Cost Model for Near Real-time Recommender Systems

József Marton, Zsolt T. Kardkovács, Péter Éberhardt

Database Research Labs
Department of Telecommunication and Mediainformatics
Budapest University of Technology and Economics
Magyar tudósok krt. 2, H-1117 Budapest, Hungary
{marton, kardkovacs, eberhardt}@db.bme.hu

Abstract: In this paper, we propose computational foundations of near real-time recommender systems by discussing how computational resources and limits on latency and response time requirements meet in a recommender system. A range of online applications use recommendation engines to help customers find what they need or are interested in, e.g. web shops, news services, and even computer assisted project planning. Such applications focus on slowly changing needs, and give at least locally time-invariant, non-concurrent (or at least non-correlated), or even offline calculated recommendation; hence, they need no real-time modelling of customers' behaviour. Nevertheless, well-known recommendation engines fail whenever large amount of rapid and continuous changes are made over time, e.g. in handling economic events.
While offline recommendation engines are widely discussed, characterization of cost components, and modelling of real-time recommendations are missing from the literature. In this paper, we discuss end-to-end data flow, i.e. how to transform and flow data from the source system through a near real-time data warehouse to end in a recommender system. We identify cost components of continuous flow execution in a multidimensional cost space. We determine criteria on how much computational resources are needed based on input data size, latency, and response time requirements. We also discuss the limits of latency and response time requirements given the computational resource constraints.

Keywords: cost model, near real-time recommender system

1 Introduction

A wide range of on-line applications utilize recommender systems to orient users to the resources they need or they might be interested in. Such applications might adjust the model based on continuously acquired data, to follow changes in client needs. Other applications might utilize static models created by experts on startup and adjusted in some off-line fashion. These applications traditionally have the

common base that, if present, only a small amount of (or even no) data flows in the model adjuster process. As data the model bases upon change slowly, the computational capacity needed to adjust the model is relatively low.

In contrast, when significant amount of data flows in continuously and model needs on-line updating, required computational capacity matters. To allow sizing of resources in recommender systems for operations of high computational complexity, we need to model the end-to-end flow of data including transformation that take place.

In case of near real-time systems sizing is more complicated as it is not enough to consider long-term average: peak load need also be handled in a specific time-frame in order information not to loose from its value. This puts more stress on the need to model the flow and give its cost components.

In this paper we characterize an end-to-end data flow of recommender systems in general. We also identify points in the flow where filtering, preprocessing, i.e. data quality assurance, transformation and model adjustation take place. Based on the model we identify cost components and relation among them to allow for near real-time processing.

The rest of this paper is organized as follows. Section 2 describes the related work and Section 3 gives generalized flow of data in a recommender system through a data warehouse. Section 4 identifies cost components and gives criteria for near real-timeness. Section 5 concludes and gives directions for future works.

2 Related Work

Recommender systems and thus related work ranges from industrial applications to research laboratories. Examples for industrial applications include on-line stores, e.g. e-Bay[1] offering top and related auctions, news portals, e.g. Yahoo News[2] featuring top stories and related searches, and community portals with recommender services based on social networks.

On the research labs' side, one can find various current topics. Web sites often have a deep structure, but users prefer having short path to reach areas they might be interested in. Context-based hotlink assignment methods enable flattening of the structure [1] thus allowing users to reach expected information with less transitions.

1 http://www.ebay.de/
2 http://news.yahoo.com/

Quality of recommender systems can be enhanced by utilizing information from social networks. Victor et. al. proposes using trust networks [3] for recommender systems. As trust is not a binary concept, they use fuzzy sets and this concept is shown to be beneficial in term of quality and even quantity of recommendation.

The so called cold-start problem occurs, whenever new users join the system and they need to be provided with personal recommendations. Besides recommendation inferred from trust-networks and preferences of users they trust [3], one other approach could be based on preferences of the global user community or some cluster of them. One other promising approach could be based on feed-forwarding of data on current activities of the selected user group (i.e. global or the appropriate cluster).

There's one other aspect of the cold-start problem. In case that brand new items appear in the system how do we determine to whom to recommend it. Feature-extraction based approach is proposed for music recommendation [2] to overcome this problem as tagging might be expensive, cause latency before particular item is available for recommendation etc.

Recommender systems can also be utilized in various other areas, e.g. in computer-assisted project planning. Yang et. al. proposes a revised case-based reasoning algorithm [4] to assist project managers planning their projects. Basically it consists of two steps. First, similar cases are retrieved from the knowledge base, to allow for data-mining in the second step and improve reasoning upon the retrieved cases.

3 Flow of Data

In this section we give in details the generalized data flow from source systems to end in the recommendation delivered to clients. We use an approach where data, the recommender model works on, reside in a data warehouse in some structure optimized for analytics. Data paths in the recommender system are shown on Figure 1.

Data route from source systems to clients targeted by the recommender system consists of three main stages. The transformation engine converts raw source data to a form that allow loading in the *DW* data warehouse. The in-data-warehouse analytics runs the model and streams output to the *Rep* recommendation repository where client requests $(Client_x : x \in \{1 \ldots N\})$ are served from. Details follow in subsections.

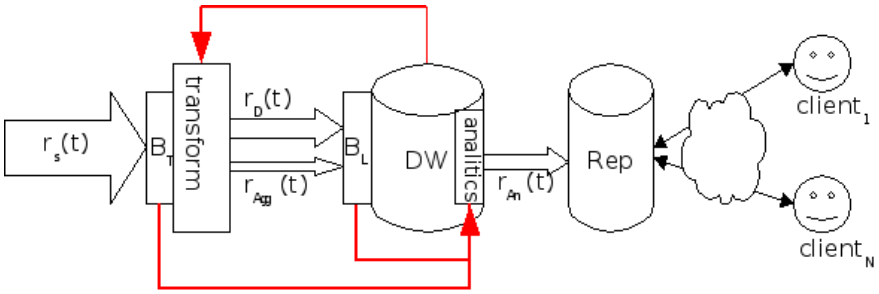

Figure 1
Data flow paths in a recommender system utilizing a data warehouse

3.1 Transformation Engine

Records from source systems S stream in the transformation buffer B_T. Buffer is utilised here in order to enable batch transformation for higher efficiency. Main role of the transformation system is to convert (i.e. preprocess) raw source records to some form optimized for analytics. Transformation engine is fed with control and auxiliary data from data warehouse that drive transformation.

Transformation engine in general emits two output sets: the detail records for analytics and aggregated records. These two record sets are denoted by D and Agg, respectively.

3.2 Data Warehouse and Analytics

The second, an inevitably the most important stage in a recommender system backed by a data warehouse is the data warehouse itself with the analytics inside.

The data warehouse component can be seen as a composition of the detail and aggregated data store fed by the transformation engine as we have seen above. Aggregated data store contains pre-calculated data to offload repeated computational tasks from the analytics component, while detail data store contains elementary records observed by the source system.

Preprocessed (detail and aggregated) data arrive in the load buffer B_L to allow for batch – hence more effective – loading into the data stores. Once data is loaded running the analytics is the next step.

The in-data-warehouse analytics has four inputs besides state it maintains in case of stateful models. Detail and aggregated data store are the main inputs of the model. Additionally, particular subset of records from the data warehouse load buffer and the transformation buffer might be fed forward. Feeding forward the contents of the mentioned buffers adds more complexity to the system as two

more source formats need to be handled by the recommender model. On the other hand it gives the system the possibility to react on critical events as soon as possible, thus partially decreasing its latency.

Output of the analytics (denoted by *An*) is transferred to the recommendation repository.

3.3 Recommendation Repository

The last but not least stage of data flow in the recommender system is the recommendation repository. Clients of the recommender system connect here. It is a fairly technical stage that off-loads end-user servicing from the data warehouse side thus might free valuable resources. Introducing this layer also allows us to isolate previous components from the outside world, i.e. isolate from clients. Should sensitive (e.g. personal or business) data reside in the data warehouse, this additional layer comes also handy for security audits.

We have seen that the output of analytics is stored here. This output might be of two forms. The first and obvious form of analytics' output is pre-calculated recommendation. In this case all answers for queries that may arise during some time frame are stored in the repository and simply returned upon query.

One other option is to store the recommendation as a function $R : C \to Re$, where *C* is the context-set the recommender takes into account while calculating its output: the recommendarion $re \in Re$.

4 Cost Model for Data Flow

Building on the generalized data flow in recommender systems in this section we give cost components that arise throughout the system.

4.1 Notations and Basic Characteristics

Notations to describe cost components of a recommender system are given in details in Table 1.

Table 1
Notations used in cost equations for data flow in recommender systems

Notation	Description
T	Time periodicity the system operates at
N	Maximum concurrent clients
s_S	Source record size
s_D	Transformed detail record size
s_{Agg}	Transformed aggregated record size
$r_{S(t)}$	Source record rate as a function of time
$r_{D(t)}$	Detail record rate as a function of time
$r_{Agg}(t)$	Aggregated record rate as a function of time
$r_{Tr}(t)$	Transformed record rate as a function of time
$r_{An}(t)$	Record rate of Analytics result
B_T	Transformation buffer, or its size[3]
B_L	Data warehouse load buffer, or its size[4]
$c_{Tr}(k)$	Transformation cost of k records
$a(M, H, F)$	Stateful analytics with M memory, H historical data and F forward-fed data
c_a	Cost function of analitics. Details given in Section 4.1
sizeof(.) *sf(.)*	The size of the indicated data set. Abbreviated as *sf*.
$(v)_X$	Value of component X of vector v.

As a rule of thumb size and rate-ralated notations can be summarized as follows. Record size and record rate as a function of time t for some given x record set as seen in Section 3 is denoted by s_x and $r_x(t)$, respectively.

The cost function of analytics is the $d \in N^+$ dimensional vector-valued function in case of d resources (i.e. d-dimensional cost space):

$$c_{a(M,H,F)}\left(sizeof\left(M\right),sizeof\left(H\right),sizeof\left(F\right)\right): R^3 \rightarrow R^d$$

[3] Context determines this choice.
[4] Context determines this choice.

The *(CPU time, Memory)* 2-dimensional cost space arises as a natural example.

Before proceeding to more complex cost formula, we examine basic properties of the amounts defined above.

To allow for data processing without loss of input data, buffer overflow is to be avoided on transformation input and data warehouse load.

That is, if the system operates with a time periodicity of *T*, following are required to be held.

$$\forall t : \int_{t}^{t+T} r_S(t) s_S dt \quad \leq \quad B_T \tag{1}$$

$$\forall t : \int_{t}^{t+T} r_D(t) s_D dt + \int_{t}^{t+T} r_{Agg}(t) s_{Agg} dt \quad \leq \quad B_L \tag{2}$$

Let's move to the transformation cost function $c_{Tr}(k)$. An optimal transformation method's cost function satisfies the following order-of-magnitude estimation using the big-*O* notation.

$$c_{Tr}(k) = O(k \cdot c_{Tr}(1)) \tag{3}$$

Should the transformation cost function not satisfy (3), performance gain could be achieved using one-by-one transformation of input records, which would contradict the optimality of the transformation method. Please note that transformation cost function is a scalar function expressing CPU time required for the transformation.

4.2 Cost of the Flow

In order to express an upper estimation of CPU time cost of a flow iteration on records gathered during the time period *[t, t+T)*, we introduce some utility functions.

The amount of fresh data gathered during the time period *[t+T, t+2T)*, i.e. during processing of previous batch is expressed as in (7). Components summarized stand respectively for data gathered in the transformation buffer, detail and aggregated records in the load buffer.

$$F_{T2} = \int_{t+T}^{t+2T} r_S(t) s_S \; dt \tag{4}$$

$$F_{D2} = \int_{t+T}^{t+2T} r_D(t) s_D \, dt \tag{5}$$

$$F_{Agg2} = \int_{t+T}^{t+2T} r_{Agg}(t) s_{Agg} \, dt \tag{6}$$

$$sf(F_2) = sizeof(F_2) = F_{T2} + F_{D2} + F_{Agg2} \tag{7}$$

Computer storage is bounded. We may thus assume that amount of historical data is bounded. Let's denote this bound by $sf(H_B)$. From now on we will use this upper bound in place of *sizeof(H)*. As long as we does not mix inequality directions (i.e. we don't mix $\leq$ and $\geq$), this would not mislead us and will definitely simplify the expressions inferred. Having the size of fresh data expressed in (7), we can now express CPU time cost of the analytics running for data gathered during time period *[t, t+T)* as given in (9).

$$c_{An1} = c_{a(M,H,F)}(sf(M), sf(H_B), sf(F_2)) \tag{8}$$

$$c_{An1CPU} = (c_{An1})_{CPUtime} \tag{9}$$

$$c_{Tr1} = c_{Tr}\left(\int_{t}^{t+T} r_S(t) \, dt \right) \tag{10}$$

Equation (10) gives us the transformation cost of records gathered during time period *[t, t+T)*. Now we have all the components to express in (11) the whole CPU cost of one iteration starting at time point *t* in the system. This very formal result simply says that CPU cost of the system for a given time period is expressed as the sum of the cost of transformation and cost of analysis.

$$c(t,T) = c_{Tr1} + c_{An1CPU} \tag{11}$$

4.3 Criteria for Near Real-timeness

We say that a recommender system modelled in the way given above is near real-time if

- response time of the system stays under a given limit most of the time, and
- processing of data does take shorter than the time interval during it was produced.

The second criterion does not allow unprocessed records to accumulate and lead to performance degradation, while the first criterion expresses clients' need for rapid responsiveness of the system.

Rapid responsiveness solely depends on the performance of the recommendation repository. Let us *L* denote the limit of response time clients find adequate. In this case, servicing the request may take at most *L/N* long on the repository side to ensure response even in the event of peak load.

Until now we used the assumption that this recommender system operates having a time periodicity of *T*. As this does not contradict near real-timeness, we can retain this assumption.

Using (11) and what we got just above, the criteria set for the data warehouse-based recommender system to operate in a near real-time fashion can be formulated as follows.

$$\forall t : c(t,T) \leq T \tag{12}$$

$$\forall t : c_{Tr1} + c_{An1CPU} \leq T \tag{13}$$

Criterion (13) also holds when $T \to 0^+$ as long as (1) and (2) (as seen in Section 4.1) holds for all *t*.

Conclusion and Future Work

In this paper we gave components of a generalized recommender system that is backed by a near real-time data warehouse and operates in-data-warehouse model. We also introduced use of feed-forwarding of subset of data to the analytics to allow reacting faster on particular behaviour changes on the clients' side.

We introduced notations and inferred the CPU cost equation in (11) in terms of input data and recommender algorithm characteristics (i.e. transformation and analytics methods). Following the way we generalized the cost equation formula for near real-time recommender systems. We also showed that the formula retains for smaller operational period of *T* as long as buffer overflow does not occur.

An interesting area for future work is to extend our cost model to utilize stochastical properties of functions describing record rates.

One other approach could be to investigate, how modification of requirements affect our model. One modification could be to require (12) to hold only at certain probability. This might lead to virtual increasement in computational capacity the recommender system has at its disposal.

References

[1] D. Antoniou, J. Garofalakisa, C. Makrisa, Y. Panagisa, E. Sakkopoulosa. Context-similarity-based Hotlinks Assignment: Model, Metrics and

Algorithm. Data and Knowledge Engineering, 2009, Accepted for future publication

[2] C.-C. Lu, V. S. Tseng. A novel Method for Personalized Music Recommendation. Expert Syst. Appl., 36(6):10035-10044, 2009

[3] P. Victor, C. Cornelis, M. De Cock, P. Pinheiro da Silva. Gradual Trust and Distrust in Recommender Systems. Fuzzy Sets Syst., 160(10):1367-1382, 2009

[4] H.-L. Yang, C.-S. Wang. Recommender System for Software Project Planning One Application Os Revised Cbr Algorithm. Expert Syst. Appl. 36(5):8938-8945, 2009