**A proposed recommendation server**

Loc Nguyen

Huong Duong Company, Ho Chi Minh city, Vietnam

**Abstract**

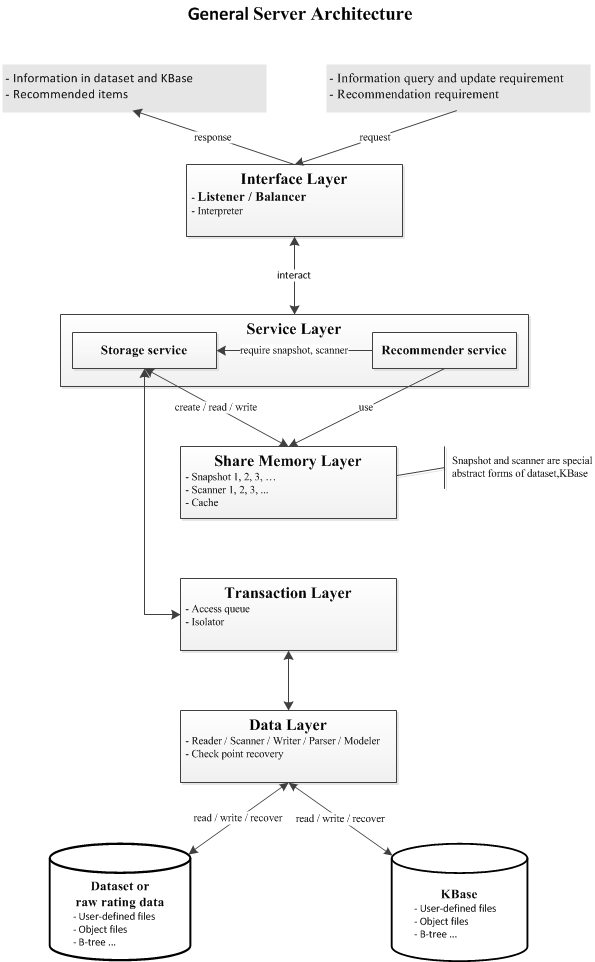
**1. Introduction to proposed recommendation server**

Recommendation server has two main goals:

* First goal is to give a list of favorite items, which responses to recommendation request from client. Note that client includes user and computer application. Both response time and quality of recommended items are optimized so that response time is minimum and quality of recommended items is maximum. This quality is measured as the level of user preference towards recommended items.
* Second goal is allowing users to retrieve and update information available in recommendation database (rating database).

**2. General architecture**

The architecture of recommendation server is divided into five layers such as *interface layer*, *service layer, share memory layer, transaction layer* and *data layer*. Each layer contains individual sub-layer, service, model or data. Following figure represents the architecture of recommendation server. These layers is described in bottom-up order.



**Figure 1**. Architecture of recommendation server

*Data layer* is responsible for manipulating rating data organized into two formats:

* Low-level format is structured as rating matrix whose each row consists of user ratings on items. Another low-level format is dataset which consists of rating matrix and other information such as user profile, item profile and contextual information. Dataset can be considered as high or intermediate format when it is modeled or implemented as complex and independent entity. Dataset is the most popular format.
* High-level format also stores user ratings as low-level format; besides, it has internal inference mechanism which allows us to deduce new knowledge such as user interests, user purchasing pattern, and etc from raw data like rating matrix, user profile and item profile. High-level format data is called knowledge base or *KBase* in short. Knowledge base is less popular than dataset because it is only used by recommendation algorithms while dataset is exploited widely.

Because data layer processes directly read and write operators so-called data operators, upper layers needs invoking data layer to access rating data. That data operators are transparent to upper layers provides ability to modify, add and remove components inside architecture. Data layer also supports checkpoint mechanism; whenever data is crashed, data layer will perform recovery tasks based checkpoints so as to ensure data integrity. Note, checkpoint is the time point at which data is committed to be consistent. Process unit of this layer, namely read or write operator, is atomic unit over whole system. Data layer interacts directly with transaction layer via receiving and processing data operators request from transaction layer.

*Transaction layer* is responsible for managing concurrence data access. When many clients issue concurrently recommendation requests relating to a huge of data operators, a group of data operators in the same request is packed as an operator bunch considered as a transaction; thus, there are many transactions. In other words, transaction layer splits requests into data operators, which in turn groups data operators into transactions. Transaction is process unit of this layer. Transaction layer regulated transactions so as to ensure data consistency before sending data operators request down to data layer. Transaction layer connects directly to data layer and connects to service layer via storage service.

*Share memory layer* is responsible for creating snapshot and scanner according to requirement of storage service. Snapshot or scanner is defined as an image of piece of dataset or knowledge base at certain time point. This image is stored in share memory for fast access because it takes long time to access data and knowledge stored in hard disk. The difference between snapshot and scanner that snapshot copies whole piece of data into memory while scanner is merely a pointer to such data piece. Snapshot consumes much more memory but gives faster access and is more convenient. Snapshot and scanner are read-only objects because they provide only read operator. The main responsibility of this layer is to create snapshots and scanners and to discard them whenever they are no longer used. Recommendation service and storage service in service layer can retrieve information of dataset or knowledge base by accessing directly to snapshot or scanner instead of interacting with transaction layer. Hence, the ultimate goal of share memory layer is to accelerate the speed of information retrieval.

*Service layer* is the heart of architecture when it realized two goals of recommendation server: giving the list of recommended items in accordance with client request and supporting users to retrieve and update rating database. Two these goals are implemented by two respective services: recommender service and storage service. Such services are main components of service layer. Recommender service receives request in the interchangeable format such as JSON format from upper layer – interface layer and analyze this request in order to understand its content such as who requires recommendation and what her / his profile is. After that recommender service applies an effective strategy into producing a list of favorite items which are sent back to interface layer in the same interchangeable format like JSON. Recommendation strategy is defined as the co-ordination of recommendation algorithms such as collaborative filtering, content filtering, etc in accordance with the coherent process so as to achieve a best result of recommendation. In simplest form, strategy identifies to a recommendation algorithm. Recommender service is the most complex service because it implements both algorithms and strategies and applies these strategies in accordance of concrete situation. Storage service is simpler when it has two responsibilities:

* Retrieving and updating directly dataset and knowledge base by sending access request to transaction layer and receive results returned.
* Requiring share memory layer to create snapshot or scanner.

Because recommendation algorithms execute on memory and recommender service cannot access dataset and knowledge base, recommender service will require snapshot or scanner from storage service. Storage service, in succession, requests share memory layer to create snapshot or scanner and receives back a reference to such snapshot or scanner. Such reference is used by recommender service.

*Interface layer* interacts with both clients (users and application) and service layer. It is the intermediate layer having two responsibilities:

* For clients, it receives request from users and sends back response to them.
* For service layer, it parses and forwards user request to service layer and receives back result.

There are two kinds of client request corresponding to two goals of recommendation server:

* Recommendation request is that users prefer to get favorite items.
* Access request is that users required to retrieve or update dataset and knowledge base.

User-specified request is parsed into interchangeable format such as JSON because it is difficult for server to understand user-specified request in plain text format. *Interpreter*, the component of interface layer, does parsing function. When users type request as text, interpreter will parses such text into JSON object which in turn sends to service layer. The result, for example: a list of favorite items, is returned to interpreter in form of JSON object and thus, interpreter translates such JSON result into text result easy to be understood by users.

Because server supports many clients, it is more effective if deploying server on different platforms. It means that we can distribute service layer and interface layer in different sites. Site can be a personal computer, mainframe, etc. There are many scenarios of distribution, for example, many sites for service layer and one site for interface layer. Interface layer has another component – *listener* component which is responsible for supporting distributed deployment. Listener which has load balancing function is called *balancer*. For example, service layer is deployed on three sites and balancer is deployed on one site; whenever balancer receives user request, it looks up service sites and choose the site whose recommender service is least busy to require such recommender service to perform recommendation task. Load balancing improves system performance and supports a huge of clients. Note that it is possible for the case that balancer or listener is deployed on more than one site.

**3. Recommendation scenario**

The popular recommendation scenario includes five following steps in top-down order:

1. User (or client application) specifies her / his request in text format. *Interpreter* component in interface layer parses such text into JSON format request. *Listener* component in interface layer sends JSON format request to *service layer*. In distributed environment, *balancer* is responsible for choosing optimal service layer site to send JSON request.
2. *Service layer* receives JSON request from interface layer. There are two occasions:
   1. Request is to getting favorite items. In this case, request is passed to recommender service. Recommender service applies appropriate strategy into producing a list of favorite items. If snapshot or scanner necessary to recommendation algorithms is not available in *share memory layer*, recommender service requires storage service to create snapshot or scanner. After that, the list of favorite items is sent back to interface layer as *JSON format result*.
   2. Request is to retrieve or update dataset or knowledge base such as querying item profile, querying average rating on specified item, rating an item, etc. In this case, request is passed to storage service. If request is to update data then an *update request* is sent to *transaction layer*. If request is to retrieve information then *storage service* looks up *share memory layer* to find out appropriate snapshot or scanner. If such snapshot or scanner does not exists or not contains requisite information then a *retrieval request* is sent to *transaction layer*; otherwise, in found case, requisite information is extracted from found snapshot or scanner and sent back to interface layer as *JSON format result*.
3. *Transaction layer* analyzes *update requests* and *retrieval requests* from *service layer* and parses them into transactions. Each transaction is a bunch of read and write operations. All low-level operations are harmonized in terms of concurrency requirement and sent to *data layer* later. Some access concurrency algorithms can be used according to pre-defined isolation level.
4. *Data layer* processes read and write operations and sends back *raw result* to transaction layer. Raw result is the piece of information stored in dataset and knowledge base or the output variable indicating whether or not update (write) request is processed successfully. Transaction layer collects and sends back the raw result to service layer. Service layer translates raw result into *JSON format result* and sends such translated result to interface layer in succession.
5. The *interpreter* component in interface layer receives and translates *JSON format result* into text format result easily understandable for users.

**4. Conclusion**

**Một trình chủ khuyến nghị đề nghị**

Nguyễn Phước Lộc

Công ty TNHH MTV Lập trình Hướng Dương, Hồ Chí Minh city, Việt Nam