Implementation of a Map-Reduce based Context-Aware Recommendation Engine for Social Music Events

Wolfgang Beer
Software Competence Center Hagenberg GmbH
Softwarepark 21
4232 Hagenberg, Austria
Email:wolfgang.beer@scch.at

Christian Derwein, Sandor Herramhof
Evntogram Labs GmbH
Leonfeldner Strasse 328
Linz, Austria
Email: [chris, sandor]@evntogram.com

***Abstract*—In our modern ubiquitously connected world the amount of ever available product and service information within our daily lives is exploding. Powerful client devices, such as smartphones and tablets allow the users to get access to an unlimited amount of information on every product or service available. As the amount of available information on products by far exceeds the users time to examine and filter detailed pieces of information in every situation, we expect that client-centric and context-aware information filtering is one of the thriving topics within the next years. A popular approach is to combine context-awareness with traditional recommendation engines in order to evaluate the relevance of a large amount of items for a given user situation. The goal is to proactively evaluate the situation of a user in order to automatically propose relevant products. Within this work we describe a general approach and the implementation of a software framework that combines traditional recommendation methods with a variable number of context dimensions, such as location or social context. The main contribution of this work is to show how to use a MapReduce programming model for aggregating the necessary information for calculating fast context-aware recommendations as well as how to overcome a typical cold start problem. The use-case at the end of this work evaluates the practical benefit of our general framework to introduce a client-centric, MapReduce-based recommendation engine for real-time recommending music events and festivals.**

Keywords–context awareness, context aware recommendation, decision support, recommendation system.

I. Introduction

Today, the world is annotated by petabytes of digital product and service information distributed across many different ubiquitously accessible global data repositories. Users of various applications and services are constantly submitting additional information or feeding the data repositories with their preferences and experiences. Smartphones and tablets act as a window for viewing and receiving this annotated information as well as to give the users an input device in order to collect additional information. E-business, marketing and e-commerce is profiting a lot by this pervasive use of additional product and customer information. Global marketplaces, such as eBay, Amazon, Apple iTunes or Google Play, offer millions of different products and services in hundreds of categories. These categories span a wide spectrum of product families from traditional hardware to software and mobile apps, eBooks, electronics, video and music streaming or even food. The huge amount of permanent available information makes it difficult or even impossible for users to manually select a relevant subset. As a human user is not able to review all available information, the selection of this subset is of crucial importance for both, the human consumer as well as for the information publishers. The most common real world scenario is a human user searching for a product or service and a huge number of companies offering information on their specific offer. Recommendation engines are one available technique to overcome this information overload and to automatically select a subset of relevant information for a human user. According to the huge number of products available in global marketplaces and the consumers limited time and motivation to check all similar products, recommendation engines provide the necessary rating and pre-filtering for human consumers. Recommendation systems, such as the product recommendation at Amazon or eBay, are already present for several years. Without traditional recommendation systems, the consumer soon gets lost within the huge amount of available products. In a previous work we proposed a general method for combining different context-dimensions along with our general context-model to describe people along with their music interests [1]. In order to solve that problem, global marketplaces soon recognized the need for transparent product recommendation within their systems. In 2006, the Netflix Prize competition was initiated with a 1 million dollar prize for achieving a ten percent or more improvement of Netflix's video recommendation algorithm. The training set that Netflix published for the price competition contained around 100 million ratings from about 500.000 anonymous customers on 17.000 videos. The contest attracted 48.000 competing teams from 182 different countries. The winning team (BellKor) from AT&T Research Labs (made up of Bob Bell and Chris Volinsky, from the Statistics Research group in AT&T Labs, and Yehuda Koren) was able to improve the performance of Netflix's recommendation algorithm by 8.43 percent. So it is obvious that traditional recommendation systems play an important role in modern consumer markets. While recommendation methods for traditional item recommendation, such as Slope One recommendation or Matrix Factorization, have been widely addressed within the last decade, many interesting aspects of client-centric recommendation systems have not been within the focus by the recommendation research community so far. Bell et al. identified several such research aspects during their work on the Netflix prize competition [2]. One of these aspects is to address the client-centric view on recommendation systems, in terms of evaluating and including the consumer's actual context during the recommendation process. Client-centric recommendation system approaches, such as implementations on smartphones and mobile devices need to focus on the user's demands in a tight relation to the users actual situation. For any mobile user the context-dimensions time, location, weather, activity and companions play a major role in any decision. Bell et al. also identified that a combination and blending of several quite simple recommendation approaches often result in excellent recommendations. In this work, we will present the implementation of a software framework that uses a MapReduce programming model approach for distributed data aggregation for blending of multiple context-dimensions. The framework is built on top of a MongoDB noSQL distributed database and uses the Apache Mahout recommendation framework for designing new context-aware and customizable client-centric recommendation models.

The remainder of this work is structured as follows: Section II gives a short overview on state-of-the-art in recommendation systems, music and event recommendation engines, map reduce data aggregation and related work on how to introduce context-awareness in recommendations. Section III focuses on the requirements a general framework for context-aware recommendation systems has to fulfill. Section IV gives an abstract overview on our approach for introducing context-information in traditional recommendation methods and Section V defines a practical software architecture and implementation of our approach. Section VI explains some evaluation results that were collected during the test phase. Section VII concludes with an application case study that introduces context-aware recommendation in the domain of social music and festival events. The last Section VIII discusses general findings, conclusions as well as further research activities.

II. State of the art

The importance of context-awareness in human-centered computing systems has been discussed by various different research communities, including ubiquitous and pervasive computing, mobile computing, e-commerce and e-business, information retrieval and filtering, marketing and management as well as within several engineering disciplines. Through the massive increase of hardware capabilities in combination with cheap broadband access in consumer electronics, such as mobile phones and tablet PCs, the need for context-related information filtering is dramatically increasing too. To discover and evaluate the context and situation of a mobile user is a key challenge within the smart filtering of relevant information out of a huge information space. The term context-aware software was first used in the Xerox PARC research project PARCTAB in 1994 [3]. In this work, the term was specifically dedicated to software that is able to adapt according to its actual location, the collection of nearby people, hosts and accessible devices. Also the possibility to track the changes of context information over time, in other words to store historic situations, was mentioned. Over the years, different research groups enriched this basic definition of context and context-aware software. Brown et al. [4] widened the scope of context information to temperature, time, season and many other factors. Due to the fact that theoretically the number of context information factors is unlimited, the definition of context by Anhind K. Dey is one of the most commonly used:

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves. [5]

This definition of context specifies that context contains any kind of information about an entity in order to understand its situation. Often the term context is limited to location information and location-awareness, but in recent years context also is enriched with the social network of a user. Collecting and evaluating the social dimension of context related to a specific user goes hand in hand with a detailed discussion about privacy and security. An interesting fact about the above definition of context is that Dey identifies three base classes that classify all objects: person, place and object. This kind of classification has practical reasons but is also fixed to a location-dependent view of context information. Over the last ten years several architectures and implementations of software middleware frameworks were published that emphasized the aggregation and interpretation of context-information. In our basic research work from 2003 we already proposed the possibility to use Event-Condition Action (ECA) rules to model the context of an entity [6]. The basic idea behind most of the research activities within context-acquisition, processing and interpretation is to use a user's context information in order to filter relevant information (e.g., on products, services, locations) from the huge collection of available information. An approach that tries to solve the same challenge is to use recommendation methods and algorithms to select a subset of information that seems to be relevant for a user. These recommendation approaches have long tradition within global marketplaces. Traditional recommendation systems take a set U of users and a set of products (items) P, which should be recommended to a user. A recommendation system then provides a utility function f that measures the relevance of a product out of set P to a given user. This utility function f ($f : U \times P \rightarrow R$, where R is an ordered set of numbers) assigns a rating to each item (or even to a compound set of items) in a way that captures the relevance or preference for a specific user. The objective of recommendation systems is to find or learn this utility function f. Function f is used to predict the relevance of items out of P and of new appearing items with similar attributes. In the literature different approaches exist for finding a function f by using an available dataset. Traditional recommendation approaches are distinguished into two major strategies: content filtering and collaborative filtering.

A. Content Filtering

The content filtering approach creates profiles for each item and user, in order to characterize and compare its nature [7]. Each profile contains a specific set of attributes, which can be used to compare objects. For example, a restaurant could have a cuisine attribute, describing the type of food it offers, a location attribute, a vegetarian tag, and so on. A recommendation function f chooses items that are similar to items the user has already chosen or rated before. The utility function compares the user's profile and calculates the similarity of a user profile with the available items. Therefore, the user profile allows the recommendation engine to create a list of items that could fit to a given user profile. Many implementations of this approach additionally refer to Linked Data information, such as RDF stores and Semantic Web repositories, to classify and search systematically for related information.

B. Collaborative Filtering

In collaborative filtering approaches, the recommendation function chooses items that were preferred by other users with similar attributes. Collaborative filtering approaches depend on either explicit or implicit user ratings of items. By rating different items a user can feed explicit ratings into the recommendation engine, while implicit feedback is collected by the system through the analysis of the users behavior (previous purchases, navigation path, search terms, etc.). Collaborative filtering is domain-free, which means that it can be applied to any application area and to different data aspects, which could be hard to formulate into an explicit profile. Collaborative filtering is more accurate than content filtering [7], but has the challenge of starting without any initial data sets (cold start problem). It is not directly possible to address new users or objects where the system has no initial data set available. Popular collaborative filtering methods are neighborhood methods and latent factor models. The Pearson's correlation coefficient $sim(u,v)$ is often used to calculate the popular neighborhood method kNearest Neighbor, in order to measure the similarity between the target user u, and a neighbor v. Within the Pearson's correlation the symbol $\overline{r}_u$ corresponds to an average rating of user u and P denotes the set of products or items.

$$sim(u,v) = \frac{\sum_{i \in P}(r_{u,i} - \overline{r}_u)(r_{v,i} - \overline{r}_v)}{\sqrt{\sum_{i \in P}(r_{u,i} - \overline{r}_u)^2}\sqrt{(r_{v,i} - \overline{r}_v)^2}} \quad (1)$$

Another method uses association rules to explicitly model the dependency and similarity of items. A rule could for example state that if a customer buys item A in combination with item B, then the engine should also recommend to buy item C. One of the most widespread methods for calculating latent factors is matrix factorization, which is described in detail in [7]. Most of the modern recommendation systems use a combination, a so called hybrid approach, of content filtering and collaborative filtering approaches to further improve the accuracy of recommendations. Beside these traditional approaches for implementing recommendation algorithms, several groups are working on the challenge of customizing recommendations and to build flexible recommendation queries. REQUEST: a query language for customizing recommendations was published by Adomavicius et. al. in 2011 [8], which promotes a custom query language to build flexible and customized recommendation queries based on multidimensional OLAP-cubes. Contributions have been made by research groups that built various application scenarios for context-aware recommendation systems, ranging from recommendation of sights within the tourism domain [9], restaurants [10], or even people (e.g., glancee.com).

Calculating recommendations out of a huge amount of distributed data sets also means to handle these distributed calculations within acceptable performance. Typical data sets for traditional recommendation systems consist of millions of ratings and products as well as of hundred thousands of users. This amount of data require strong processing power and a data aggregation method that can cope with distributed and parallel processing of data sets. A popular and stable framework for processing distributed data sets is Apache Hadoop, which builds upon the HBase and implements a software framework that supports data-intensive distributed applications. The underlying HBase is an open source, non-relational, distributed database modeled after Google's BigTable approach and is written in Java.

III. Framework Requirements

This section discusses general requirements for implementing a framework that supports the design of context-aware recommendation systems. To discuss all requirement in detail would exceed the scope of this work, so we focus on several requirements that had a high priority for our use-case in Section VII.

A. Flexible and dynamic customization

A client-centric view on the recommendation process, demands for a flexible user interface to enable the customization and fine tuning of recommendation impact factors for non-technical users. So the users should be able to control the learning and recommendation process at a most fine grain level, while the configuration and presentation should be on an abstract and understandable level. The user should be able to specify a variable number of impact factor dimensions and even to add custom defined impact factors. The framework should normalize all the chosen impact factors and automatically provide a list of recommended items that is sorted according to the weighted sum of normalized impact factors.

B. Temporal aspect

Temporal aspects [11] deal with the change of the context and with the change of the content profiles over a timeline. A recommendation framework has to consider the fact that the importance of specific datasets may change over time. It makes a big difference, if a person has bought an item yesterday or 10 years ago. A general framework has to cope with this varying impact.

C. Transparency

To raise the users' confidence in recommendations, it is of crucial importance to give immediate and transparent feedback on recommendations. The recommendation framework has to provide a human understandable explanation for a given recommendation set. Sundaresan, from eBay research, published a great article about the 6 questions you have to address during the design and implementation of recommendation engines [12] (What, Where, When, Why, Who and How). He also points out that recommendation engines that address the transparency aspect (the Why question), offer a better conversion rate in e-commerce applications. There are several user studies that clearly show that addressing the transparency aspect improves the performance of recommendation engines [13].

D. Performance

The performance of the calculation and delivery of recommendations for a user is one of the most critical non-functional requirements. The acceptance of a user much depends on whether the information is shown at the right time. This is even more important for sensing the context and delivering the recommendation results to a mobile user, as especially this environment is changing a lot within a quite short period of time. Recommendations that consider the location and activity of a user have to react in time to provide recommendations in the specific situation, when a user expects them. As actual recommendation approaches harvest and analyze a huge amount of data, the requirement for performance during the distributed data retrieval and processing is critical for every implementation.

E. Quality

As users are implicitly benchmarking recommendation engines according to the quality of recommendations they are able to provide, it is necessary for a general framework to provide a standard approach for evaluating the quality of recommendation engines. A framework has to provide implicit and explicit quality evaluations, which means that the framework constantly evaluates the quality of results by using test data sets, as well as to explicitly ask the users for quality feedback.

IV. Approach

Within the scope of this work a general approach for the implementation of context-aware recommendation systems is presented. This approach mainly proposes to introduce a map-reduce programming model for processing large context information data sets with a parallel and distributed cluster of noSQL databases. The combination of highly dynamic context information with traditional recommendation algorithms puts high demands in particular on the performance of calculations as well as on the performance of data aggregation. Especially within the process of combining and aggregating raw sensor information, to gain abstract context information, the efficiency and performance of clustered data aggregation is a critical aspect. A general approach for context-aware recommendation systems has to define the impact of context related, dynamic information on the recommendation process. Compared to the traditional recommendation approaches, which were already discussed in Section II, we combine these traditional collaborative filtering approaches with user related context-information. This also means that for each individual application scenario there exists a quite specific collection of context aspects that offer high relevance for the recommendation of entities in a certain situation. While the location information might not be relevant for recommending books in an online bookshop, it is of crucial importance for the recommendation of nearby restaurants. Within our proposed approach, each dimension of a given context, such as location, weather or companions is represented through an impact function. An impact function defines the influence of one dimension of a given context on the overall relevance within the recommendation process. All impact functions are of the given form $f_i : U \times P \times C \to R$, where U represents the set of users, P the set of products and C a dimension of a given context (e.g., location, number of nearby friends, etc.). The weighted sum of all normalized impact functions results in an overall relevance for a product p, a given user u and context c, where w_i represents a weight that the end user defined for a specific dimension of the context:

$$f(u,p,c) = \sum_{i<P} f_i(u,p,c)w_i(c) \tag{2}$$

A general framework for context-aware recommendation systems has to offer the basis for customizable recommendation engines that consist of a variable and dynamic set of impact functions that can either be predefined by the framework (e.g., aspects such as distance, user ratings, history, friends, ...), or explicitly defined by users. Furthermore, the users are able to dynamically define and adapt the weight of different impact functions according to their preferences, which is shown in Figure 1. As Figure 1 shows, the customized recommendation system within this example contains six different impact functions. Each of these impact functions calculates the relevance

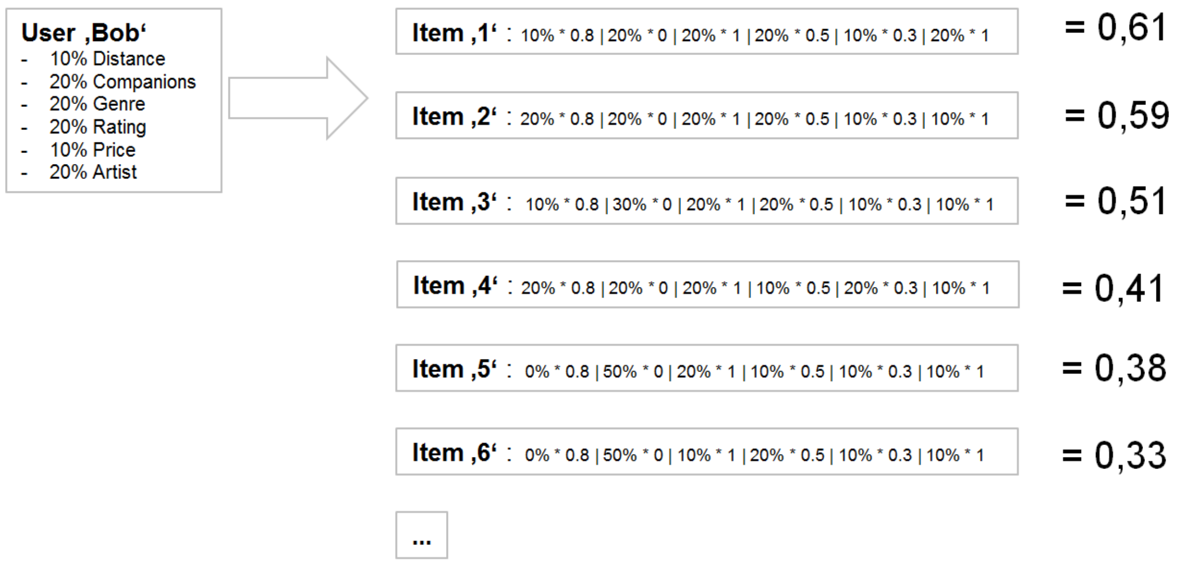
User ‚Bob'
- 10% Distance
- 20% Companions
- 20% Genre
- 20% Rating
- 10% Price
- 20% Artist
Item ‚1' : 10% * 0.8 | 20% * 0 | 20% * 1 | 20% * 0.5 | 10% * 0.3 | 20% * 1
= 0,61
Item ‚2' : 20% * 0.8 | 20% * 0 | 20% * 1 | 20% * 0.5 | 10% * 0.3 | 10% * 1
= 0,59
Item ‚3' : 10% * 0.8 | 30% * 0 | 20% * 1 | 20% * 0.5 | 10% * 0.3 | 10% * 1
= 0,51
Item ‚4' : 20% * 0.8 | 20% * 0 | 20% * 1 | 10% * 0.5 | 20% * 0.3 | 10% * 1
= 0,41
Item ‚5' : 0% * 0.8 | 50% * 0 | 20% * 1 | 10% * 0.5 | 10% * 0.3 | 10% * 1
= 0,38
Item ‚6' : 0% * 0.8 | 50% * 0 | 10% * 1 | 20% * 0.5 | 10% * 0.3 | 10% * 1
= 0,33
...

Fig. 1. General approach for a weighted combination of context with collaborative filter dimensions

of a product for a given user and context dimension. As it was already mentioned, a context dimension could be the distance, friends (companions), product category, ratings, or prices. This multidimensional approach is not limited to a fixed set of impact functions, but can be enhanced by including additional context dimensions. Therefore, a designer of a domain specific recommendation system has to provide a domain-specific set of additional functions, in order to improve the quality of recommendations for the users in different application domains. The radar chart in the lower left corner of Figure 2 visualizes the weight an individual user defined for a given set of impact functions. Each user is able to specify his own, very personal weight of each context or collaborative filter dimension. The example setting shown in Figure 2 defines a typically high weight for the distance between the user and a given event (here given by 30% impact), 5% impact for the price of an event and moderate weight for the artist (20%) performing an event, companions and friends coming along(15%) and the collaborative filtering result (20%). By giving the user the possibility to define his own personal weights it is possible to modify the recommendation result on a quite fine granular level. An important aspect of this approach is also the ability to calculate the impact of each context dimension by using completely different strategies. While the impact of location and distance could be calculated through a simple spatial query, the rating impact function could be implemented as traditional collaborative filtering approach. These individual impact functions are then calculated by using a map-reduce programming model approach. The huge amounts of raw data sets are collected within several parallel map steps and aggregated in subsequent reduce steps until the result is fully available. The following section describes in detail the overall system architecture as well as the implementation details of this approach.

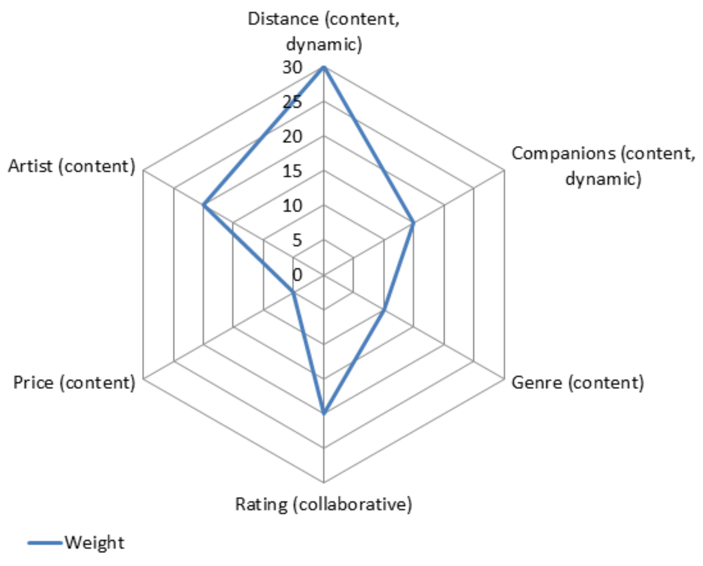

Fig. 2. User defined weight for available context and collaborative filter dimensions

V. IMPLEMENTATION

The general software architecture for building a context-aware recommendation system is derived into a typical client-server architecture model. This client-server architecture uses a MapReduce programming model, as it was already mentioned in the general approach in Section IV along with several critical subsystems. As it is shown in Figure 3, the server defines all necessary subsystems for data access and third-party information retrieval, user interfaces for manual content selection and correction, as well as the context-sensitive

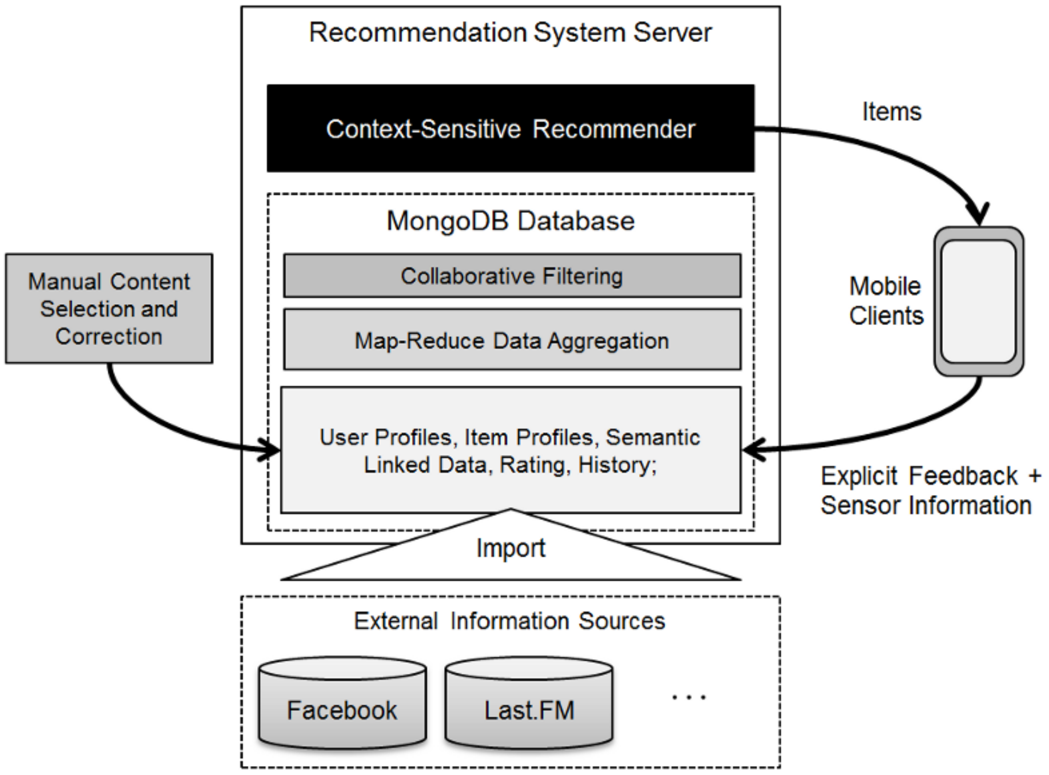
Recommendation System Server
Context-Sensitive Recommender
Items
MongoDB Database
Collaborative Filtering
Map-Reduce Data Aggregation
User Profiles, Item Profiles, Semantic Linked Data, Rating, History;
Manual Content Selection and Correction
Mobile Clients
Explicit Feedback + Sensor Information
Import
External Information Sources
Facebook
Last.FM
. . .

Fig. 3. General software architecture for the implementation of a context-aware recommendation system

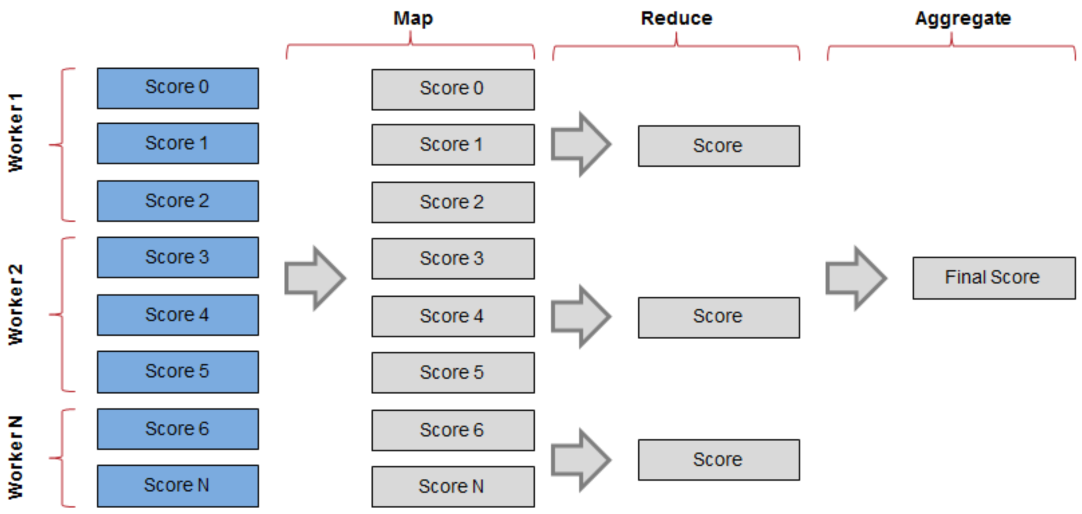

Fig. 4. The MapReduce data aggregation process

recommender. Main part of the server implementation is the management of raw context information along with bound user and item profiles. Semantic data that is used to calculate the similarity and relation between given users or items is stored by using linked data repositories. All additional semantic data can be accessed by using semantic web standards and query languages, such as RDF and SPARQL. The purpose of accessing these sources of semantic information is to receive additional item-based similarity measurements that are used in combination with traditional collaborative filtering result. External sources of semantic information, such as Facebook or Last.FM, are either directly imported and duplicated, or directly accessed through a defined service interface. The decision if an external information source is either imported or directly accessed depends on the third-parties' service level agreements. On top of the management of raw context and profile information the map-reduce data aggregation layer is responsible for collecting and aggregating these raw information into abstract context-information. Within the map-reduce layer several individual map-reduce processes are calculating normalized context-dimensions that are combined to a common rating table between users and items. The MapReduce programming model defines two fundamental steps: Map and Reduce. During a parallel Map step all distributed databases collect the available data sets. Following code shows a typical structure implementing a Map-function within the noSQL database MongoDB:

```
function()
{
    emit(
    {
        user_id : this.user_id,
        item_id : this.ref_id
    },
    {
        score : this.score
    }
    );
}
```

Within this Map function all database sets are collected that contain a score between users and items and emitted as intermediate result. The Reduce step gathers all these intermediate results and calculates an overall score for all user item relations, as it is shown within the following example:

```
function(key, values)
{
    var total = 0;
    for ( var i = 0;
          i < values.length;
          i++ )
    {
        total += values[i].score;
    }
    return { score : total };
};
```

After the last Reduce step has been performed, the resulting data set is organized as a sparse score matrix between individual users and available items. The collaborative filtering layer on top uses traditional recommendation engines to fill the missing gaps within this sparse user-item score matrix. Typical algorithms used within the collaborative filtering layer are slope one recommendation or matrix factorization methods. Figure 4 visualizes the stepwise, distributed MapReduce process for parallel data aggregation.

A. Cold Start Problem

Another important implementation detail is how to handle and avoid the initial cold-start problem that is typical for collaborative filtering solutions. Within our approach a combination of content based filtering with aggregated substitute ratings is used to fill the gap of missing explicit user ratings. As the relevance of social events for a person is very much related to the actual distance, the recommendation engine first uses a sorted list of distances in combination with selected indirect factors that are extracted from the users contexts. As the initial system not only lacks of a large number of explicit user ratings but also of a large number of users, additional user information is gathered from connected Facebook profiles. These passive user profiles are not considered as active participants of the system but act as a critical mass for calculating hidden factors within given user/item ratings. According to the given software architecture and sensor availability, we selected the following indirect factors for calculating an aggregated substitute rating:

Library
: The smartphone of the user contains a given music artist. Some mobile platforms also provide some simple statistics about the last time the user listened to that artist or how often.

Forecast
: A forecast is given if the user expects to visit an event and shares that information with his friends.

Checkin
: Similar to other widespread location-aware social platforms, like Foursquare, a Checkin is explicit information that the user arrived at a given event.

View
: Several different view statistics record the users history of reading artist, event or venue information.

An aggregated combination of these indirect rating factors is then used to fill the sparse rating matrix. The metamorphosis between indirect ratings and explicit user ratings is a continuous process in which explicit user ratings iteratively replace the substitute values whenever available.

As a result of the collaborative filtering process, the server database contains a matrix of given ratings, user and product profiles as well as additional semantic information. The recommendation module is responsible for combining the different

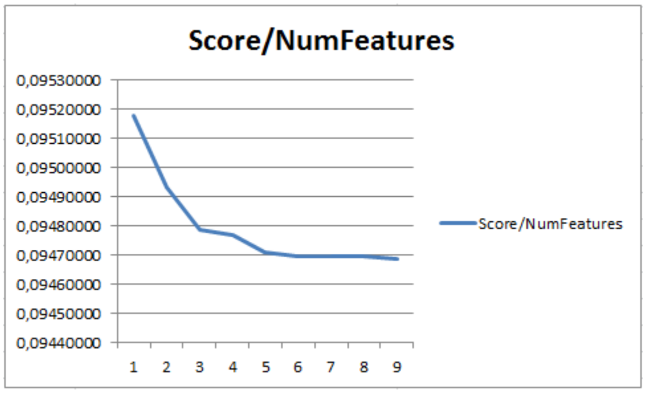

Fig. 5. Evaluation score by a varying number of features

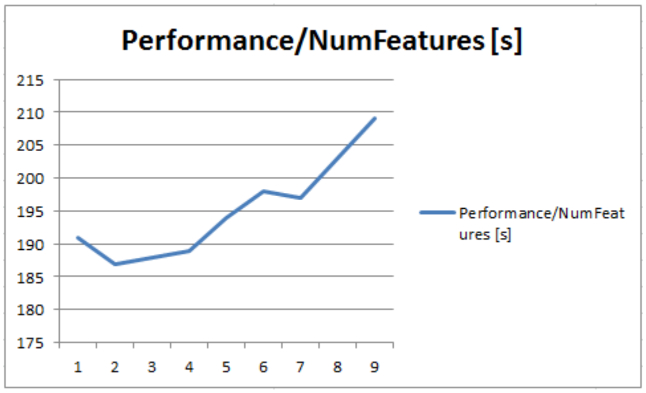

Fig. 6. Evaluation of performance by a varying number of features

dimensions of the recommendation approach in Section IV and to communicate the resulting ratings to the clients. The client-server communication is implemented as a lightweight REST (REpresentational State Transfer) service approach. On the client-side, a local application is visualizing the resulting list of recommendations and is collecting the necessary context information in combination with the user's feedback on the given recommendations.

VI. Evaluation

Within this evaluation section some detailed results in terms of recommendation quality and performance are visualized. The quality of recommendation results within this work is evaluated by calculating a quality score by using an average absolute difference evaluation method (mean average error score) that divides the available ratings into 70% training data and 30% evaluation data. The collaborative filtering method used within this evaluation is based on a matrix factorization projection of given users and event items onto a feature space. Figure 5 shows the evaluation score with varying number of features (starting with one feature to nine features). Our evaluation showed that increasing the number of features above an amount of five does not significantly improve the overall recommendation score. At this stage of our evaluation data, the evaluation score itself does not provide any useful information as most of the ratings are binary ratings automatically extracted from passive user profiles. So the range of the ratings are between zero and one.

Beside the evaluation of how the number of features improves the recommendation score we also evaluated the performance of the overall calculation. The performance tests were done on a standard Windows 7 Laptop device with 4GB RAM and a Intel Core i5 64bit CPU with 1.70GHz. Figure 6 shows that the overall calculation of the weighted recommendation ratings by using the discussed MapReduce programming model is performed within around 190 seconds. The database contains 408,634 passive and active user profiles and 22,901 different events (items). After the MapReduce programming model aggregated the scores, as it was shown in Section V the resulting number of ratings is 399,905.

The evaluation of this framework shows that it is a valid concept and approach for implementing a context-sensitive recommendation engine that uses a MapReduce programming model in combination with collaborative filtering. The evaluation of the quality of recommendations does not provide any significant results as the data set does not contain enough explicit user ratings so far.

VII. Use-Case: Evntogram

The following use-case was selected out of a running project in cooperation with EVNTOGRAM, which is a platform operator for personalized and context-sensitive recommendation of music events. The philosophy of EVNTOGRAM is to analyze the users' music favorites and activities, as well as their social interaction, in order to offer personalized and context-aware recommendations for events, specifically in the domain of music events, such as concerts and music festivals. A recommendation approach, explained in Section IV and Section V, helps to include various context-dimensions into the calculation of the relevance of an event for a given user. EVENTOGRAM records these context-dimensions, such as the users' activities, social interaction, music listening habits and individual ratings, in order to sort a list of music events according to the calculated relevance, as it is shown in Figure 7. In a first prototype EVENTOGRAM is trying to find out, which subset of context-dimensions is providing good recommendations for the users. In that sense 'good' means the feedback the user is providing for a given ordering of items. After the initial release of the EVNTOGRAM client app for the platforms Android and iOS recommendation calculations were performed for around 500 active and more than 400,000 passive user profiles. Passive user profiles represent profiles that come from external sources, such as Facebook and help to overcome the so called cold start problem. The operation of the EVNTOGRAM platform within the last month returned one critical user review concerning the degree of transparency for event recommendations. The criticism is related to the quite unclear process of proposing events by our recommendation engine to the users. A detailed explanation of the recommendation process would help the user to understand why a rating

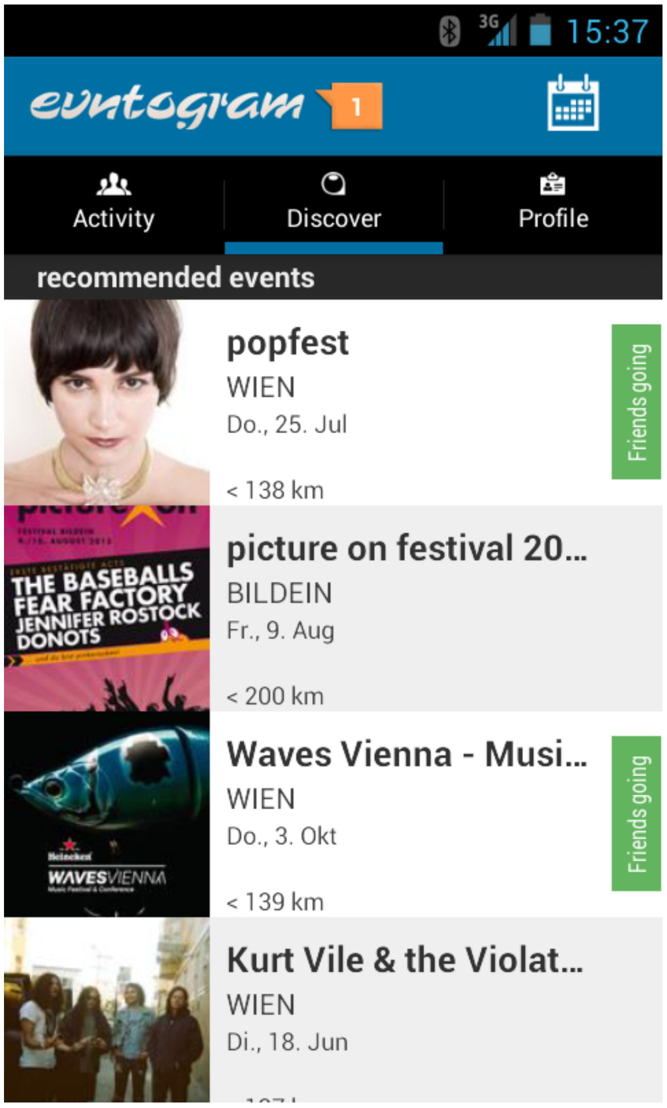

Fig. 7. List of recommended Music Events delivered by the EVNTOGRAM Android Client

for an event was calculated.

VIII. Conclusion

In this work, we propose a general MapReduce-based approach, as well as software architecture for the implementation of context-aware recommendation systems. The approach as well as the framework offers high flexibility according to the definition and configuration of new context dimensions in form of impact functions, which influence the recommendation of items for given users. The framework is domain-free, which means that this approach can be implemented and adapted for different application domains. The context-aware recommendation of items of all kind, ranging from products in e-commerce to activities and services in sport and fun will get much attention in future software development. A customization of a domain-specific recommendation engine on top of our proposed approach could be implemented with reduced development effort, as it is mainly reduced to a simple selection of context dimensions. We think that a general framework for designing and implementing such recommendation systems for different application domains is of great importance. The next steps within our work will be to gather empirical feedback from the community within the given use-case of recommending music related events and to improve the degree of transparency for the recommendation process.

References

[1] W. Beer, W. Hargassner, S. Herramhof, and C. Derwein, "General framework for context-aware recommendation of social events," in *Proceedings of the Second International Conference on Intelligent Systems and Applications (INTELLI)*. IARIA, 2013, pp. 141–146.

[2] R. M. Bell, Y. Koren, and C. Volinsky, "The bellkor solution to the netflix prize," accessed: 31/01/2013. [Online]. Available: http://www2.research.att.com/ volinsky/netflix/ProgressPrize2007BellKorSolution.pdf

[3] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on Mobile Computing Systems and Applications*. IEEE, 1994, pp. 85–90.

[4] P. J. Brown, J. D. Bovey, and X. Chen, "Context-aware applications: From the laboratory to the marketplace," *IEEE Personal Communication*, vol. 4, no. 5, pp. 58–64, Oct. 1997.

[5] A. Dey and G. Abowd, "Towards a better understanding of context and context-awareness," in *CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness*, 2000.

[6] W. Beer, V. Christian, A. Ferscha, and L. Mehrmann, "Modeling context-aware behavior by interpreted eca rules," *Euro-Par 2003 Parallel Processing*, pp. 1064–1073, 2003.

[7] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Computer*, vol. 42, no. 8, pp. 30–37, Aug. 2009.

[8] G. Adomavicius, A. Tuzhilin, and R. Zheng, "Request: A query language for customizing recommendations," *Info. Sys. Research*, vol. 22, no. 1, pp. 99–117, Mar. 2011.

[9] W. Beer and A. Wagner, "Smart books: adding context-awareness and interaction to electronic books," in *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia (MoMM)*. New York, NY, USA: ACM, 2011, pp. 218–222.

[10] V.-G. Blanca, G.-S. Gabriel, and P.-M. Rafael, "Effects of relevant contextual features in the performance of a restaurant recommender system," in *In RecSys11: Workshop on Context Aware Recommender Systems (CARS-2011)*, 2011.

[11] Y. Koren, "Collaborative filtering with temporal dynamics," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*. New York, NY, USA: ACM, 2009, pp. 447–456.

[12] N. Sundaresan, "Recommender systems at the long tail," in *Proceedings of the fifth ACM conference on Recommender systems (RecSys)*. New York, NY, USA: ACM, 2011, pp. 1–6.

[13] R. Sinha and K. Swearingen, "The role of transparency in recommender systems," in *Extended Abstracts on Human factors in Computing Systems (CHI EA)*. New York, NY, USA: ACM, 2002, pp. 830–831.