Disconnected Query Evaluation and Proactive Caching in Distributed Mobile Recommender Systems

Frederik Leonhardt

Saarland University, Campus, 66123 Saarbrücken, E-Mail: frederik.leonhardt@stud.uni-saarland.de

Andreas Emrich

German Research Center for Artificial Intelligence (DFKI), Institute for Information Systems (IWi), Stuhlsatzenhausweg 3, 66123 Saarbrücken, E-Mail: andreas.emrich@dfki.de

Alexandra Theobalt

German Research Center for Artificial Intelligence (DFKI), Institute for Information Systems (IWi), Stuhlsatzenhausweg 3, 66123 Saarbrücken, E-Mail: alexandra.theobalt@dfki.de

Peter Loos

German Research Center for Artificial Intelligence (DFKI), Institute for Information Systems (IWi), Stuhlsatzenhausweg 3, 66123 Saarbrücken, E-Mail: peter.loos@dfki.de

Abstract

A slow or unavailable internet connectivity in remote scenarios with possibly poor network availability forces permanently available mobile recommender systems to utilise disconnected query evaluation and proactive caching. The need emerges to design robust systems, which can deliver reliable results even without existing internet connectivity. This paper will demonstrate an efficient approach for proactive caching and disconnected query evaluation to support a mobile recommender system in monitoring critical local data of its users in real-time. A use case scenario to support in-training recommendations bound to vital parameters of the trainees is shown, and finally performance metrics are discussed for the caching and recommendation mechanisms.

1 Introduction

With the increasing popularity of mobile services and recommender systems more and more people use them on a daily basis. This leads to scenarios in which users rely on the availability of such services even in critical situations at places not covered by internet connectivity, e.g. when pursuing recreational activities in forest areas and national parks. Permanently available mobile recommender systems aim to support users in such situations by delivering recommendations with reasonable quality.

Recent approaches on mobile caching have concentrated on low-level optimization like minimising down- and uplink traffic between server and clients [19], [16] and simple time-based cache invalidation

[7], whereas latter work explores more sophisticated cache replacement strategies incorporating location information and other criteria, which are evaluated by the use of arbitrary functions to determine candidates for cache eviction.

This paper presents cache invalidation and replacement strategies suitable for disconnected query evaluation in a mobile, distributed recommender system. In this scenario, a mobile client needs to recommend data items based on its local cache, which might contain stale data since no connection to the server has been available for a certain period of time. The first goal of this work is to find a way to accurately and eagerly cache items while a server connection is available, which then can be accessed by the client to generate recommendations. The second goal is to minimise the quality loss of query responses caused by stale items.

Users in mobile environments often query for location-dependent information, and mobile recommender systems need not only to incorporate user profiling information, but also location-dependent context information to provide useful feedback. To support the system, context processing can be backed by one or more specific ontology domain models [6].

This work applies the design science methodology [4] and presents an innovative artefact based on the review of existing literature in the related areas of caching and disconnected query evaluation. In the course of this paper the author describes the design and implementation of a proof-of-concept solution for a disconnected mobile recommender system.

The rest of the paper is organised as follows: Section 2 contains a review of related work in the areas of caching and mobile recommendation. Requirements arising for our approach are discussed in Section 3 Section 4 describes the architectural decisions and Section 5 the implementation of a proof-of-concept prototype. The results of an evaluation on cache performance are presented in Section 6. Section 7 concludes the paper and discusses possible future work on this topic.

2 Related Work

In the following different approaches for cache retrieval, replacement and invalidation are discussed with respect to their suitability for mobile, location-dependent data and their applicability to distributed and offline recommendations.

2.1 Cache Retrieval Strategies

To perform offline query evaluations, complex data objects need to be cached on the mobile client. A standard approach used to describe which values should be cached is based on the concept of *Cache Groups* [11]. They can be organised in object structures, but don't necessarily need to reflect whole data objects. A hybrid caching approach, where only certain values of objects are cached can be useful to maintain a small cache size [12]. Values defined by the cache groups are always fetched and stored together. By utilising cache groups the performance of a database can be increased [3].

While cache groups help to fetch all related data in chunks, they lack intelligent data prefetching and proactive caching capabilities. In mobile environments such prefetching decisions can be made based on location-aware, semantic caching strategies. They usually utilise factors like the movement direction of the user, distance to the data object, and the data object's valid scope. Chim et al. propose a model with viewer and object scopes and a movement direction prediction. Based on this model they evaluate a prefetching mechanisms for objects in virtual environments, which proves to be performance effective

[13]. Chan et al. utilise a mobile ad hoc network to make movement predictions by determining the probability and length of a stay in a given valid scope [1].

2.2 Cache Replacement Strategies

A cache typically has a maximum capacity, and upon reaching this limit existing values have to be evicted from the cache before new values can be stored. The algorithm used for the selection of eviction candidates is referred to as *Cache Replacement Strategy*. Those strategies are usually optimised for minimal cache-miss or maximal cache consistency. Existing cache replacement approaches for mobile environments consider not only server-side constraints, but also heuristics like network delay and anticipated access times [9], [18]. Additionally location-aware cache replacement strategies have been described.

Plenty of temporal cache replacement strategies exist, mostly designed for the caching needs of WWW data and stationary clients. They include traditional policies like *LRU*, *LFU* and *LRU-k*, which assume that recently accessed items have a high probability of being accessed again and frequently accessed items will also be retrieved often in the future [7], [18], [20]. Other approaches use access probability and broadcast frequency like *PIX*, access history and retrieval delay like *Gray* or are using an "optimal" strategy like *Min-SAUD*, which in practice turned out to depend too much on heuristics [10]. Those assumptions do not hold up for mobile clients without restrictions as they tend to show a strong relation to location-based queries [9], [18] as location-based services are becoming more and more popular [20].

In response to this development several mobility-aware cache replacement have been proposed. One of the earliest strategies is *FAR (Furthest Away Replacement)*, where data objects are partitioned in two sets based on the movement direction of the client. Items not in the direction of movement are always evicted first [9]. The *PA (Probability Area)* strategy tries to combine temporal and spatial data by taking into account valid scopes of objects (i.e. spatial areas, in which objects are valid) and their access probability [9]. Zheng et al. propose a location-dependent cache replacement strategy named *PAID(-U/-D)*, which adds data distance, valid scope and movement direction to the calculation of the access probability of an area.

All those strategies come with a common weak point: They either exclusively consider temporal or location-based data, or a combination thereof. Other important factors are not weighted into the equation, like the cost of retrieving an item over a wireless channel [18]. Lai et al. propose *MARS*, a cost-based mobility-aware replacement scheme which proves to be efficient [18]. The *SMaC* approach is optimised to cache map tiles by utilising two separate temporal strategies (LRU and LFU) on super-tiles containing adjacent map tiles [20], which outperforms a pure LRU strategy due to the knowledge of the tile structure and the assumption that adjacent tiles have a higher probability of being visited.

2.3 Distributed Recommendations

Distributed recommendations can be divided in two categories: Those with partial query evaluation on the client and those with offline evaluation on the client. Hu et al. propose a system for partial query evaluations for spatial queries in mobile environments [16]. First, the client executes a query based on a cached index. If necessary, a *remainder query (RQ)* is submitted to the server for additional evaluation. This approach is combined with a semantic, proactive caching strategy. In the case of no RQ the approach can be seen as a disconnected query evaluation, since no server connection is necessary and the caching strategy provides the system with sufficient data [16].

3 Requirements

As the discussion of related work has shown, none of these approaches offer a consistent solution for the depicted problem space. It has shown that a variety of cache replacement strategies already exist, but that location-based strategies are often tightly associated to their use case. Some cell-based strategies for the use in mobile cellular networks or ad hoc networks have been proven very efficient in those areas [8]. However, existing solutions only have limited use in proactively caching data items, since they assume to have a mostly steady connection to the backend server. To make an optimal selection of location-dependent caching items, several location forecasting models have been discussed. They generally use heuristics to determine the probability of a user entering a certain spatial area and can be extended to fit our use case (cf. Section 5). Cache groups can help to fetch all related values of a "recommendation" item, but reduce the overall item size by omitting uninteresting data.

An offline recommendation scenario can only be beneficial for the user if the mobile engine is able to exploit real-time data which cannot be transferred to a server easily. For example, this applies for vital data produced by sports equipment like heart rate and ECG. Such sensors are gathering data several hundred times a second (e.g. an ECG sampling frequency of 1000 Hz as described in [21]) and often only a slow mobile connection or no connection at all is available on-the-go.

This leads to the requirement of a server-backed, but yet independent mobile recommendation engine (*R1*), which can select interesting data items through a combination of forecasting models and proactive caching (*R2*).

- **R1:** Design of an offline recommendation engine to perform evaluations based on pre-cached items and live sensor data.
- **R2:** A proactive caching strategy for interesting location-dependent data items to use with disconnected query evaluations.

Existing work shows that specialised partial evaluations already have been described for spatial queries, where *remainder queries* may be evaluated on the server side. In contrast to this approach, we propose a partial evaluation on the client side. Based on user data, the server sends a partially evaluated query to the client, which then incorporates local and time-critical data like current sensor data. To utilise such partial evaluations, the mobile engine needs to query a backend server occasionally. Therefore a full client-server architecture needs to be developed (*R3*).

For partial query evaluations on the mobile client it is not necessary to preliminarily cache all interesting objects. Through the existing connection data items can be invalidated via regular IR broadcasts. This reduces the risk of decreasing the recommendation quality by resting upon stale values for the query evaluation. In this case an efficient replacement strategy needs to be designed, which minimises the need of server polls (*R4*). A similar forecasting technique like proposed for *R2* can be used to make an optimal item eviction decision.

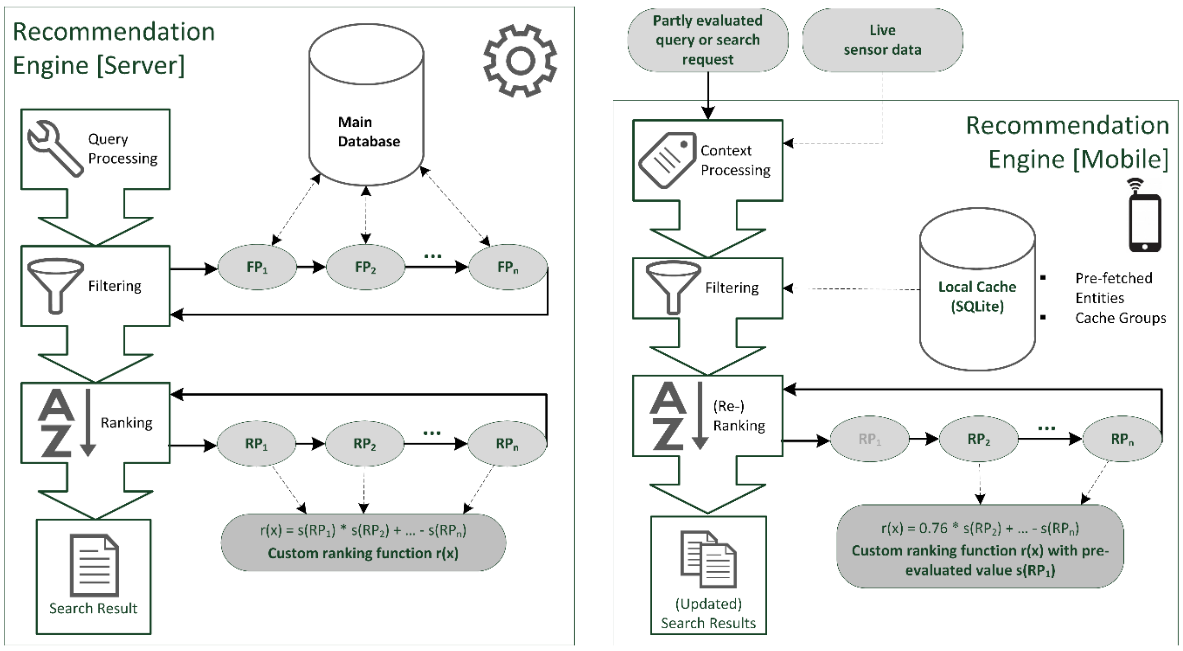
- **R3:** Allow a partial evaluation of queries on the mobile device, which requires a client-server architecture.
- **R4:** An efficient location-based cache replacement and invalidation strategy for partial evaluations on the mobile client.

4 Conceptual design

This section describes the conceptual design of the proposed mobile recommendation system. First, the server-client framework connecting mobile devices with a recommendation backend server is presented. Then, the recommendation engine itself is discussed. Two aspects of the engine are specified separately, the mobile and the server component. Finally, the mobile caching approach in our system is described and the used cache invalidation strategy is briefly discussed.

4.1 Server-client architecture

To meet requirement *R3* the system has been designed with a client-server architecture. This allows the server to partly evaluate queries and send the results to the client for further processing. The server maintains the database with all data items, implements a recommendation engine and the cache manager. This allows to perform data mining operations on the collected, historical data, and results thereof can be used by server-side ranking plugins to improve the recommendation quality. The server can send partly evaluated search results to the client, and the cache manager is responsible for proactive updates to the client's cache and invalidation of its entries. The client has a mobile recommendation engine and a local cache. It can perform offline recommendations or complete search results from the server. The caching component can request entities and cache updates from the main database on the server.



Recommendation Engine [Server]
Main Database
Query Processing
Filtering
FP₁
FP₂
FPₙ
Ranking
RP₁
RP₂
RPₙ
r(x) = s(RP₁) * s(RP₂) + ... - s(RPₙ)
Custom ranking function r(x)
Search Result
Partly evaluated query or search request
Live sensor data
Recommendation Engine [Mobile]
Context Processing
Filtering
Local Cache (SQLite)
Pre-fetched Entities
Cache Groups
(Re-) Ranking
RP₁
RP₂
RPₙ
r(x) = 0.76 * s(RP₂) + ... - s(RPₙ)
Custom ranking function r(x) with pre-evaluated value s(RP₁)
(Updated) Search Results

Figure 1: Design of the recommendation engine: (a) server-side engine with fully evaluated query, (b) client-side engine with partly re-evaluated query utilising local cache.

4.2 Recommendation Engine (Server)

Fig. 1(a) depicts the three main parts of the server-side recommendation engine: Query processing, filtering and ranking. They are designed to be modular and each component is extendable by custom plugins. Additionally, the recommendation engine is fully configurable at runtime. This includes any plugins, the ranking function and cache groups. Some background on the server-side framework for context-aware mobile recommendations can be found in [14] and concrete implementations of filter and ranker plugins have been discussed in [15].

The query processor performs tokenising and stemming of search terms and enhances the query with additional semantic data, supported by an ontology domain model. In the filtering step the plugins select a suitable set of object candidates from the database, which then is handed over to the ranking. Every filtering plugin FP can extend a criteria query to add additional restraints. The ranking mechanism on the server needs to be designed to support partial evaluations (cf. Section 3, requirement *R3*). This is shown in Fig. 1, where each ranking plugin RP contributes to the main ranking function r_T. In the end, a query is either in a completely evaluated state or a partly evaluated state and can be transferred to the client for further processing. The ranking functions are described in detail below.

$$r: T \rightarrow [0,1], t \mapsto s \tag{1}$$

Equation 1 shows the definition of a ranker's function r, which maps a given instance t of object type T to a score s. In the next step this score s is required to be normalised to [0,1], and the ranker is responsible to return such a normalised score.

$$r_T: [0,1]^n \rightarrow \mathbb{R}, \vec{s} \mapsto x, with\ \vec{s} = (s_{1,} \cdots, s_n)^T \tag{2}$$

Furthermore, the system allows the definition of an arbitrary ranking function $r_T(s_{1,}s_{2,} \cdots, s_n)$ where T is the object type, n the number of rankers, and $s_{1,} \cdots, s_n$ are the individual ranker scores for a given instance t of T. Each object type T may run with its individual ranking function and a set of unique rankers. Eventually r_T is used to calculate an aggregated ranking score for each object t in the ranking pipeline. In Fig. 1 an example is given by a ranking function $r(x)$, which multiplies the first two ranking scores (e.g. for a location-based falloff), then adds the next scores and subtracts the last ranking score.

In case of a disconnected evaluation, the server would populate some of the scores but leave others unassigned. Those unassigned values then can be filled at runtime by the client, e.g. by mobile rankers which take current sensor data into account. This scheme allows computationally intensive calculations and evaluations which need knowledge of exhaustive amounts of data to be performed on the server. This response can be cached by the client and be re-evaluated continuously.

4.3 Recommendation Engine (Client)

Fig. 1(b) depicts the three main parts of the client-side recommendation engine: Context processing, filtering and ranking. Like on the server, the ranking is extendable by custom plugins. But on the client, it needs to support disconnected queries (i.e. offline recommendations) and partial re-evaluations of existing results coming from the server as specified in requirement 1 in chapter 3.

Whenever an evaluation is triggered, the necessary data is collected in the context processing step. It takes either a search request or a partly evaluated result and the current sensor readings. Analogous to the filtering step on the server, a suitable set of object candidates is selected from the local cache for further processing in the ranking plugins. The ranking step is very similar to the server-side in terms of mathematical processing. But opposed to the server, only ranking plugins eligible for offline evaluation are executed, e.g. an already pre-computed value (by the server) of RP_1 can be seen in Fig. 1(b).

4.4 Mobile Caching

The cache design on the mobile client is just covering the read-only case, we assume the client does not change any recommendation objects. Object updates will only be performed by the server and can be propagated via the cache invalidation strategy. This component is designed to fulfil requirements *R2* and *R4*. Cache updates can either be triggered by client update requests, or proactive updates originating from the server. The cache group definitions are used to extract all required data from the database and to transform the data into the respective cache group format. When configuring the ranking plugins, the developer can define which cache groups need to be available at runtime for a successful execution of the given plugin.

Cache retrieval and replacement strategies are depending on the actual data model and use case and are described separately in Section 5. As pointed out in Section 3 the cache retrieval and replacement policies will be using a similar function to determine cache members and evictions respectively. The function should be configurable and not restricted to time-, access- or location-dependent factors.

The cache invalidation strategy used is a hybrid poll and broadcast-based solution. The mobile recommendation engine accepts a certain amount of stale data in its cache, so it will not wait for an IR to evaluate waiting queries. This is absolutely necessary for the case of offline recommendations, where an IR may not arrive at all. The server sends simple VIR broadcasts in fixed intervals to notify clients of data changes. Additionally, the client may poll for a cache update any time, e.g. when the user decides to initialise the system for offline usage.

5 MENTORbike: A prototypical Proof-of-Concept

The presented approaches are specifically useful in situations that rely on location-dependent, fast-changing contextual data, which have decisive impact on recommendation results. A possible application area includes mobile health monitoring scenarios.

MENTORbike is a system utilising a pedelec (a bicycle where the rider's pedalling is assisted by an electric engine), a wireless body area network (WBAN) of sensors, a smartphone and a web portal to support users with their fitness training [2]. It allows the cyclist to train with a steady heart rate or pedalling power while cycling outdoors instead of using a traditional ergometer in a gym. Additionally it is aimed at helping rehabilitation patients suffering of cardiovascular disease to automatically maintain their vital thresholds (e.g. heart rate) assigned to them by a health professional. Here disconnected query evaluation is crucial: For rehabilitation patients it is important to forecast possible developments of vital parameters according to the route or weather conditions (e.g. “*If you ride up that hill, your heart rate will surpass the defined threshold of 140 BPM”* or “*Temperature will rise by 5°C in the next hour, causing your blood pressure to rise beyond the defined threshold”*).

For the described scenario two recommendation objects are of interest. Firstly the necessary engine power for the next route segments should be recommended. This value mainly depends on the route characteristics, but also personal information like patient classification, heart rate safety limits and training history may influence the support a user needs to receive. Most of the data can be evaluated on the server even before starting the training, but vital sensor data has to be monitored throughout the usage so that the engine power can be adapted if problems occur. A safety mechanism needs to work even without internet connectivity: If the user overexerts in a rehabilitation environment and the engine cannot balance the strain, the system needs to issue a warning to both the responsible trainer and user.

The second use case describes recommendations of point of interests (POIs) and actual route segments. However, in this paper we concentrate on POI recommendations exclusively. A POI recommendation is highly location-dependent, but also involves profile information of a user (e.g. *“The user likes old castles.”*) and time-dependent information (*“The castle can be visited from 10am to 8pm.”*). Profile information and time-dependent information can be pre-calculated into a “fit”-score, which is user-dependent and can be cached on the device. The location information then is evaluated on-the-go, as it keeps updating. An approach to suggest routes and POIs for runners has been described in [15], with the drawback of needing a steady internet connection to adapt recommendations for the user.

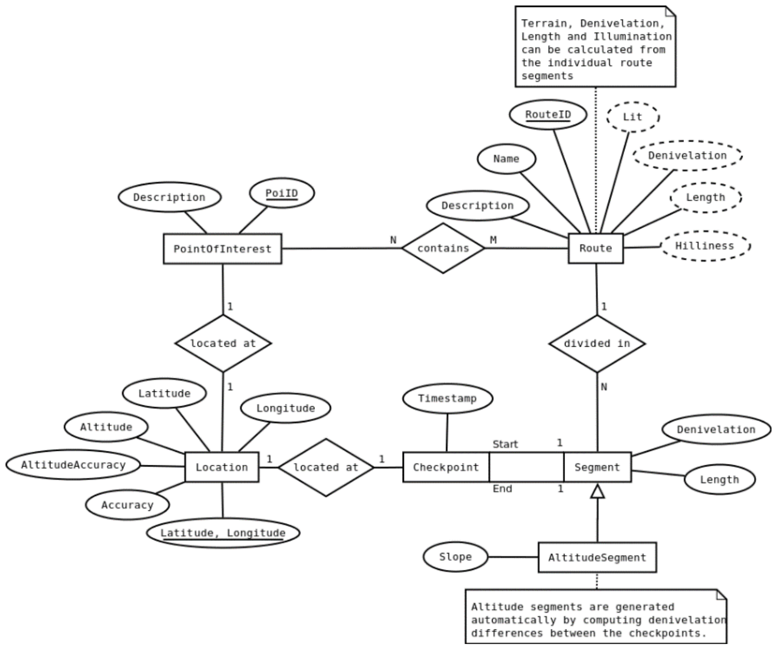


Figure 2: Simplified route data model as ERM

The cache group depicted in Fig. 3 is derived from the route data model in Fig. 2. It simplifies the data model and additionally contains pre-calculated values derived from historical route usage data like heart beat forecast in an altitude segment. This forecast is calculated on the server-side using the exponentially weighted moving average (EWMA) of the last route runs by similarly classified users. Another pre-computed value is the fit factor of a given point of interest (POI) to a user's interest profile. As those values do not change quickly they can be cached together with the route data. They are used in the local query evaluations on the mobile device to predict the engine power needed for the next route segments and to generate recommendations based on the user's profile respectively.

The cache is fetched at the beginning of a training with an appropriate proactive caching strategy to cover most of the useful objects for future recommendations. As the route information is fixed in the rehabilitation scenario, this makes prefetching of data easy. The user can deviate from the initial route, and objects have to be fetched on-the-go with a more traditional cache replacement strategy. On the one hand, it is planned to utilise semantic knowledge about route characteristics and driving direction for a good forecast, but on the other hand also collaborative filtering methods (e.g. the preferred items of others) and user profile information for the caching strategy.

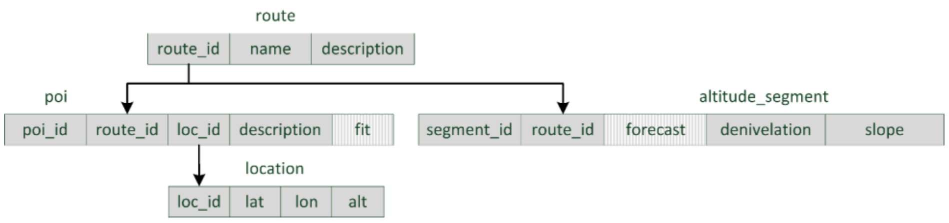


Figure 3: Cache group for routes, striped attributes are personalised and pre-calculated by the server

6 Evaluation

6.1 Cache Metrics

Traditionally, caching strategies are evaluated by a variety of performance metrics like response delay, cache size, cache hit ratio, false miss rate or caching efficiency [16], [10].

The cache retrieval and replacement strategies developed specifically for the MENTORbike scenario need a thorough evaluation. This can either happen by simulation or experimental tests. For a robust and reproducible test setup a set of tracked route recordings will be used to simulate real system usage. Together with a set of point of interests this allows an evaluation of the proposed cache replacement strategy for POIs.

The *cache hit ratio (CHR)* is defined as the number of requests retrieving a value from the cache successfully divided by the total number of requests, which yields in the percentage of requests hitting the cache. A higher value denotes a better result.

$$CHR = \frac{n_{hit}}{n_{total}} \qquad (3)$$

The *cache size (CS)* definition is straightforward. A lower value denotes a better result. Caching evaluations always need the system to be “warmed up”, i.e. they should not be performed after a cold start. Usually the system is assumed to be stabilised after 5-10 minutes of continuous usage, or - in case of cache replacement evaluations - when the cache actually is filled, so that items are being replaced.

6.2 Setup and Results

As described in Section 3, an evaluation was carried out by simulating route runs with a pre-recorded route track (GPX). In the course of the evaluation, various combinations of cache prefetching strategies, cache replacement strategies and recommendation strategies have been evaluated in regard to their cache miss rates. The simulation moves the actor's position in discrete steps, and retrieves recommendations on each step. Here the miss rate is calculated ($1 - CHR$) and then the recommended items are put into the cache, while the replacement strategy choses suitable candidates for eviction. When the simulation has finished, i.e. the actor has moved through the complete route, a weighted mean average of the miss rates at all visited locations, weighted by the respective cache sizes, is calculated. The route length is roughly one kilometre (1008 meters) and the layout follows a simple circle track over a university campus. The data base offers 75 POIs, of which 27 are in the direct vicinity (distance <1 km) of the test route. Due to the small number of available POIs, the cache size has been limited to 10 entries. The recommendation strategy recommends five POIs per request. The actor's interests have been configured to match some of the POIs, containing “relax”, “beverage” and “public transit”. The following strategies have been evaluated:

A recommendation strategy gives the user a top-N list of best-recommended objects. The *Simple POI Recommender* just recommends the nearest POIs to the current user's location. The *Context-Aware POI Recommender* issues recommendations based on location and user preferences.

A prefetching strategy is responsible to initially fill the cache with meaningful objects. The *Simple Location-based Prefetching* (Simple Prefetching) fills the initial cache with POIs near the start location, the *Extended Location-based Prefetching* (Extended Prefetching) uses all route checkpoints. The *Context-Aware Prefetching* (C-A Prefetching) tries to make better assumptions about the POIs needed by the user by additionally incorporating the user's interests and POI availability. A replacement or eviction strategy decides which cache entry will be evicted if the cache capacity is reached and a new entry is added. The *Simple Location-based Replacement* (C-AC/SL) just evicts the POI with the biggest distance to the current location. The *Context-Aware Replacement* (C-AC/C-A) uses user interests and date. Additionally, scenarios with a simple *LRU-strategy* are evaluated for comparison. In this case always the oldest cache entry is evicted.

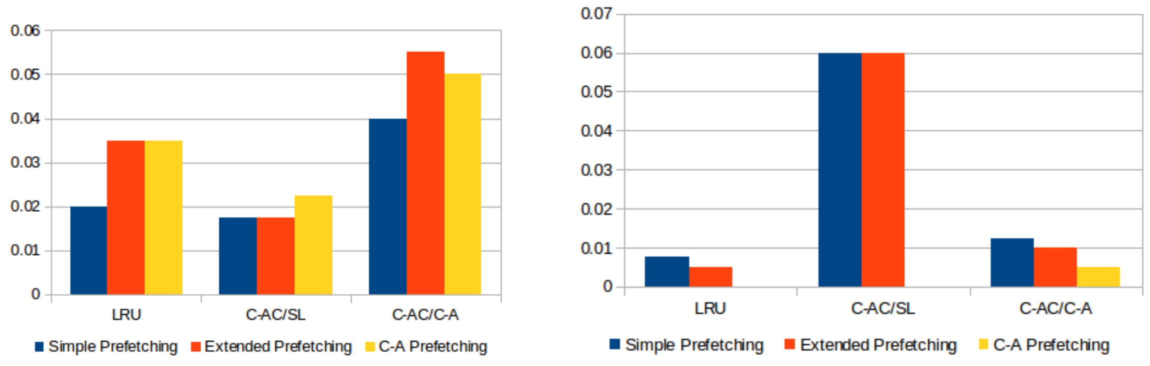


Figure 4: Cache miss rate: (a) simple POI recommender, (b) context-aware POI recommender

Fig. 4(a) shows the results for a simple recommendation strategy only incorporating the user's location. In this case, simple caching strategies like LRU or a simple location-based replacement yield in good results. Once recommendations incorporate more contextual information, more sophisticated cache strategies are necessary. The results for a context-aware recommender shown in Fig. 4(b) indicate that a simple location-based replacement strategy is not sufficient anymore. The results of LRU and context-aware caching strategies are both very good, although the route layout and the comparatively small number of POIs are beneficial for a LRU strategy - in a non-circle track LRU is assumed to perform significantly worse. Also noteworthy is the good performance of a context-aware prefetching strategy, which benefits from the context-aware recommendation strategy used for this evaluation.

The evaluation has shown that a mobile recommender system profits from caching strategies which use contextual information such as location and user interests to pre-fetch and replace their contents.

7 Conclusion & Outlook

In this paper a novel conceptual approach for a disconnected mobile recommendation engine has been presented. An investigation of existing work in the areas of mobile caching, mobile recommendation and disconnected query evaluation has led to requirements for a reliable, connectivity-independent mobile recommender system. They include an adaptable client-server architecture and an approach to pre-evaluate recommendation queries partly on the server. Furthermore, the review has shown that there are many approaches for efficient mobile caching, and some innovative ideas regarding partial query evaluations. But it also has revealed that no sophisticated system for dependable offline evaluations exists.

Subsequently, based on the requirements derived from related work, such a system was designed conceptionally and a use case scenario has been described. This use case is MENTORbike, where vital data coming from a wireless sensor network needs to be evaluated on-the-fly, even without a connection to the server. Suitable criteria for evaluating the caching strategy have been identified and described in detail. Based on a first prototypical implementation, the evaluation criteria presented in this paper have been applied in a laboratory test environment. This has shown that mobile recommender systems can benefit from context-aware caching strategies. However, the presented approach has some limitations.

Similarly to recommender systems, the proposed cache replacement strategy bears a cold start problem since no suitable data is available for precise predictions in the beginning. Keeping in mind the learning phase needed to adapt, a default fall-back replacement strategy should be implemented to cope with the cold start problem.

In the future, a complete implementation of the conceptual design will be realised for the Android mobile platform. Further evaluation will be conducted in form of a field test with real patients in an outdoor scenario with limited internet availability. The focus of this evaluation will concern cache prefetching strategies without cache replacement on the go, taking more contextual information, like predicted and actual sensor data, into account. To improve the caching strategy, future work could investigate a cache replacement strategy based on mobile intention recognition with geo-spatial data like proposed in [17]. Our approach is not restricted to the presented use case, but can be applied to a wide range of generic, mobile applications. With the presented work an important and generic step towards robust offline recommendation systems has been taken, with application scenarios not being restricted to mere training use cases. Also other use cases can benefit from a permanently available recommendation system, for example touristic applications for places where mobile data roaming is not feasible.

8 Acknowledgements

The research in this paper has been funded by the German Federal Ministry of Education and Research (BMBF) within the project MENTORbike (grant ID: 1IS11034).

9 Literature

Journal Articles

[1] E. Chan, Y. Wang, W. Li, and S. Lu, “Movement prediction based cooperative caching for location dependent information service in mobile ad hoc networks,” *J. Supercomput.*, vol. 59, no. 1, pp. 297–322, Apr. 2010.

[2] A. Emrich, A. Theobalt, D. Werth, and P. Loos, “Motivation-oriented Mobile Training: A Novel Concept for Rehabilitation and Personal Fitness,” *Int. J. Eng. Innov. Technol.*, vol. 3, no. 1, 2013.

[3] T. Härder and A. Bühmann, “Value complete, column complete, predicate complete,” *VLDB J.*, vol. 17, no. 4, pp. 805–826, Jan. 2007.

[4] A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design science in information systems research,” *MIS Q.*, vol. 28, no. 1, pp. 75–105, 2004.

[5] K. Y. Lai, Z. Tari, and P. Bertok, “An analytical study of cache invalidation algorithms in mobile environments,” *Int. J. Pervasive Comput. Commun.*, vol. 2, no. 1, pp. 3–14, 2006.

[6] K. W.-T. Leung, D. L. Lee, and W.-C. Lee, “PMSE: A Personalized Mobile Search Engine,” *Knowl. Data Eng.*, vol. 25, no. 4, pp. 820 – 834, 2012.

[7] E. J. O’Neil, P. E. O’Neil, and G. Weikum, “The LRU-K page replacement algorithm for database disk buffering,” *ACM SIGMOD Rec.*, vol. 22, no. 2, pp. 297–306, Jun. 1993.

[8] J. Xu, X. Tang, and D. L. Lee, “Performance analysis of location-dependent cache invalidation schemes for mobile environments,” *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 2, pp. 474–488, Mar. 2003.

[9] L. Yin, G. Cao, and Y. Cai, “A generalized target-driven cache replacement policy for mobile environments,” *J. Parallel Distrib. Comput.*, vol. 65, no. 5, pp. 583–594, May 2005.

[10] B. Zheng, J. Xu, and D. L. Lee, “Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments,” *IEEE Trans. Comput.*, vol. 51, no. 10, pp. 1141–1153, 2002.

Conference Articles

[11] A. Bühmann and T. Härder, “Making the Most of Cache Groups,” in *12th International Conference on Database Systems for Advanced Applications*, 2007, pp. 349–360.

[12] B. Y. Chan, A. Si, and H. V. Leong, “Cache management for mobile databases: design and evaluation,” in *14th International Conference on Data Engineering*, 1998, pp. 54–63.

[13] J. H. P. Chim, M. Green, R. W. H. Lau, H. V. Leong, and A. Si, “On Caching and Prefetching of Virtual Objects in Distributed Virtual Environments,” in *Proc. of the sixth ACM international conference on Multimedia*, 1998, pp. 171–180.

[14] A. Emrich, A. Chapko, and D. Werth, “Context-aware Recommendations on Mobile Services: The m:Ciudad Approach,” in *Smart Sensing and Context. European Conference on Smart Sensing and Context (EuroSSC-2009)*, 2009, pp. 107–120.

[15] A. Emrich, A. Theobalt, F. Leonhardt, S. Knoch, D. Werth, and P. Loos, “A Pervasive Mobile Assistance System for Health and Fitness Scenarios,” in *47th Hawaii International Conference on System Sciences (HICSS-47)*, 2014.

[16] H. Hu, J. Xu, W. S. Wong, B. Zheng, D. L. Lee, and W.-C. Lee, “Proactive Caching for Spatial Queries in Mobile Environments,” in *21st International Conference on Data Engineering (ICDE’05)*, 2005, pp. 403–414.

[17] P. Kiefer and K. Stein, “A Framework for Mobile Intention Recognition in Spatially Structured Environments,” in *Proc. of the 2nd Workshop on Behavior Monitoring and Interpretation*, 2008, pp. 28–41.

[18] K. Y. Lai, Z. Tari, and P. Bertok, “Cost efficient broadcast based cache invalidation for mobile environments,” in *Proc. of the 2003 ACM Symposium On Applied Computing*, 2003, pp. 871–877.

[19] K. Y. Lai, Z. Tari, and P. Bertok, “Location-aware cache replacement for mobile environments,” *IEEE Glob. Telecommun. Conf. 2004. GLOBECOM ’04.*, vol. 6, pp. 3441–3447, 2004.

[20] G. Pelosi and G. Psaila, “SMaC: Spatial Map Caching Technique for Mobile Devices,” in *Proc. of the 2010 ACM Symposium on Applied Computing*, 2010, pp. 1829–1830.

Online Articles

[21] Zephyr Technology, “Zephyr BioHarness 3 Datasheet,” 2012. [Online]. Available: http://www.zephyranywhere.com/media/pdf/BioHarness_3_User_Manual-FCC2012-SEP-12.pdf.