symmetry

MDPI

Article

An Efficient MapReduce-Based Parallel Processing Framework for User-Based Collaborative Filtering

Hanjo Jeong [1] and Kyung Jin CHA [2,*]

[1] School of Information Convergence, Kwangwoon University, Seoul 01897, Korea; hanjojeong@kw.ac.kr
[2] Department of Business Administration, Kangwon University, Chuncheon 24341, Korea
* Correspondence: kjcha7@kangwon.ac.kr; Tel.: +82-33-250-6134

Received: 16 May 2019; Accepted: 29 May 2019; Published: 3 June 2019

Abstract: User-based collaborative filtering is one of the most-used methods for the recommender systems. However, it takes time to perform the method because it requires a full scan of the entire data to find the neighboring users of each active user, who have similar rating patterns. It also requires time-consuming computations because of the complexity of the algorithms. Furthermore, the amount of rating data in the recommender systems grows rapidly, as the number of users, items, and their rating activities tend to increase. Thus, a big data framework with parallel processing, such as Hadoop, is needed for the recommender systems. There are already many research studies on the MapReduce-based parallel processing method for collaborative filtering. However, most of the research studies have not considered the sequential-access restriction for executing MapReduce jobs and the minimization of the required full scan on the entire data on the Hadoop Distributed File System (HDFS), because HDFS sequentially access data on the disk. In this paper, we introduce an efficient MapReduce-based parallel processing framework for collaborative filtering method that requires only a one-time parallelized full scan, while adhering to the sequential access patterns on Hadoop data nodes. Our proposed framework contains a novel MapReduce framework, including a partial computation framework for calculating the predictions and finding the recommended items for an active user with such a one-way parallelized scan. Lastly, we have used the MovieLens dataset to show the validity of our proposed method, mainly in terms of the efficiency of the parallelized method.

Keywords: MapReduce; collaborative filtering; parallel processing; hadoop; recommendation system

1. Introduction

Collaborative filtering is a method for recommender systems, which is a software system that provides more preferable data items to a user by predicting the user's preference of data items that the user has not yet seen [1,2]. The collaborative filtering can be divided into two methods: one is user-based collaborative filtering [3], and the other is item-based collaborative filtering [4]. This research is more focused on the user-based collaborative filtering, which requires more complex processing and computation.

As the amount of users and items are growing fast, the amount of rating data has grown rapidly. Thus, the recommender systems are used to maintain and manage the large amount of data, and to process and analyze the data in parallel. The Hadoop framework [5] is introduced to process and analyze such large data. Hadoop stores and manages large sets of data with the Hadoop framework, consists of the HDFS (Hadoop Distributed File System) [6] and MapReduce [7] framework. HDFS is a distributed, scalable, and reliable file system with a simple key-value data structure, and it is optimized for batch-oriented programming with the sequential read on disks. MapReduce is a parallel processing programing framework consisting of JobTracker and TaskTracker components, where JobTracker

co-ordinates job processes and TaskTracker co-ordinates maps and reduces tasks on the data nodes in HDFS.

There are a number of research studies regarding the parallel processing of collaborative filtering with Hadoop MapReduce framework [8–12], but none of them have addressed the sequential processing for performing the MapReduce tasks. Realizing the sequential processing of the MapReduce tasks is an important aspect for the MapReduce-based parallel processing, since the MapReduce tasks require disk input/outputs (I/Os) on the data nodes to read input data and write output data. Thus, assuring sequential access and less full scans on the disks during the MapReduce tasks is critical to preserve the efficiency of the parallel processing.

In this paper, we proposed a new MapReduce-based collaborative filtering method that can process and analyze the rating data in parallel for collaborative filtering. We focused on adhering to the sequential access pattern and minimizing the number of a full scans to develop the method. We also evaluated the efficiency of our proposed method with MovieLens dataset, which is the benchmark dataset for the collaborative filtering. The rest of this paper is organized as follows. Section 2 presents the preliminary study related to this research. Section 3 introduces our proposing method that can process and analyze the user rating data with the MapReduce parallel processing framework for the user-based collaborative filtering. Section 4 shows the experiment and performance evaluation of our proposed method. The conclusion is given in the last section.

2. Preliminaries

2.1. User-Based Collaborative Filtering

The prediction of the user-based collaborative filtering is based on the user rating values, which have previously been given to items not only by the active user, but also by the other users. A well-known recommender system based on user-based collaborative filtering is the GroupLens system [13]. The GroupLens system recommends information items, such as news articles and movies, to the active user by filtering the items based on the predictions made for the active user regarding the user's likeness on the unseen items. The user's likeness of an unseen item is determined based on the user ratings of neighboring users on the unseen item. The neighbor users are selected if they have similar rating patterns with the active users for the commonly-rated items in their profile.

The following equations are used to calculate a prediction for the active user's rating for an unseen item based on the Pearson Correlation Coefficient used in GroupLens:

$$p(u_a, d_u) = \overline{r_{u_a}} + \frac{\sum_i (r_{u_i,d_u} - \overline{r_{u_i}}) * w(u_a, u_i)}{\sum_i w(u_a, u_i)} \quad (1)$$

$$w(u_a, u_i) = \frac{\sum_{d_s} (r_{u_a,d_s} - \overline{r_{u_a}}) * (r_{u_i,d_s} - \overline{r_{u_i}})}{\sigma_{u_a} * \sigma_{u_i}} \quad (2)$$

where $p(u_a, d)$ represents a prediction for the active user u_a for an unseen (unrated) item, d_u; u_i represents the other users who have commonly rated items with the active user; $w(u_a, u_i)$ is the similarity (correlation) weight between the active user and the other user in terms of rating patterns as defined by the Pearson Correlation Coefficient; d_s is the commonly seen (rated) items between the active user and the other user; and $\overline{r_{u_a}}$ and $\overline{r_{u_i}}$ represent arithmetic means for the ratings of data items obtained from the active user and the other users, respectively.

2.2. MapReduce

MapReduce is an open-source software framework for parallel-data processing based on the Hadoop Distributed File System (HDFS). In general, HDFS consists of a name node and multiple numbers of data nodes, and it uses a simple data model consisting of key and value pairs for the

convenience of horizontal scaling and maximizing the parallelism of the data processing. The simple key-value data model is also efficient for the sequential processing of data on the disks, as HDFS basically uses disk I/O-based batch processing, which is more suitable for the large amount of data processing compared to the memory-based processing.

The MapReduce framework consists of multiple mapping tasks and multiple reduction tasks; the mapping tasks are performed on the input splits that are horizontally partitioned data of the input data for the parallel data processing. Also, there are multiple numbers of reduction tasks that are performed on the data nodes to aggregate the outputs of the map tasks in parallel [14].

3. MapReduce-Based Parallel Processing Method for Collaborative Filtering

This section introduces a new MapReduce-based parallel processing method for the user-based collaborative filtering described in Section 2.1. Figure 1 represents the whole process of MapReduce, and is designed to maximize the parallelism for the collaborative filtering method. This example process is created with an assumption that there are only 100 users and 100 items in the system for the explanation. However, it works on any numbers of users and items as the processes can be run record-by-record.

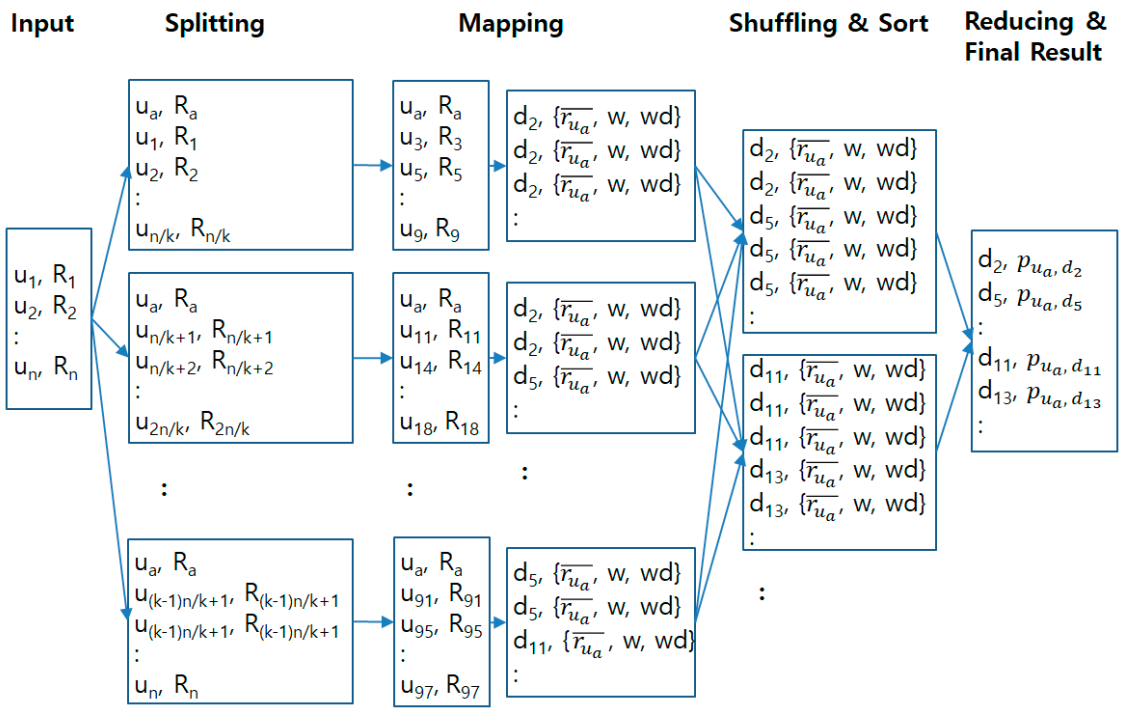

Figure 1. MapReduce process of collaborative filtering for an active user.

As shown in Figure 1, each input record is defined to consist of a user-id (key) and the user's rating data set (value), and the rating data set consists of an item-id and its rating score pairs. The input data is split into k numbers of splits, which are simply divided by the user-ids having the same number of the records in each split. Then, each split is assigned to a map task. This data structure of the input record and the number of splits can ensure the parallel scan of the entire data to find the candidates of the neighbor users and the unseen items in the mapping phase, which allows the system to find all of the unseen items for an active user and simultaneously calculate the necessary partial computation to calculate the rating prediction of the active user for the unseen items. It also makes the system sequentially process such data, thereby enhancing the efficiency of the mapping task, which requires data-read on the disks. After finding the unseen data from each mapper, the final prediction is computed on the reduction phase by aggregating the outputs from the mapping phase.

Our method is developed based on the user-based collaborative filtering method described in Section 2.1. Thus, the formulas for the partial computation on the MapReduce tasks are created based on Equations (1) and (2). In addition, our method defines the entire rating data of each user as a record, which is the basic unit of the data processing in MapReduce tasks. Therefore, the similarity weight defined in Equation (2) can be computed completely in the record-based computation. However, Equation (1) requires multiple users' rating data, so that it should be partitioned for each formula to require only one user's rating data. Equation (3) shows the partitioned formula of Equation (1) that is simply separated based on each user. More details of the partial computations and procedure of the MapReduce tasks are described in the following subsections.

$$p(u_a, d_u) = \overline{r_{u_a}} + \frac{((r_{u_{n1},d_u} - \overline{r_{n1}}) * w(u_a, u_{n1})) + ((r_{u_{n2},d_u} - \overline{r_{u_{n2}}}) * w(u_a, u_{n2})) + \ldots}{w(u_a, u_{n1}) + w(u_a, u_{n2}) + \ldots} \quad (3)$$

where u_{n1} and u_{n2} represent the first and the second neighbor users, respectively.

3.1. Mapping Phase

In the mapping phase, the neighboring user candidates are found by comparing the active user's rating data and the other user's rating data record-by-record. The candidates are the other users having more than one commonly-rated item with the active user. If the user of the current record turns out to be a candidate, the unseen items are found from the candidates' rating data and the partial prediction regarding the unseen items is also calculated at the same time. The unseen items are defined as the items that are not-yet rated by the active user, but rated by the candidates, since the unseen items are the recommended item candidates for the active user based on the neighboring users' ratings. Figure 2 represents the pseudo-code for the map task.

```
function Map(user u_i, ratings R_a)
    for the first record
        u_a ← u_i
        R_a ← R_i
        calculate mean of ratings of active user, r_ua
    Emit_Set ← Empty
    for all of the remained records
        get the number of common items beween R_a and R_i, n(d_common)
        if n(d_common) > 1
            calculate w(u_a, u_i)
            find a set of unseen items, D_i
            for all unseen items, d_u ∈ D_i
                calculate wd(u_i, d_u)
                add (d_u, partial_computation{r_ua, w(u_a, u_i), wd(u_i, d_u)}) to Emit_Set
            end for
        else continue the for loop
    end for
    sort the Emit_Set based on ids of d_u
    for all of elements in Emit_Set
        Emit(d_u, partial_computation{r_ua, w(u_a, u_i), wd(u_i, d_u)})
    end for
end function
```

Figure 2. Pseudo-Code of the Map Task.

The input and output key-value pairs are defined as bellows:
INPUT:
< KEY: *userID*, VALUE: *a set of user ratings* (itemID: rating) >
OUTPUT:
< KEY: *itemID*, VALUE: *partial computation* ($\overline{r_{u_a}}$: $w(u_a, u_i)$: $wd(u_i, d_u)$) >

The input key is a user-id and the input value is the entire set of the rating value pairs, which is the same as the structure of each record in the input splits. The output key represents an item-id that should be one of the unseen items of the active user, so that each mapper outputs only the unseen items, which are potential recommendation items for the active user from the result of the final prediction. The output value is a combined value with the mean of the current user's ratings, correlation weight between the active user and the current user, and the partial weighted deviations of commonly-rated items between the active user and the current user. These output values are aggregated to calculate the final prediction of the preference of the active user for the unseen items in the reduction phase.

For the details of the map-task process, each input split contains the active user's rating data in the first record, so that the first record is assigned to the active user's rating data. Then, the mean of the active user's rating scores is calculated. This mean value is used once to adjust the neighboring users' rating scores in the range of the active user's rating scores, but it is carried out for each record, since the Hadoop MapReduce process is performed record-by-record to materialize the parallel and sequential processing of the disks.

From the second record, the map-task process compares the active user's rating data and the other user's rating data user-by-user. At first, it compares the items in the rating data between the active user and the current user, and it finds the number of commonly-rated items by both of the users. If the number of the commonly-rated items is more than one, the current user can be determined as neighboring user candidate, since the correlation weight can be calculated when there is at least two data points.

If the current user is a candidate of the neighboring users, it calculates the similarity weight between the active user and the current user, $w(u_a, u_i)$, as shown in Equation (4), defined by the Pearson Correlation Coefficient. Note that Equation (4) is identical to the similarity weight computation of the user-based collaborative filtering, as in Equation (2).

$$w(u_a, u_i) = \frac{\sum_{d_s} (r_{u_a,d_s} - \overline{r_{u_a}}) * (r_{u_i,d_s} - \overline{r_{u_i}})}{\sigma_{u_a} * \sigma_{u_i}} \tag{4}$$

Then, it finds a set of the unseen items for the active user among the rated items of the current user, and it calculates the weighted deviation of the unseen item using the mean ratings of the current user, $wd(u_i, d_u)$, as defined in Equation (5). Equation (5) is identical to the partial numerator in Equation (3), which corresponds to only one candidate of the neighboring users. This partial computation makes sure that it is able to perform the map task record-by-record.

$$wd(u_i, d_u) = (r_{u_i,d_u} - \overline{r_{u_i}}) * w_{u_a,u_i} \tag{5}$$

The unseen item-id and its partial computation values are added to an emitted set, and each mapper sorts the emitted set in the order of the item-ids. There are multiple partial-computation values for an unseen item, since it can appear for multiple users. However, the user-ids are not needed to carry this out, since the partial computation contains all of the necessary partial information to calculate the final prediction for the corresponding users. Lastly, each mapper emits the sorted outputs with records consisting of an unseen item-id (key) and its related partial-computation values as a combined value (value).

3.2. Reducing Phase

In the reduction phase, the emitted partial-computation data from the mappers are shuffled, sorted, and reduced in order to calculate the final prediction. As shown in Figure 1, the partial-computation data is shuffled and sorted by item-id, so that the reducer can sequentially process and aggregate it with only one disk scan. Finally, it calculates the final prediction of the unseen items based on the aggregated data and emits the final results. Figure 3 represents the pseudo-code for the reduction task.

```
function Reduce(item d_u, partial_computation [c1, c2, ...])
    prediction(u_a, d_u) ← 0
    sum_wd(u_i, d_u) ← 0
    sum_w(u_a, u_i) ← 0
    r̄_{u_a} ← r̄_{u_a}
    for all partial_computation c ∈ partial_computation [c1, c2, ...]
        sum_wd(u_i, d_u) ← sum_wd(u_i, d_u) + wd(u_i, d_u)
        sum_w(u_a, u_i) ← sum_w(u_a, u_i) +w(u_a, u_i)
    end for
    prediction(u_a, d_u) ← r̄_{u_a} + sum_wd(u_i, d_u) / sum_w(u_a, u_i)
    Emit(d_u, prediction(u_a, d_u))
end function
```

Figure 3. Pseudo-code of the reduction task.

The input and output key-value pairs of the reduction tasks are defined as bellows:
INPUT:
< KEY: *itemID*, VALUE: *partial_computation* ($\overline{r_{u_a}}$: $w(u_a, u_i)$: $wd(u_i, d_u)$) >
OUTPUT:
< KEY: *itemID*, VALUE: $prediction(u_a, d_u)$>

The input keys are the unseen item-ids of the active user, and the input value is a list of the partial-computation data as it is outputted from the mappers. For each unseen item, the final prediction of the rating score of the active user is calculated as defined in Equation (6). The final prediction obtained from this aggregation of the partial-computation values should be identical to the prediction calculated on the entire data, as in Equation (1)

$$p(u_a, d_u) = \overline{r_{u_a}} + \frac{\sum wd(u_i, d_u)}{\sum w(u_a, u_i)} \tag{6}$$

4. Experiments

For the experiments of this research, we used the MovieLens 20M dataset [15], which has been created by the GroupLens research group. The MovieLens 20M dataset contains 20 million ratings for 27,000 movies by 138,000 users.

As the goal of this research was to transform the user-based collaborative filtering method to a MapReduce-based parallel processing method, thus, the accuracy of the proposed method in this research should be identical to the original collaborative filtering method. Therefore, we mainly evaluated the proposed method in terms of its efficiency. The speedup measure was used to evaluate the efficiency of our system, which is a de-facto standard measure for evaluating the performance of the MapReduce-based collaborative filtering systems [16,17]. The speedup measure represents the ratio between the execution time of the serial processing by a single processor and the execution time of the parallel processing by multiple processors [18,19]. In the Hadoop environment, the number of data nodes can be regarded as the number of processors in the parallel processing.

Figure 4 represents the results of the efficiency evaluation. We evaluated the efficiency with the different datasets with ratings data from 1000, 10,000, and 100,000 randomly chosen users.

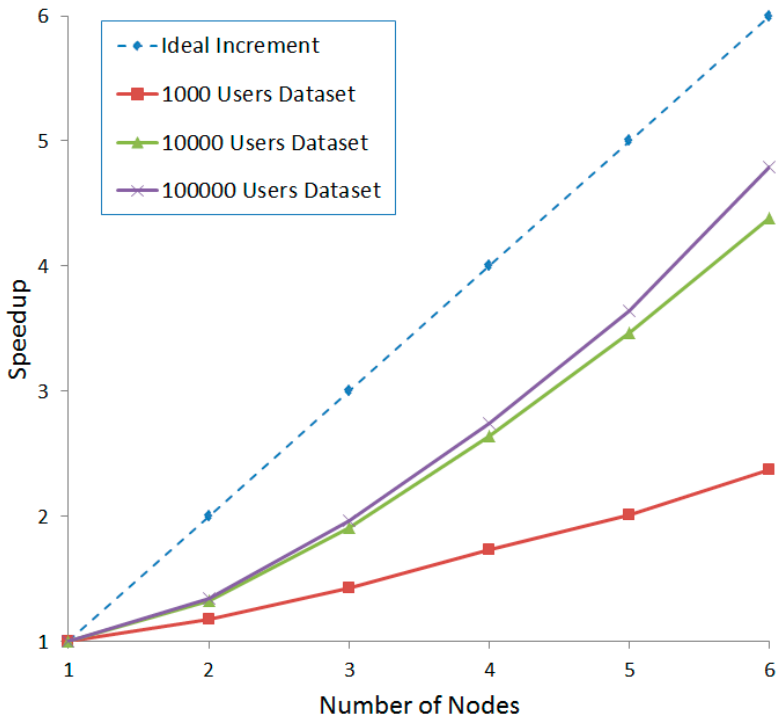

Figure 4. Performance comparison for different numbers of nodes in a Hadoop Cluster.

As shown in Figure 4, the speedup measure is 1 for the time spent with only one node, since it is identical to the serial processing with a single processor. The result shows that the speedup is a bit lower than the ideal increment because the MapReduce-based method reads and writes the interim results on the disk. However, the speedup result is increased exponentially to the number of nodes; this might prove that the proposed method efficiently provides parallelism on more time-intensive tasks, such as finding the candidates of the neighboring users that have commonly-rated items and computing the weighted deviation and the correlation weight for the number of users.

5. Conclusions

There are already many research studies on the MapReduce-based parallel processing method for collaborative filtering. However, most of research studies have not considered the sequential-access requirement for executing HDFS-based MapReduce jobs and minimizing required full scans on the entire data, because HDFS sequentially accesses data on the disk.

In this paper, we presented a novel MapReduce-based parallel processing framework for the user-based collaborative filtering, mainly focused on minimizing the number of full scans while adhering to the restriction of the sequential-access pattern. Our proposed framework includes the partial computation framework and the MapReduce-based parallel data processing framework for calculating the predictions, not only in parallel, but also with only a one-time scan. Therefore, our proposed method allows the system to process and analyze the data for the recommendation task with only a one-time parallelized scan, while satisfying the sequential-access restriction of Hadoop data nodes. That is, our system does not necessarily re-scan previously scanned data, which is an enormous time-consuming job for the HDFS. With such strength, the results of the experiments showed that our proposed method does not sacrifice much of the advantages of the MapReduce-based parallel processing in terms of efficiency.

Author Contributions: Conceptualization, H.J. and K.J.C.; methodology, H.J. and K.J.C.; software, H.J.; validation, H.J.; formal analysis: H.J. and K.J.C.; investigation, H.J. and K.J.C.; resources, H.J.; data curation, H.J.; writing—original draft preparation, H.J.; writing—review and editing, K.J.C.; visualization, H.J.; supervision, K.J.C.; project administration, K.J.C.; funding acquisition, H.J.

Funding: This research was supported by a Research Grant from Kwangwoon University in 2018.

Conflicts of Interest: The authors declare no conflict of interest. In addition, the funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Resnick, P.; Varian, H.R. Recommender systems. *Commun. ACM* **1997**, *40*, 56–58. [CrossRef]
2. Adomavicius, G.; Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.* **2005**, *17*, 734–749. [CrossRef]
3. Bell, R.M.; Koren, Y. Improved neighborhood-based collaborative filtering. In Proceedings of the KDD cup and workshop at the 13th ACM SIGKDD international conference on knowledge discovery and data mining, San Jose, CA, USA, 12 August 2007; pp. 7–14.
4. Linden, G.; Smith, B.; York, J. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Comput.* **2003**, *1*, 76–80. [CrossRef]
5. Hadoop. Available online: http://hadoop.apache.org/ (accessed on 15 May 2019).
6. Borthakur, D. The hadoop distributed file system: Architecture and design. *Hadoop Proj. Website* **2007**, *11*, 1–21.
7. Dean, J.; Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Commun. ACM* **2008**, *51*, 107–113. [CrossRef]
8. Verma, J.P.; Patel, B.; Patel, A. Big data analysis: Recommendation system with Hadoop framework. In Proceedings of the 2015 IEEE International Conference on Computational Intelligence & Communication Technology, Ghaziabad, India, 13 February 2015; pp. 92–97.
9. Sharma, S.; Sethi, M. Implementing Collaborative Filtering on Large Scale Data Using Hadoop And Mahout. *Int. Res. J. Eng. Technol. (IRJET)* **2015**, *2*, 102–106.
10. Shen, F.; Jiamthapthaksin, R. Dimension independent cosine similarity for collaborative filtering using MapReduce. In Proceedings of the 2016 8th International Conference on Knowledge and Smart Technology (KST), Chiangmai, Thailand, 3 February 2016; pp. 72–76.
11. Cai, R.; Li, C. Research on collaborative filtering algorithm based on MapReduce. In Proceedings of the 2016 9th International Symposium on Computational Intelligence and Design (ISCID), Hangzhou, China, 10 December 2016; pp. 370–374.
12. Tang, H.; Cheng, X. Personalized E-commerce recommendation system based on collaborative filtering under Hadoop. *World* **2017**, *1*, 146–148.
13. Konstan, J.A.; Miller, B.N.; Maltz, D.; Herlocker, J.L.; Gordon, L.R.; Riedl, J. Grouplens: Applying collaborative filtering to usenet news. *Commun. ACM* **1997**, *40*, 77–87. [CrossRef]
14. White, T. *Hadoop: The Definitive Guide*, 4th ed.; O'Reilly Media: Cambridge, UK, 2015.
15. MovieLens Dataset. Available online: https://grouplens.org/datasets/movielens/ (accessed on 15 May 2019).
16. Karydi, E.; Margaritis, K. Parallel and distributed collaborative filtering: A survey. *ACM Comput. Surv. (CSUR)* **2016**, *49*, 1–37. [CrossRef]
17. Liu, Q.; Li, X. A New Parallel Item-Based Collaborative Filtering Algorithm Based on Hadoop. *JSW* **2015**, *10*, 416–426. [CrossRef]
18. Gupta, A.K.; Varshney, P.; Kumar, A.; Prasad, B.R.; Agarwal, S. Evaluation of MapReduce-Based Distributed Parallel Machine Learning Algorithms. In *Advances in Big Data and Cloud Computing*; Rajsingh, E.B., Veerasamy, J., Alavi, A.H., Peter, J.D., Eds.; Springer: New York, NY, USA, 2018; pp. 101–111.
19. Diedhiou, C.; Carpenter, B.; Shafi, A.; Sarkar, S.; Esmeli, R.; Gadsdon, R. Performance Comparison of a Parallel Recommender Algorithm Across Three Hadoop-Based Frameworks. In Proceedings of the 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Lyon, France, 24–27 September 2018; pp. 380–387.