A Pluggable Architecture for Building User Models From Massive Datasets

Heath Hohwald, Enrique Frias-Martinez, and Nuria Oliver

Data Mining and User Modeling Group
Telefonica Research, Madrid, Spain
{heath,efm,nuriao}@tid.es

Abstract. In many situations, it is common that a large single source of data serves as input to multiple application areas, each of which may use a different user model. It is often the case that each user model is assembled using a different process, however, in general, it is more efficient to have a single architecture for building different user models for different application areas. We propose an architecture based on MapReduce that allows for processing terabytes of information in a timely fashion using pluggable components, each one capable of including different features in the final user model. A metamodel is used for specifying the characteristics of the desired user model, which can include a short-term user model and a long-term user model, and the architecture is responsible for building it from the specified data and pluggable components. We present an instantiation of the architecture for telecommunication (telco) applications and evaluate how the architecture escalates with a real dataset. Our evaluation indicates that complex user models for millions of users can be obtained in a timely fashion.

1 Introduction

User Modeling (UM)is an ad-hoc process in which the dimensions that define each user model are built considering the application and the data available. Nevertheless, the ad-hoc nature of the process does not imply that the architecture used to generate UMs has to be ad-hoc as well. On the contrary, the different dimensions that compose a UM can be processed, from an architectural point of view, using the same concepts. A user model is a set of information structures designed to represent one or more of the following elements [11]: (1) representation of assumptions about preferences and/or abilities about one or more types of users; (2) representation of relevant common characteristics of users (stereotypes); (3) the classification of a user in one or more of these stereotypes; and/or (4) the formation of assumptions about the user based on the interaction history. For all four cases, a user model is typically represented as a vector, where each dimension represents an element that provides valuable information for personalization and/or prediction. Although this vector is not necessarily the end product of the modeling process (for example, the result can be just a number indicating in which stereotype a user is included) at some point of the process a vectorial representation of the characteristics of each user will be used.

The construction of a vector for user modeling can be carried out using a user-guided (or adaptable) approach [7] or an automatic (or adaptive) approach [7] . The adaptive approach implies the processing of massive amounts of data using typically statistical or data mining techniques in order to construct the final model. This fact implies that UM is a computational intensive task due to (1) the number of users that have to be modeled (especially in commercial environments); (2) the complexity of the process of generating each user model and (3) the need for some applications to generate a short-term and a long-term user model. The need for an architecture that is able to efficiently process and generate UMs also arises from the application side. There are a variety of applications that require UM to be regenerated frequently and with strict time constraints (churn prediction [4], fraud detection [9], real-time recommendation, etc.) in order to capture variations in behavior. In this context, one of the key challenges in UM is the ability to generate millions of UMs from tera-bytes of information in a timely fashion.

Until now, the literature in UM has not focused much attention on architectures for UM that solve the previous limitations mainly because: (1) the data available was limited, (2) the number of users was also limited and (3) the applications presented did not have demanding time constraints. Nevertheless, with the increasing availability of individual information in a variety of environments such as the Web, telecommunications (telco), medical, etc., a platform that can generate massive and complex user models in a timely fashion is becoming increasingly needed. For example, in the telecommunication environment, generating daily user models for 20 million individuals from the information stored during several months of activity would be considered a medium sized problem.

In this paper we present an architecture for user modeling, based primarily on the MapReduce paradigm, that tackles the previous problems. The proposed architecture: (1) generates user models (independently of the data available, complexity of the model, and the number of user models to generate) in a timely fashion; (2) has the capability of generating at the same time short and long term user models (if needed) and (3) is pluggable in the sense that the process used for generating values for each dimension that comprises the user model vector can be plugged from an existing library. The proposed architecture is independent from the application environment or specific applications and can be used in a variety of typical UM environments such as web user modeling, telecommunication user modeling, student modeling, etc.

The rest of the paper is organized as follows: first we describe related work (Section 2) and after that a description of the architecture is given in Section 3. Section 4 presents an instantiation of the architecture for telco applications, and Section 5 evaluates the performance characteristics of the model using a large data set.

2 Related Work

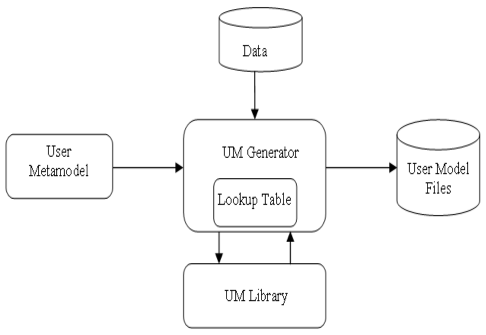
In the literature of UM the concepts of architecture for UM as referred in this paper are usually included as a part of servers for UM[5][3]. Servers for UM include two major functionalities: (1) serve as a central repository of information of a user and (2) contain the functionality to transform raw data about users into user models. We are concern with the second part of the functionality. AHA! [5] is a good example of a server for user modeling specifically designed for learning environments and for user modeling on-the-fly. It does not only provide the framework for UM but also includes mechanisms for content adaptation. CUMULATE [3] is another example of UM server that uses a distributed architecture. The work presented in this paper focusses on generating UMs from huge amounts of data in a timely fashion. It can not be considered a UM server because it is not concerned with delivering and using the UM. Nevertheless it can can be part of a generic UM server.

The most related work to our research is in the field of data mining and web mining frameworks[14], some of them also focussing on high performance [8]. Nevertheless, in general, these frameworks are not focussed on user modeling. In this context [12] presents a model called WIM (Web Information Mining) for web mining prototyping, that provides a model an an algebra to express web mining tasks. The work we present is similar in the sense that it is generic, but while [12] focuses mainly on the algebra, we focus mainly on the processing capabilities. Our architecture can also be instantiated to generate web user models as it will be shown.

Note that the approach we take with our proposed architecture is very different from updating UM on-the-fly when new information arrives, as done in AHA! [5]. Such an approach requires a much more complex data representation and has to be built using ad-hoc architectures. Also, when short term and long term models are being generated, the advantages of having on-the-fly UMs disappears as all models have to be updated when time goes by, independently of the activity of each user. In our approach, we use the advances in platforms and methods for data-intensive distributed computing processing for efficient UM generation.

3 Architecture for Terabyte-Scale User Modeling

The architecture, as presented in Figure 1a, has four main components: (1) the Data available to generate UMs; (2) a User Metamodel that describes the components of the long term (LT) and short term (ST) UMs ; (3) a UM Library that contains the most common functions used to generate UMs and (4) a terabyte scale UM generator that instantiates the architecture and generates the set of UMs from the information contained in the metamodel using the library. The output of the architecture is the set of UMs for the users selected.



(a) Architecture

```
ParFor
    ParFor $i=1 \ldots M$
        Launch($ST\text{-}UM_i$)
    EndFor
    Wait($ST\text{-}UM_i$) $i=1 \ldots M$
    Aggegate ($ST\text{-}UM_i$)

    ParFor $i=1 \ldots N$
        Launch($LT\text{-}UM_i$)
    EndFor
    Wait($LT\text{-}UM_i$) $i=1 \ldots N$
    Aggegate ($LT\text{-}UM_i$)
EnfFor
```

(b) Pseudocode

Fig. 1: Architecture and Pseudocode for the UM Generator

3.1 Data Input & Output

The data available to generate the UMs can be a single file containing all the logs or a set of files. It is very common that the logs collected are stored using some time considerations, for example one file of activities is generated for each month or each day. The output of the architecture can be presented and managed as a single file, but typically it will be a set of files along with a lookup table that indicates for each user the file that contains the associated user model. This fact is related with implementation and will be detailed in the corresponding subsection. Typically the files will be stored in a distributed file system, for example the Hadoop Distributed File System [2].

3.2 User Metamodel

The metamodel is a file that contains all the parameters that describe the UM that is to be generated. The use of a metamodel allows the architecture to factor out commonality across several application areas that use the same parameters or dimensions. It also allows for the easy use of pluggable components. The metamodel represents a very high level abstraction and is very useful for separating user-modeling code from actual model construction, thus allowing for models to be built without detailed knowledge of the data layout or the distributed progamming code underlying the architecture. The metamodel contains the following parameters:

1. *INPUT* is a pointer to the files/directory that contain the data to be processed. Typically each file will contain the activity logs for a given period of time. Each entry in the file will have an indication of a user and some activity of that user in a specific moment in time.
2. LT_s, LT_e, ST_s, and ST_e indicate the start and end times to be considered to generate the Long Term (LT) and the Short Term (ST) models. Only one

of the temporal models can be generated. For example, only a short term model is generated if $LT_s = LT_e$.

3. LT-UM is a vector $(LT - UM_1, \cdots, LT - UM_N)$ indicating the parameters that define the long term user model. Each dimension of the vector indicates some knowledge obtained from the information of that user in the data pointed by INPUT. The processes for generating these dimensions for a given context, i.e. telecommunications applications or web mining, can be, for the best part, standardized and are included in the UM Library.
4. ST-UM is a vector $(ST - UM_1, \cdots, ST - UM_M)$ indicating the parameters that define the short term user model. Note that LT-UM and ST-UM can be different and can have a different number of dimensions. Often they share many dimensions in order to measure significant recent deviations in ST-UM from LT-UM, but the short term model may include dimensions that only have significance over a short term period.
5. FILTERS indicates conditions that the user activity must meet in order to generate a corresponding user model. One common condition used for filtering is ensuring some minimum level of user activity, thus user models are only built for users that are reasonably active.
6. NEW-PARAMETER, contain one or more indications of what process to use to generate a dimension of either the LT-UM or ST-UM for the case where the library does not already contain the necessary process.

3.3 UM Library

The UM library contains the functions to calculate the dimensions specified in the LT-UM and ST-UM. Although in general each dimension will be the output of some function, it can happen that different dimensions are produced by the same function, i.e. a function can produce more that one output, and all or only part of them are needed by the specification of LT-UM or ST-UM. For a given context, the UM Library will contain the typical elements that user models use in that environment. For example in web mining there can be a function to identify the number of visits of a user to a webpage or the number of times that a specific element of the interface has been used. In a telecommunication environment a function might calculate the total talk time for a given subscriber, the total number of phone calls made or received by a subscriber, or the number of individuals in a subscriber's social network. Note that new elements can be dynamically added to the Library using the NEW-PARAMETER element of the User Metamodel.

3.4 Terabyte-Scale UM Generator

The UM Generator applies the User Metamodel to the Data and produces the User Models using the necessary elements of the UM Library as well as any NEW-PARAMETERs. The Generator matches each one of the elements of ST-UM and LT-UM to the corresponding functions of the UM Library with a lookup table. If the function is not available it must be provided by NEW-PARAMETER and will subsequently be included in the library and added to the table for future use. Figure 1b presents the pseudocode for the Generator, which launches in parallel each element of ST-UM and LT-UM, waits for all elements to finish and finally aggregates the results. Implicit is the fact that the different dimensions are independent in the sense that the value of one dimension of the model is not needed to compute the value of other. Such limitations are easily avoidable by creating functions that output more that one value at the same time. An example in the context of modeling users for telco applications would be the dimension total number of calls, which can be calculated as the number of received calls plus the value of made calls. Instead of having three functions, one for each value, it is much better to have just one function that outputs the three values, letting the UM Generator decide which one of these outputs are needed.

TODO: discuss aggregation and add aggregation figure

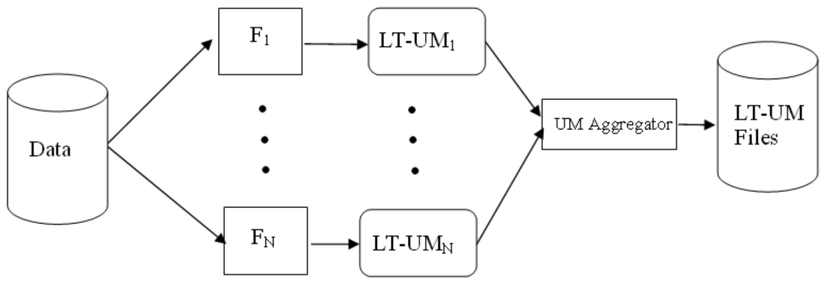


Fig. 2: Software Architecture of the Execution for LT-UM. An equivalent process is run in parallel for ST-UM.

3.5 Implementation

The architecture for user modeling we are presenting is heavily influenced by the concepts and philosophy used in the MapReduce [6] programming model. MapReduce is a framework for processing massive amounts of data in parallel using a (typically large) cluster of computers. The data can be unstructured (i.e. not a database) which applies to large quantities of data used for UM, such as server logs, telecommunication Call Detail Record (CDR) logs or search engine query logs. From the Map/Reduce perspective the UM Generator is a direct implementation of mapping elements of the vectors to machines and then reducing the outputs to construct the final user model. The architecture detailed above has been implemented using Hadoop [1], an open-source Java implementation of MapReduce, and HDFS [2], the associated distributed file system.

The functions included in the UM library will usually be implemented as MapReduce processes, although the functions can be implemented using traditional programming paradigms. Nevertheless, the choice of implementation will deeply impact the computational throughput because of the large volume of data and the need to aggregate all the results to produce the final UMs.

4 Terabyte-Scale User Modeling for Telco Applications

Telco applications focus on understanding how individuals use their cell phone, and how that knowledge can be used for providing better services. Common applications include churn prediction[13][4] or fraud detection [9], all of which focus on user activity. All these applications use the same data source CDR (Call Detail Records), and the different user models can have elements in common which can be shared using the pluggable nature of the architecture. In all these applications the behavior of the user is represented with a vector that contains a set of elements, and these vectors are used for training classification or regression algorithms. These are also examples of application areas where it is common to have as input several terabytes of data from tens of millions of users and where obtaining user models in a timely fashion is key. For example, churn or fraud detection models are usually run on a daily basis with all the available information to detect users ar risk. Note that the execution of the classification algorithms can also be run as part of our architecture.

4.1 UM Library for Telco Applications

The functions implemented in the UM Libray for Telco Applications have been selected from the wide literature in churn prediction, fraud detection, personalization and recommendation and also from our own experience. In some cases, like [10], we can find elements already developed using the MapReduce philosophy that can be directly added to the Library. Here we briefly detail some of the functions implemented:

1. CALLS(): for each user identifies the total number of phone calls made, total duration of the calls, total number of phone calls received, total duration of calls receive, percentage of calls made to other networks and percentage of non-local calls. This is an example of a function that for computational purposes produces multiple outputs. If only a subset is specified by the meta-model the UM Generator will ignore the dimensions not needed.
2. SMS(): The same as in the previous case but considering SMSs and where duration of the call is substituted by the number of bytes in the text message.
3. DEGREE(): for each user obtains the IN-DEGREE, the OUT-DEGREE, and the TOTAL-DEGREE of voice calls, also for each one of these outputs the values for individuals that use the same carrier and other carriers is used.
4. DEGREE-SMS(): the same as the previous function but considering SMSs only.
5. CALLS2NUMBER(PHONE-NUMBER): for each user it identifies the number of calls and the total calling time to a specific PHONE-NUMBER or group of numbers. This function is very useful to quantify accesses to specific services.

6. TERMINAL(): for each user it identifies how many times a new terminal has been used and the age of the current terminal.

Figure 3 presents an examples of the function XXX included in the UM Library for Telco Applications coded with hadoop. DESCRIPTION OF THE CODE. TODO: Add code, describe it

ejemplo de codigo

Fig. 3: Example of Hadoop code for implementing the XXX function.

4.2 User Metamodels

In this section we describe two examples of UMs built with the UM Library for Telco Applications: Churn Prediction and Fraud Detection. Churn is one of the main problems of any telco operator. In general the reasons for churn are multiple but there are a variety of factors that are good indicators such as the number of individuals in a subscriber's social network that use other carrier, the number of complaints the subscriber makes to the operator, or a large number of phone calls to other service providers. The UM generated by our architecture will be fed to an SVM that has already been trained to identify people at risk of churning. This feeding happens on a daily basis, thus the need to have the models of all users in a timely fashion. In this context using a ST Model and a LT Model is very useful for measuring recent changes in individual behavior and we assign ST_s and ST_e to span 5 days and LT_s and LT_e to span 30 days. ST-UM and LT-UM contain in this case the same dimensions: (1) CALL2NUMBER(CUSTOMER-SERVICE-NUMBER) to have information about the frequency of calls to the Customer Service of the Carriers; (2) CALL2NUMBER(OTHER-CUSTOMER-SERVICE-NUMBER)to have information about the frequency of call to Customer Services of other carriers; (3) TERMINAL() to have an indication of the frequency with which the user acquires new hardware; (4) DEGREE().TOTAL-DEGREE, only the TOTAL-DEGREE output of the function DEGREE is used the UM;(5) DEGREE().TOTAL-DEGREE-OTHER-CARRIERS; and (6) CALLS().NUMBER-CALLS.

CAMBIAR ESTO ... Fraud detection in the telco context aims at detecting individuals that acquire a cell phone and do not intend to pay their contract. Note that the problem focuses only on new clients. In general fraudulent subscribers are characterized by high peaks of consumption and calls to the same social circles [9]. As in the previous case, once the UM are generated they are fed into an SVM that has already been trained to identify fraudulent behaviors. As for the metamodel, it only defines a short term model of 14 days and no long term model. sT-UM is defined with the following dimensions: (1)CALLS(), with all the inputs that the function produces being stored; (2) SMS(), with all the inputs being included in the model; (3) DEGREE(), and (4) DEGREE-SMS(). The FILTER indicates that only new clients have to be considered. Again this is an application very time sensitive, and the sooner fraudulent activity is identified the better.

5 Architecture Performance Evaluation

In order to evaluate the performance and scalability of the architecture proposed in 3, a reference implementation was developed using a combination of Java and Hadoop. The performance of the architecture, measured in terms of total running time in seconds for producing user models, was evaluated in terms of two primary variables: (1) the number of CPUs available in the compute cluster, and (2) the number of dimensions in the short and long term user models.

5.1 MetaModels Used for Performance Evaluation

From the UM library, 7 different variables were chosen that exhibited similar performance characteristics (within one percent difference in running time). Using the chosen set of variables, 14 total metamodels were created, 7 consisting only of a short term user model and 7 consisting only of a long term user model. The short term metamodels, denoted as $MM_{S1}, MM_{S2}, ..., MM_{S7}$, all specified 3 days of data, and the ith short term metamodel specified a user model consisting of the first i variables (and no long term model). The long term metamodels, denoted as $MM_{L1}, MM_{L2}, ..., MM_{L7}$, all specified 30 days of data, and the ith long term metamodel specified a user model consisting of the first i variables (and no short term model).

5.2 Data Sets Employed for Testing

For the purpose of the performance evaluation, anonymized CDR data from a major mobile phone operator in a developed country was used. Data was made available as a set of files where each file corresponded to all CDRs for a given day of global activity. The 3 day short term model and the 30 day long term model thus consisted of 3 and 30 files. The CDR information present in each record included the encrypted originating phone number, the encrypted destination phone number, the duration of the call in seconds, and the time and date when the call originated. The total number of calls for the short term and long term user models was $2.18 * 10^8$ and $2.17 * 10^9$, comprising 10.1 and 103.0 gigabytes of data with a total of 300,000 users.

5.3 Cluster Hardware and Software

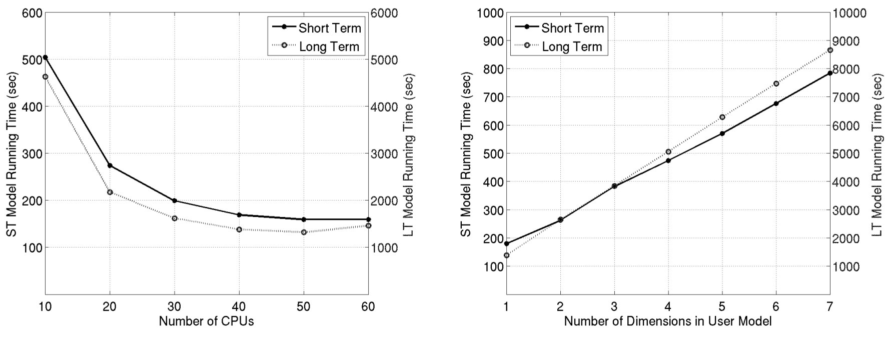
The experiments were run on a compute cluster of five nodes. Each node consisted of 16 GB of RAM, 4 hard drives with 1 Terabyte storage capacity, and 4 quad core processors. The nodes were all connected with a fast 100 GB network switch. While performing the experiments, the compute cluster had no other significant processes running.

For the results presented, version 0.20.1 of Hadoop was deployed. Changing several default settings were found to increase performance for executing a variety of metamodels. Those with the most impact on performance were: changing the maximum number of both map and reduce tasks run on each machine from 2 to 6, increasing the size of the JVM for each task from 200 MB to 1 GB, increasing the size of sort buffers from 100 MB to 500 MB, increasing the sort factor from 10 to 100, increasing io file buffer size from 4 KB to 64 KB and increasing the proportion of the total heap size used for retaining map outputs in memory during the reduce stage to 90%.

5.4 Performance Results

One of the assumptions of the proposed architecture is that it is highly scalable. In order to test this premise, different HW configurations were used in order to disable a specified numbers of available CPUs. For each configuration, both MM_{S1} and MM_{L1} were built using the specified number of CPUs. The results are presented in Figure 4a, where the x-axis reflects the number of CPUs enabled in the cluster, and the left (short term) and right (long term) y axes show the total time taken in seconds to build the short and long user models. There is a factor of 10 difference in the left and right y axes, reflecting the fact the the long term model is built from roughly 10 times as much data. In both cases, the time taken to build the user model initially decreases as more CPUs were added but eventually additional CPUs did not further decrease running time, with performance reaching a maximum at about 50 CPUs. This pseudo-exponential performance curve is in agreement with previous finding [10], and likely results from memory saturation as each CPU corresponds to a map or reduce process that may need a substantial amount of memory for sorting results. Even with only 10 available CPUs, more than 2 billion CDRs were processed and a long term user model of one dimension was built in about 1 hour and 17 minutes.

Another assumption of the proposed architecture is that as more dimensions are added to the user model, performance scales well to accommodate additional dimensions. In order to test the relationship between total running time and the number of dimensions in the user model, the cluster was configured to always have 40 CPUs, using 8 CPUs from each of the 5 machines. The 7 short term metamodels $MM_{S1}, MM_{S2}, \cdots, MM_{S7}$ and the 7 long term metamodels $MM_{L1}, MM_{L2}, \cdots, MM_{L7}$ were each fed into the architecture and the corresponding user models were built. The resulting running times are presented in Figure 4b, where the x-axis represents the number of dimensions in the user model and the left (short term) and right (long term) y axes show the total time



(a) Time vs. Number of CPUs

(b) Time vs. User Model Size

Fig. 4: Time Needed to Build Long and Short Term User Models

taken in seconds to build the short and long user models. Both short and long term models are seen to exhibit linear scalability. As in figure 4a, there is a factor of 10 difference in the left and right y axes, and the two lines closely follow one another evince a strong linear relationship. The correlation coefficient between the number of model dimensions and the short and long term running times are 0.9994 and 1.0000, providing strong evidence for linear scalability. It is worth noting that in the experiments presented, building a long term model with 7 different dimensions from a real data set with more than 2 billion records took a little more than two hours, implying that the architecture can easily support daily updates of the user models for the data sets considered.

6 Conclusions & Future Work

We have presented a pluggable architecture for building both long and short term models. The concept of a metamodel is used, allowing for the abstraction of commonality from different user model building processes. The resulting architecture can be used to quickly build different short and long term user models from massive data sets with little to no new software development. The proposed architecture is very generic and can be used for any application area that has record-based log files. The pluggable components were implemented in Java using MapReduce but the architecture can execute components written in any language.

The architecture was implemented and its performance has been tested under several scenarios. We found that increasing computer resources, as measured by the number of available CPUs in a compute cluster, decreases the running time needed for building user models to a certain point, after which more CPUs no longer decrease model build time. We also verified that the architecture can scale lineraly with the number of dimensions in the user model. The results highlighted the ability of the architecture to produce user models in a timely fashion and thus be used for applications with strong time constraints.

For future work, we would like to expand the set of components included in the UM library. We will add more components relevant to different telecom application areas as well as either writing or reusing existing components for web mining, thus allowing the architecture to support the building of long and short term user models for personalization or recommendation systems.

References

1. Hadoop information. `http://hadoop.apache.org`.
2. Hdfs architecture. `"http://hadoop.apache.org/common/docs/current/hdfs_design.html"`.
3. P. Brusilovsky, S. Sosnovsky, and O. Shcherbinina. User modeling in a distributed e-learning architecture. *Lecture notes in computer science*, 3538:387, 2005.
4. K. Dasgupta, R. Singh, B. Viswanathan, D. Chakraborty, S. Mukherjea, A. Nanavati, and A. Joshi. Social ties and their relevance to churn in mobile telecom networks. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pages 668–677. ACM, 2008.
5. P. De Bra, A. Aerts, B. Berden, B. De Lange, B. Rousseau, T. Santic, D. Smits, and N. Stash. AHA! The adaptive hypermedia architecture. In *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, page 84. ACM, 2003.
6. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. pages 137–150, 2004.
7. J. Fink, A. Kobsa, and A. Nill. Adaptable and adaptive information access for all users, including the disabled and the elderly. *COURSES AND LECTURES-INTERNATIONAL CENTRE FOR MECHANICAL SCIENCES*, pages 171–174, 1997.
8. R. Grossman and Y. Gu. Data mining using high performance data clouds: Experimental studies using sector and sphere. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 920–927. ACM, 2008.
9. S. Hill, D. K. Agarwal, R. Bell, and C. Volinsky. Building an effective representation for dynamic networks. *Journal of Computational & Graphical Statistics*, 15:584–608(25), September 2006.
10. U. Kang, C. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system - implementation and observations. *IEEE International Conference on Data Mining*, 2009.
11. A. Kobsa. Generic user modeling systems. *User modeling and user-adapted interaction*, 11(1):49–63, 2001.
12. Á. Pereira, R. Baeza-Yates, N. Ziviani, and J. Bisbal. A model for fast web mining prototyping. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 114–123. ACM, 2009.
13. C. Wei and I. Chiu. Turning telecommunications call details to churn prediction: a data mining approach. *Expert systems with applications*, 23(2):103–112, 2002.
14. C. Wood and T. Ow. WEBVIEW: an SQL extension for joining corporate data to data derived from the web. *Commun. of ACM.*