Recommendation Systems and Privacy

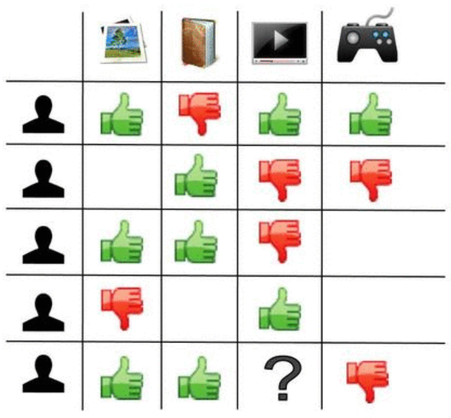
Stéphane Henriot
ENS Cachan

Recommendation Systems

During the last few years, the volume of available data (on the Internet, for instance) has become huge. Indeed, that makes it impossible for a user (no matter what is the domain) to read the whole available content. That is the reason why, in many domains (for instance shopping, news or multimedia content) recommendation systems (or recommenders) enable the user to find the content he might be interested in.

Collaborative Filtering

The idea behind collaborative filtering recommendation systems is that we can dynamically use the opinions of other users to help recommending content. Indeed, if the opinions of a subset of users are very close for content they share, we can use their opinion on other items to predict opinions and recommend items (assuming the opinions will also be close). However, that raises issues such as complexity (scalability) and privacy (communicating opinions).

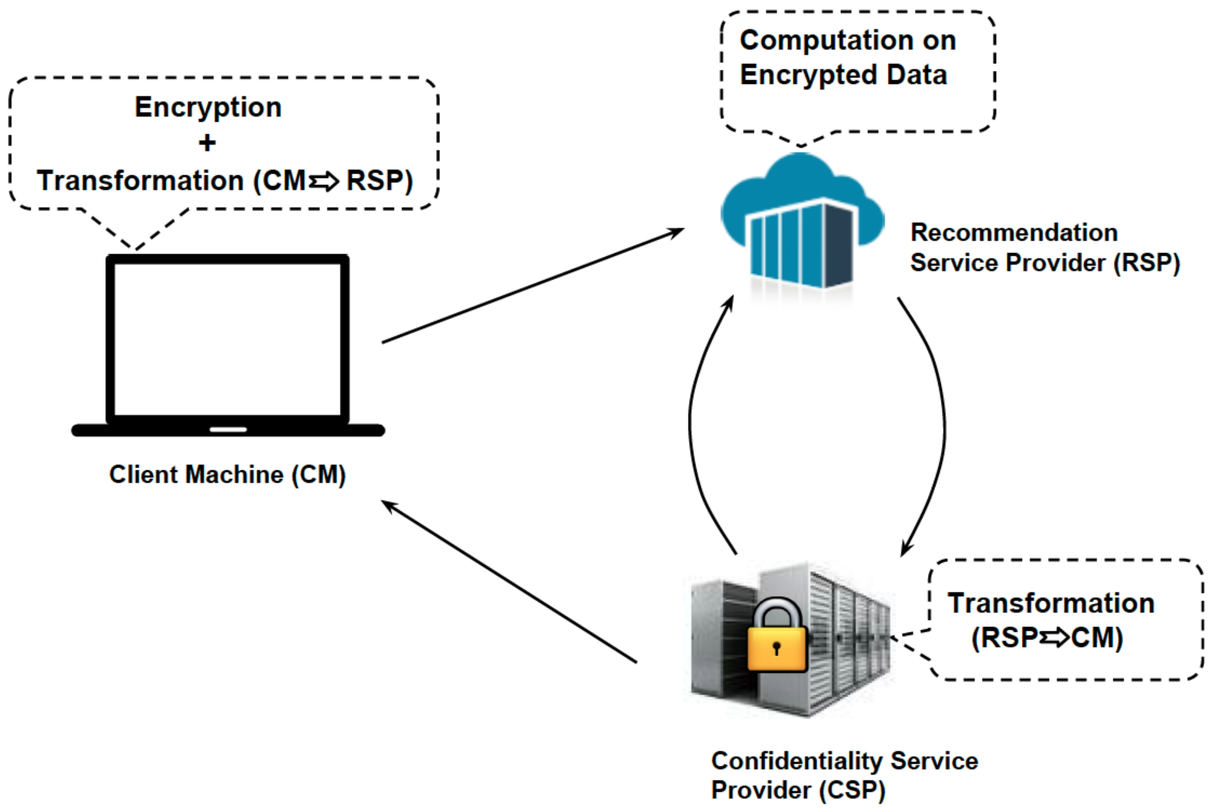


"Collaborative filtering" by Moshanin - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons

CryptRec

Most of the current recommendation systems seem to introduce privacy "Big Brother" issues, since all the users' information has to be given to one entity. Distributed systems tend to solve these issues, however privacy concerns remain, since the information can still be retrieved by spying on the network.

CryptRec is a collaborative filtering recommendation system currently being developped at the EPFL, by Patra Rhicheek and others. The main idea is to use cryptography in order to compute personalized recommendation and ensure confidentiality (reveal very little information about the users). The idea for the users is to send encrypted data. However, a given user must be able to decrypt only its own recommendation. Then, one could try to give each user a different key, but it would be hard for the recommendation system to compute recommendation based on data encrypted with different keys. That's why a transformation ensures everything is encrypted with the same key.

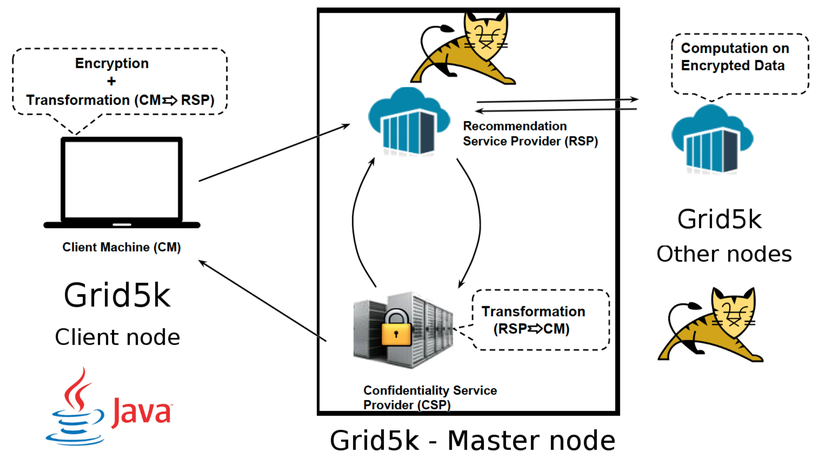


CryptRec basic scheme

Improving CryptRec

CryptRec's initial implementation has been done on a single server. Only one client can use the system at a time and the security parameter is quite low. The main challenge would be to make CryptRec a distributed system, in order to improve the scalability (latency) and security.

I rewrote parts of the existing implementation in order to make it more generic and distributed. A cluster such as Grid5k can be used to host the system, each node acting as a web server.



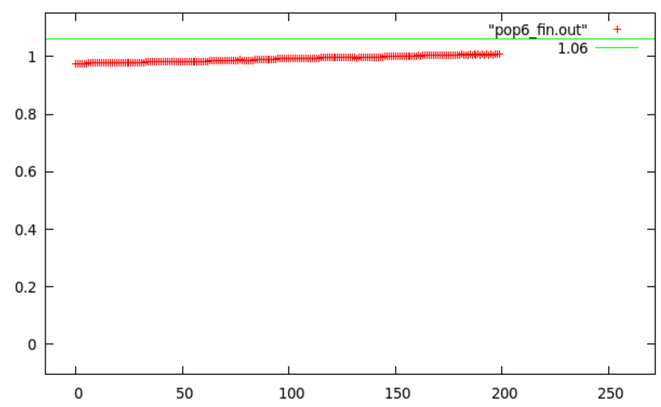
CryptRec Grid5k distributed implementation

However, the system uses too much memory to run on most machines. One can find a decomposition and use a MapReduce scheme as provided by Hadoop, but each round of communication increases the recommendation latency. There is a trade-off between memory, latency and security.

HyRec

Hybrid recommenders such as HyRec try to provide correct recommendations very quickly and through a very simple system. In hybrid systems, the server and client share the information and computing cost. For instance in HyRec, the server gives the client the last neighbors and neighbors of neighbors so that the client itself can compute the K nearest neighbors (KNN) and its own recommendation. The experimentation shows that thanks to that iterative process, the KNN sets computed quickly converge to the global KNN sets. However, there are several obstacles that can prevent HyRec to handle billions of items and users. For instance, for each request, the server has to send the full profiles of $O(K^2)$ users. The client then has to go over the whole profiles to find the popular items. Computing the similarities also includes floating point operations that can be costly. The challenge here is to improve the scalability while keeping a good recommendation quality.

We come with a few ideas that could make the system faster. First of all, the data can be updated with offline computations, from time to time. Then, a relevant (smaller) set of items can be used to partition the users (most popular or controversed items, for instance). A way to share the computing cost with the user could be to partition the users in a tree maneer an then let the client find his path. I implemented and tried this on the MovieLens 100k dataset. Each item is mapped to the most similar item to improve the density (in the top x popular items with respect to the cosine similarity) and the ratings are replaced with the average for each popular item.



The Mean Absolute Error of the new method (red curve) is close to the full KNN method (value shown in green), even with low values of x.

About this project

This project has been conducted as an ARPE (abroad year) from November 2014 to July 2015 in Switzerland. It took place in the Distributed Programming Laboratory (LPD) from the EPFL (École Polytechnique Fédérale de Lausanne). I worked with Rhicheek Patra, under the supervision of Prof. Rachid Guerraoui, LPD director.