QoS feature keywords). It performs the improved evidence-based search and recommendation algorithms (i.e., as indicated in Figure 3 and Figure 4) in parallel as follows: each user query for items/apps/apps is tagged with evidences of these feature vectors are used (i.e., which are calculated from users reviews during the offline phase of the algorithm). Since, structured data (i.e., user spaces, item spaces and their corresponding trust scores) are available in the data warehouse each mapper can now handle the correct service spaces, user spaces and user-service spaces as indicated in Figure 4 and Figure 5.

```
pEb-CBF-Rank: Service Reviews, Descriptions and Query

For Each Selected Interval (0)(BY Hour, Day, OR Size)
   Push new/updated review data to HDFS
Each Mapper in Parallel (1) (only for new reviews):
  For Each SERVICE (S[] as Service Vector)
    For All REVIEWS (T as Combined Text Document of R,D)
       Calculate TF-IDF and select top k and cache
  Cached:  S[k] Service Vector (Sorted from TF-IDF score)
Each Mapper in Parallel (2): (only for new reviews):
  For Each USER (U[] as User Vector)
      For All REVIEWS (T as Combined Text Document of R,Q)
         Calculate TF-IDF and select top k and cache
  Cached:  U[k] User Vector (Sorted from TF-IDF score)
Each Mapper in Parallel: with S[] (3)  and  with U[] (4)
  For Each SERVICE (S[]/U[] as QoS <B,D,U> tuples){
   For Each REVIEW (T as Text)(1 to N)   {
     For each QoS in S[]/U[] {
       In Parallel Calculate SENTIMENT of T (P,S)
                     (Polarity, Subjectivity)
       In Parallel Convert (P,S) to <B,D,U>
       Calculate W - TIME-SENSITIVITY of T (Range 0 to 1)
         [('ReviewDate'-'ServiceFirstAvailableDate') /
         ('Today'- 'ServiceFirstAvailableDate')]
       In Parallel Update S[i]/U[i] <B,D,U>
                     (Consensus OR Conjunction)
       Based on current <B,D,U> : subject to weights of W
  Cached:  Produce S*[], U*[] (with evidence <B,D,U>s)
  // User and Service Space Metadata generation Complete!

Each Mapper in Parallel: (With respective to Q)
  For each size k and perform CBF on S[]/U[]
Each Reducer in Parallel:
```

Figure 4. Evidence-based search and ranking (pEb-CBF-Rank)

```
pEb-CLF-Rec: Service Reviews, Descriptions and Query

Execute Steps (0)-(3) Cntd Addition to Figure 4
Each Mapper in Parallel(4): (only for new/updated reviews):
  Pass all reviews and augment or update
  Cached: produce local U-I [] rating vector
Each Mapper in Parallel: User's U[], U[QoS] <B,D,U> tuples
 For Each CANDIDATE SERVICE (S[], S[QoS] <B,D,U> tuples)
   Calculate Two COSINE Similarity
       Each based on the vales from
        - (1) S[] with U[]
        - (2) S[QoS] with U[QoS] [For Each (B,D,U)]
    If two users are A and B: For each common R rating,
    Then COSINE Similarity: ((Σ Ar * Br)/(||A|| * ||B ||))
     Where ||A|| = ( Σ (Ar)2 )0.5
// User-Service Space Metadata generation Complete!

Each Mapper in Parallel: (With respective to Q)
  For each size p,k and perform CLF on S-I[]
Each Reducer in Parallel:
  Produce local candidate set C[p*k]
    Locally sorted: C[p*k] based on search ranking score
Use custom range practitioner (globally) produce C*[p*k]
```

Figure 5. Evidence-based Recommendation (pEb-CLF-Rec)

For each app inside each mapper, the following parts of the basic content-based algorithm are executed. When a query is forwarded to the system, during the online phase of the algorithms, each mapper uses part of the meta-data by executing the algorithm to measure accuracy and performance matrixes. Then we randomly pick approximately 5% of the users from each dataset to remove one service from each of the selected users and we parse each of the dataset though both prevalent and evidence-based algorithms to generate top-N search and recommendation (S&R) result sets for each user. If the test service is found in the set (at any particular rank i) of resulting apps list to the relevant user then it is considered as a Hit. We use a custom range petitioner to sort the candidate set directed to each reducer to produce a globally sorted candidate service set based on their confidence scores. In Figure 4 and Figure 5, steps 1-4 constitutes the offline phase and rest makeup the online phase of the algorithm.

When all the results are obtained, we produce another set randomly to cross validate and generate results iteratively to get the average of all matrices. To compare the sequential algorithm with the parallel algorithm, we use average end-to-end time calculations. In the parallel version of the algorithm, the end-to-end time is measured by combining the average mappers time and reducer times in online phase with offline phase.

2.1 Experimentation and Evaluation Criteria

We perform the offline stage after partitioning and loading data to the HDFS. Then each mapper locally parses the reviews by executing the algorithm to measure accuracy and performance matrixes. Then we randomly pick approximately 5% of the users from each dataset to remove one service from each of the selected users and we parse each of the dataset though both prevalent and evidence-based algorithms to generate top-N search and recommendation (S&R) result sets for each user. If the test service is found in the set (at any particular rank i) of resulted services list to the relevant user (considered as a *Hit*). We use a custom range partitioner to sort the candidate set directed to each reducer to produce a globally sorted candidate service set based on their confidence scores. When all the results are obtained, we produce another set of used to cross validate and generate results iteratively to get the average. We evaluate each of the result-set for both *pEb-CBF-Rank* and *pEb-CLF-Rec* using HR *(Hit Ratio)* and ARHR *(Average Reciprocal Hit-Rank)* measurements that are commonly used in S&R evaluations. The **HR** is a measure of the number of test services that each algorithm included in the result-set of S&R. If n is total #services of the result-set, then $HR = (Number\ of\ Hits)/n$. The **ARHR** is for evaluating the relative significance of the position (p_i) in the result-set of S&R. Hits which appear earlier are scored higher than the Hits appear later in the result-set ordering. If service is positioned at p_i in the result-set of size n, then $ARHR = (1\backslash n) \sum(1\backslash p_i)$. The globally sorted final result-sets are evaluated using the rank (i.e., number of 3,5,10,15 or 20 set of services) by producing 100 times to get the average.

2.2 Results and Analysis of pEbRank*n*Rec-Batch

To evaluate this parallelized approach, we increase the size of data to 10 times the size of both the sequential experiments datasets (i.e., when algorithms perform inside a single instance). We then apply Search/Ranking (pEb-CBF-Rank) to the Android marketplace Review dataset [10] (which now includes 34,169,077 reviews of mobile 2,702,594 apps and apps) and apply Recommendation techniques (pEb-CLF-Rec) to the Amazon Marketplace Reviews

dataset (which now includes Reviews 34,686,770 reviews of products 2,441,053) respectively. Both datasets had around 6 million users, users > 50 reviews more than 50 thousand, 80 median words per review and timespan Jan 1995 – October 2016. Our experimental setup was made up of Java and Python running on an environment containing Amazon EC2 (Amazon Elastic Computing Cloud) free tier Linux t2.micro instances with 64-bit platform support, 15 node EMR (Elastic MapReduce) Cluster running version 2.4 Hadoop.

We present the results from the experiments conducted with a parallel version of the trust-based selection algorithm (i.e., pEb-CBF-Rank). For the search/ranking study, we have used the Android Marketplace dataset. The experiments compare our evidence-based approach for search/ranking with the prevalent approaches of Content-based filtering (i.e., CBF and sequential sEb-CBF-Rank, with parallel pEb-CBF-Rank). Table 4 indicates the HR and ARHR percentage improvements obtained by these approaches using the Android marketplace-based dataset. It is evident from Figure 6 that the proposed parallel evidence-based search and ranking algorithm performs better than the prevalent content-based filtering (CBF) algorithm. This is equally true with our original sequential version in terms of the HR and ARHR achieved. We performed these parallel algorithms iteratively 100 times to get the average of quality matrices including the average runtime. Since, we randomly take a portion of the dataset out to cross validate the results at each step there is a slight difference between the sequential and parallel version of the results. However, they perform nearly within the same range of values when we average the results.

Table 4. Percentage improvements of HR and ARHR on Android Marketplace based dataset experiments

Technique / Result Set Size	Top 3	Top 5	Top 10	Top 15	Top 20
CBF (HR)	11.67	21.34	22.67	24.81	26.72
sEb-CBF-Rank (HR)	12.22	25.69	30.81	32.62	33.9
pEb-CBF-Rank (HR)	11.76	25.32	30.14	32.22	33.65
CBF (ARHR)	5.3	9.92	10.76	12.4	12.86
sEb-CBF-Rank (ARHR)	6.78	14.27	17.11	17.38	17.92
pEb-CBF-Rank (ARHR)	5.42	13.76	16.87	17.16	17.56

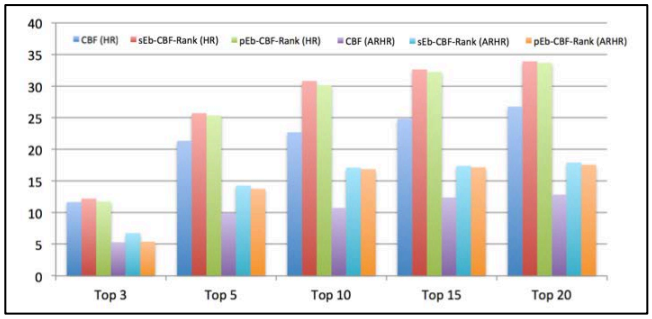

Figure 6. HR & ARHR comparison of Android Marketplace dataset

Next, we present the results from the experiments conducted with parallel version of the trust-based recommendation algorithm (i.e., pEb-CLF-Rec). For this recommendation study, we used the Amazon Marketplace dataset. The experiments compare the evidence-based approach for search with the prevalent approaches of Collaborative filtering (i.e., CLF, sequential version (sEb-CLF-Rank), and parallel version (pEb-CLF-Rec) of the algorithms).

Table 5. Percentage improvements of HR and ARHR on Amazon Marketplace based dataset experiments

Technique / Result Set Size	Top 3	Top 5	Top 10	Top 15	Top 20
CLF (HR)	19.17	21.8	24.26	26.72	30.72
sEb-CLF-Rank (HR)	21.08	23.82	26.65	29.18	32.18
pEb-CLF-Rank (HR)	20.64	23.13	25.95	28.58	31.38
CLF (ARHR)	9.58	10.38	11.55	13.03	15.03
sEb-CLF-Rank (ARHR)	10.64	12.53	13.66	15.36	18.36
pEb-CLF-Rank (ARHR)	9.93	12.02	13.21	14.89	17.52

Table 5 indicates the HR and ARHR percentage improvements obtained by these approaches against the Amazon marketplace-based dataset. It is evident from the Figure 7 that the proposed parallel evidence-based search and ranking algorithm perform better than the prevalent collaborative filtering approach (CLF) and equally with our earlier sequential approach in terms of HR and ARHR. In summary, by analyzing the results, we observe that the percentage improvement of HR and ARHR is equal to prevalent approaches given the results were generated in near real-time. Again, as mentioned previously, we randomly take a portion of the dataset out to cross validate the results at each step there is a slight difference between the sequential and parallel version of the results. However, they perform nearly during the same range of values when we average the results. This shows the importance of QoS consideration in our evidence-based approach in real-time, and this validates our approach.

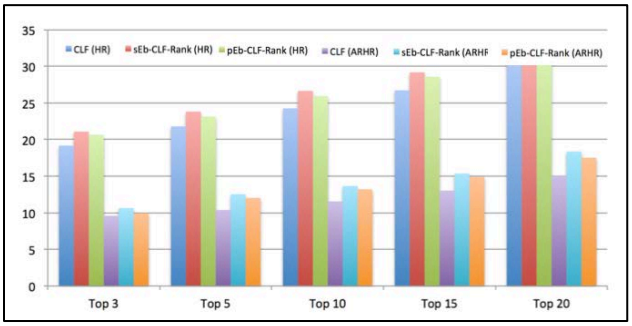

Figure 7. HR & ARHR comparison of Amazon Marketplace dataset

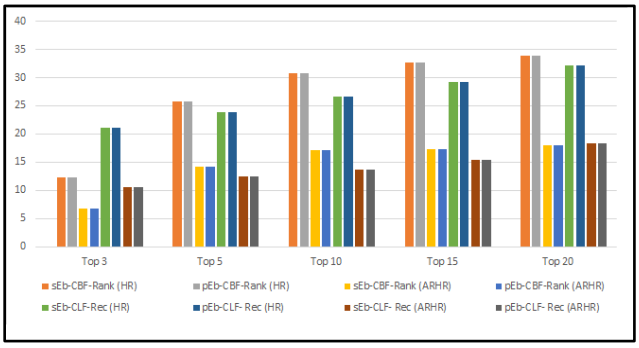

Figure 8. Results comparison of pEb-CBF-Rank without randomization

As an additional step to verify the behavior of sequential and parallel version of the algorithms, we removed the random portion

of the dataset partitioning. At each iteration of the experiment, both the sequential and parallel algorithms now receive the same test and training data without random selection between the approaches by manually overriding the data in HDFS. The experiment was performed 100 times to get the average values of both HR and ARHR matrices and Figure 8 indicates results of this additional experiment. As expected, the results indicate that both sequential and parallel algorithms were able to produce the same HR and ARHR values, since they both received the same data sets in HDFS.

3. pEbRanknRec-Stream Processing with Spark

This section describes TruSStReMark adaptation to the stream processing concepts. Since there were improvements with parallelized versions of the algorithms (i.e,. pEb-CBF-Rank and pEb-CLF-Rec), we also realized that MapReduce based batch processing is also limiting the performance of these algorithms. Therefore, we thought that this increase in time can be reduced further by running the algorithms using a data streaming framework such as Apache Spark based Sparks Streaming oriented processing to replace the Apache Hadoop based batch processing framework. Since Spark Streaming can directly use HDFS, the overall architecture remains the same (as indicated in Figure 8). The advantage that stream processing brings is its ability to perform micro-batch based transformations, without flushing the data to the disk. For example, earlier, during the batch processing, TruSStReMark executes mapreduce jobs in batches (e.g., to extract related sentiments from QoS of apps using the reviews, to convert sentiments to subjective logic based tuples and augment apps' trust vectors using subjective logic based operators).

The end-to-end time is measured to perform each streaming step and iteratively to measure the average runtime. The streaming version performs faster which is expected due to not having the overhead associated with the batch oriented processing associated with MapReduce jobs. For example, each time a batch job executes the results are needed to be written to distributed disk such that the next batch jobs have a consistent view of the data to preserve the states. However the streaming version of this does not need to write data and preserved them in the distributed version of the memory to perform sequence of jobs by having a consistent view of the states. This improves the performance of our algorithms while performing the bath oriented jobs to calculate the QoS oriented sentiments, to convert them to the associated subjective logic based tuples, and then update the service trust scores using subjective logic based operators. The experiments are performed with this improved stream processing architecture using Amazon EMR (Apache Spark with Spark Streaming) clusters. The results and analysis are discussed in next subsections.

3.1 Results and Analysis of pEbRank*n*Rec-Stream

Table 6 indicates the HR and ARHR percentage improvements by comparing the batch oriented version of the pEb-CBF-Rank (i.e., bt-pEb-CBF-Rank) to stream oriented version of the pEb-CBF-Rank (i.e., st-pEb-CBF-Rank). As indicated earlier the slight differences are due to the random requirement of data partitioning to training and test datasets which is mandatory to calculate the HR and ARHR measures.

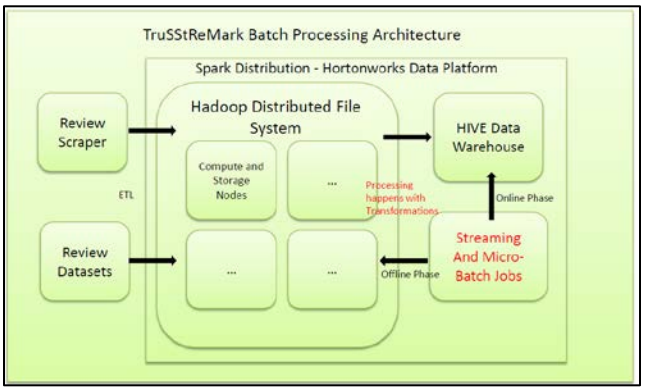

Figure 9. TruSStReMark Stream Processing Architecture

Table 6. HR and ARHR on batch and streaming of pEb-CBF-Rank

Technique / Result Set Size	Top 3	Top 5	Top 10	Top 15	Top 20
bt-pEb-CBF-Rank (HR)	11.76	25.32	30.14	32.22	33.65
st-pEb-CBF-Rank (HR)	12.34	24.05	31.64	30.60	31.96
bt-pEb-CBF-Rank (ARHR)	5.42	13.76	16.87	17.16	17.56
st-pEb-CBF-Rank (ARHR)	5.69	13.07	17.713	16.30	16.68

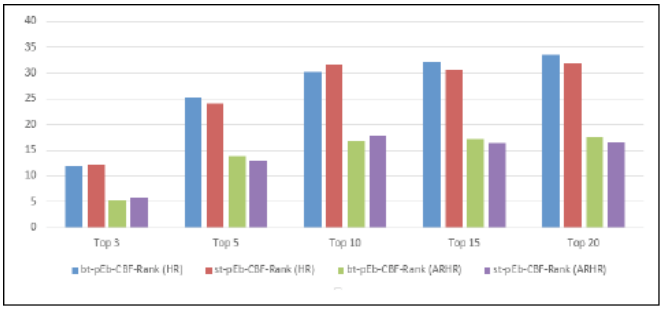

Figure 10. Comparison of bt-pEb-CBF-Rank and st-pEb-CBF-Rank

Figure 10 indicates these results in graphical notation. As expected we concluded that apart from the slight differentiation of the quality measures due to the random requirements in calculating HR and ARHR, the batch and streaming version of the trust-based selection algorithms performs equally well in quality matrices.

Similarly, Table 7 indicates the HR and ARHR percentage improvements by comparing the batch oriented version of the pEb-CLF-Rac (i.e., bt- pEb-CLF-Rac) to stream oriented version of the pEb-CLF-Rac (i.e., st- pEb-CLF-Rac). As indicated earlier the variations of the values are due to the randomness of the data partition while calculating the quality matrices.

Table 7. HR & ARHR on batch & streaming results of pEb-CLF-Rac

Technique / Result Set Size	Top 3	Top 5	Top 10	Top 15	Top 20
bt-pEb-CLF-Rec (HR)	20.64	23.13	25.95	28.58	31.38
st-pEb-CLF-Rec (HR)	19.60	24.28	27.24	27.15	29.81
bt-pEb-CLF-Rec (ARHR)	9.93	12.02	13.21	14.89	17.52
st-pEb-CLF-Rec (ARHR)	9.43	12.621	13.87	14.14	16.64

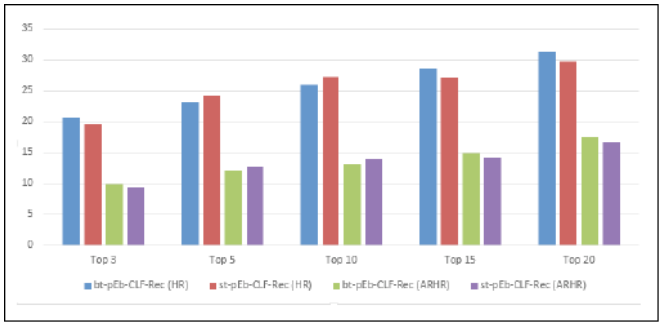

Figure 11. Comparison of bt-pEb-CLF-Rec and st-pEb-CLF-Rec

Figure 11 indicates the trust-based recommendation algorithms versions of the batch and streaming results in graphical notation. As expected we concluded that HR and ARHR matrices of the batch and streaming version of the trust-based recommendation algorithms performs equally well.

4. Performance and Runtime Analysis

We evaluated the performance of the pEb-CBF-Rank and the pEb-CLF-Rec against prevalent approaches. Since the main concern is performance of the algorithms, we compared the runtime comparison of algorithms (prevalent CBF/CLF with both sequential sEb-CBF/CLF-Rank/Rec, and parallel pEb-CBF/CLF-Rank/Rec) with the increasing number of reviews in the datasets.

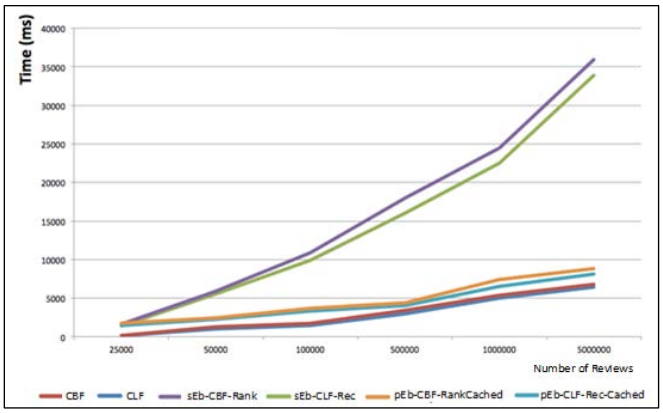

Figure 12. Aggregated average runtimes of the parallel algorithms

As the number of reviews increases, both the number of users and number of apps also increase. To compare the sequential algorithm with the parallel algorithm we use average end-to-end time calculations. In the parallel version of the algorithm, the end-to-end time is measured by combining the average mappers time and reducer times in the online phase with the offline phase.

Since the main concern is performance of the algorithms, we compared the runtime comparison of algorithms (prevalent CBF/CLF with both *sequential* sEb-CBF/CLF-Rank/Rec, and *parallel* pEb-CBF/CLF-Rank/Rec) with the increasing number of reviews in the datasets. As the number of reviews increases, both the number of users and number of services also increase.

From the comparisons in Figure 12, it is clear that both prevalent and trust-based algorithms take more time to perform their operations. Since, the sequential evidence-based S&R algorithms increase at a much higher rate, due to due to its additional calculations. Our parallelized algorithms using MapReduce/Hadoop environment perform equally when dataset size is much higher. Each data set inside local node of the cluster is small and it was able to perform the algorithm efficiently locally to produce global results.

Also, caching experiments are performed by directly using the HIVE meta-data store. From the comparisons in Figure 11, it is clear that both prevalent and trust-based algorithms take more time to perform their operations. Since, the sequential evidence-based S&R algorithms increase at a much higher rate, due to its additional calculations. Our parallelized algorithms using MapReduce/Hadoop environment perform equally when the dataset size is much larger. Each data set inside the local node of the cluster is small hence, it was able to efficiently perform the algorithm locally in order to produce global results.

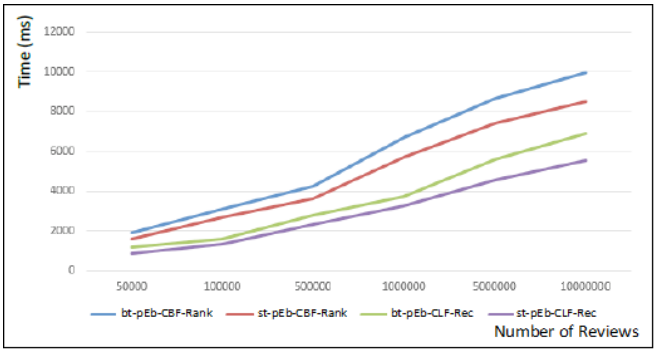

Figure 13. Aggregated average runtime of batch vs streaming

Figure 13 indicates analysis about streaming versions of the algorithms in terms of average runtime compared to average runtime of the batch processing versions of the algorithms. The streaming version performs faster which is expected due to not having the overhead associated with the batch oriented processing that MapReduce jobs require. Each time a batch job executes the results need to be written to a disks which are distributed such that the next batch jobs have a consistent view of the data in order to preserve the states. We also noted that the gap is being widen when the number of reviews are increased. This is also expected, as when the load increases the stream processing performs much better than batch oriented processing version of the algorithms.

5. Related Works

There are not many related efforts on evidence-based searching, ranking and recommending when considering software apps and services from marketplaces. Though, many prevalent approaches propose techniques to improve collaborative and content-based filtering algorithms. The work proposed by Fu et al. [22] experiments on a recommender system to parallelize data using Hadoop echo-system. It collects data from users, commodities, and transactions. Although they describe and address the challenges in processing and generating recommendations with large data sets,

they do not consider external evidences to further improve results quality. Zhou et al. [23] propose a real time search and recommender system targeting microblogs. They only consider user tags (i.e., hashtags) to compute the similarity. In the context of software, it is not enough to consider the service type and description tags. The work proposed by Jiang et al. [24] experiments with a Hadoop based recommendation mechanism to improve collaborative filtering performance at scale. The difference between this our approach is that we consider individual service features and attributes while their approach considers confidence about web services as a whole. Zhang et al. [25] propose a ranking model for scientific publication (e.g., dblp). The model named as Knowledge-Social-Trust, which is a graph-based network. It recommends in real-time by quickly calculating the relative importance of citations by crawling repositories. Compared to our use case in service marketplace, it has no concept of citing or referring one service to another. Therefore, their evidence-based model can be adapted to the context of marketplaces if these stores introduce service referencing or advanced user interactions.

6. Conclusion and Future Works

This paper presents a framework ("TruSStRemark"), which investigates algorithmic modifications necessary to parallelize and perform evidence-based search and recommend (S&R) of services. Using the TruSStReMark framework we suggested algorithmic modifications necessary to parallelize and perform evidence-based search and recommendation (S&R) of apps. The framework uses datasets from Amazon and Android marketplaces. The two batch processing versions of the parallel algorithms (i.e., pEb-CLF-Rec and pEb-CLF-Rec) in the pEbRanknRec algorithms were executed using an EMR environment in AWS using Hadoop echo-system. The main algorithmic calculations are based on sentiments from the large volumes of textual reviews, which are then numerically converted to subjective logic based BDU tuples to apply aggregation operators). Hive warehouse is used as the caching meta-store, where algorithms quickly find the aggregated trust scores of QoS related to apps. When compared to prevalent approaches, the results indicate that our parallelized algorithms improve the average performance of the trust-based algorithms and are able to generate better or equally well with both datasets in terms of HR and ARHR. Next, the batch processing part of the Hadoop jobs are streamlined using Sparks streaming techniques.

When compared to the batch oriented technique, the stream oriented algorithms (i.e., st-pEb-CLF-Rec and st-pEb-CLF-Rec) were able to perform equally well with the advantage of reduced aggregated average runtimes. The advantage of our approach is that it is based on heterogeneous and dynamic software features (QoSs) of apps that enable better temporal comparisons between software apps. The results of this study indicate that the evidence-based approaches provide better selections and recommendations both in terms of quality and relative ranking of apps. The results of this study indicate that the evidence-based approaches provide better selections and recommendations both in terms of quality and relative ranking of software services. We plan to improve performance further by using an iterative version of MapReduce with intelligent caching. Also, other similarity measurement techniques such as matrix factorization used in the latent-models applications are our future explorations in the context of large datasets in marketplaces.

References

[1] Konstan, J. A., 2008. Introduction to recommender systems. ACM Special Interest Group on Management of Data (SIGMOD) Conference.

[2] Ning, X. and Karypis, G. 2015. Recent Advances in Recommender Systems and Future Directions. In Proceedings of Pattern Recognition and Machine Intelligence (PReMI) Volume 9124: 3-9.

[3] Gallege, L. S., Gamage, D. U., Hill, J. H., and Raje R. R. 2011. "Understanding the trust of software-intensive distributed systems," Concurrency and Computation: Practice and Experience, pp. 114-143.

[4] Gallege, L. S., Gamage, D. U., Hill, J. H., and Raje R. R. 2016. "Towards a Comprehensive Method for Integrating Trust into Enterprise DRE Systems," in Proceedings of Real-time Computing Systems and Applications (RTCSA 2011).

[5] Gallege L.S., 2013. "TruSSCom: Proposal for Trustworthy Service Representation, Selection and Negotiation for Integrating Software Systems," in Proceedings of the 2013 International Conference on Systems, Programming, and Applications: Software for Humanity (SPLASH 2013).

[6] Gallege, L. S., Gamage, D. U., Hill, J. H., and Raje R. R. 2013. "Trust contract of a service and its role in service selection for distributed software systems," in Proceedings of 8th Annual Cyber and Information Security Research Conference (CSIIRW 2013).

[7] Gallege, L. S., Gamage, D. U., Hill, J. H., and Raje R. R. 2013. "Trustworthy Service Selection Using Long-Term Monitoring of Trust Contracts," in Proceedings of 17th IEEE International Enterprise Computing (EDOC 201).

[8] Gallege L. S. and Raje, R. R. 2016 "Towards Selecting and Recommending Online Software Services by Evaluating External Attributes," in Proceedings of the 11th Annual Cyber and Information Security Research Conference (CISR 2016).

[9] Shafer, G. 1976. A Mathematical Theory of Evidence. Princeton University Press. Princeton.

[10] Jøsang, A. 1997. Artificial Reasoning with Subjective Logic. In Proceedings of the Second Australian Workshop on Common-sense Reasoning, Perth, Australia.

[11] Pang, B. and Lee L. 2002. ThumbsUp? Sentiment Classification using Machine Learning Techniques. In Proceedings of Empirical Methods in Natural Language Processing (EMNLP).

[12] Amazon Snap Review dataset. 2015. http://snap.stanford.edu/data/

[13] Android Marketplace from Google Play Store. 2016. Reviews from Mobile App Market-place, https://play.google.com/store/apps.

[14] Wikipedia Inc, Content Based Filtering, Accessed on 12/01/16, URL: https://en.wikipedia.org/wiki/Recommender_system#Content-based_filtering, 2016.

[15] Wikipedia Inc, Collaborative Filtering, Accessed on 12/01/16, URL: https://en.wikipedia.org/wiki/Collaborative_filtering, 2016.

[16] Dean, J. and Ghemawat, S. 2004. MapReduce: Simplified Data Processing on Large Clusters. In Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI) p.137-150.

[17] H. Karau, A. Konwinski, P. Wendell and M. Zaharia, Learning Spark: Lightning-Fast Big Data Analytics, 1st ed., O'Reilly Media, Inc., 2015.

[18] Apache Inc, Spark Streaming, Accessed on 12/01/16, URL: https://en.wikipedia.org/wiki/Apache_Spark#Spark_Streaming, 2016.

[19] TextBlob Project. 2016. Python based Text Processing Tool Suit.

[20] Wikipedia Inc., Linear Regression, Accessed on 08/15/16, URL:https://en.wikipedia.org/wiki/Linear_regression, 2016.

[21] Wikipedia Inc., Miltivariate Linear Regression, Accessed on 08/15/16, URL:https://en.wikipedia.org/w/index.php?title=Multivariate_linear_regression, 2016.

[22] Fu, C. and Leng, Z. 2010. A Framework for Recommender Systems in E-Commerce Based on Distributed Storage and Data-Mining, In Proceeding of *International Conference on E-Business and E-Government (ICEE)*, Guangzhou. pp. 3502-3505.

[23] Zhou, X., Wu, S., Chen, C., Chen, G., and Ying, S. 2014. Real-time recommendation for microblogs. Information sciences (IJJS 2014).

[24] Jiang, J., Lu, J., Zhang, G. and Long, G. 2011. Scaling-Up Item-Based Collaborative Filtering Recommendation Algorithm Based on Hadoop, *IEEE World Congress on Services*, Washington, DC.

[25] Zhang, J., Votava, P., Lee, T. J., Adhikarla, S., Kulkumjon, I. C., Schlau, M., Natesan, D., and Nemani R. 2013. Technique of Analysing Trust Relationships to Facilitate Scientific Service Discovery and Recommendation. In Proceeding of international Conference on Service Computing (SCC 2013).