ARBUD: A Reusable 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 applications, each of which may use a different user model. It is often the case that each user model is created using a different process; however, in many cases it would more efficient to use a common architecture for building different user models in different application areas. In this paper, we propose a distributed-computing architecture based on MapReduce that allows for the efficient processing of massive datasets using reusable components that compute different features of the final user model. A metamodel is used for specifying the characteristics of the desired user model – which can include both short-term and long-term user models – and the architecture is responsible for building the user model from the specified data and reusable components. We present an instantiation of the architecture in the context of telecommunications applications and empirically evaluate the scalability of the proposed architecture with a real dataset. Our results indicate that complex user models for millions of users can be obtained in just a few hours on a small computer cluster.

1 Introduction

A user model is a set of information structures designed to contain one or more of the following elements [11]: (1) A representation of the assumptions about preferences and/or abilities of one or more types of users; (2) a 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 his/her interaction history. A user model is typically represented as a vector, where each dimension contains valuable information for personalization and/or prediction. Although this vector is not necessarily the end product of the modeling process (for example, the result may 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.

Hence, a common task in user modeling consists of defining the dimensions (or features) that define each user model, depending on the application at hand and available data. This process is often somewhat *ad hoc* in nature though this does not imply that the architecture used to generate user models has to be *ad-hoc* as well. For example, the different dimensions that compose a user model could be processed, from an architectural point of view, using the same functions.

Once the dimensions of the user model are defined, the construction of the user model vector can be carried out using a user-guided (or adaptable) or an automatic (or adaptive) approach [7]. The automatic approach implies processing massive amounts of user activity data by means of statistical or data mining techniques in order to construct the final model, such that user modeling becomes a computational intensive task due to: (1) The large number of users that have to be modeled – particularly in commercial environments; (2) the complexity of the process to generate each user model, and (3) in many cases, the need for multiple models, *e.g.* short-term and long-term user models.

An architecture that is able to efficiently process and generate user models is also needed from an application's perspective. There are a variety of applications that require user models to be regenerated frequently and with strict time constraints in order to capture variations in behavior (for example, in churn prediction [4], fraud detection [9], real-time recommendations, etc.). In this context, one of the key challenges in user modeling is the ability to generate millions of user models from potentially terabytes of data in a timely fashion (*e.g.* hours rather than days or even weeks).

Until recently, the user modeling literature has not devoted much attention to scalable architectures for efficient and large-scale user modeling, mainly because: (1) The amount of available data and the number of users were limited, and (2) the applications where user models were required did not have demanding time constraints. Nevertheless, the increasing availability of large scale human activity data in a variety of domains (*e.g.,* the Web, telecommunications, medical, etc.) creates the need for platforms that can generate massive and complex user models in a timely fashion. As an example, in the telecommunications industry, generating daily user models for three different application areas for 20 million individuals from the information stored during several months of activity would be considered a medium sized problem. The construction of such models presents numerous engineering challenges since typical software methods are not capable of handling such large data sets and are not able to meet the demanding time constraints.

In this paper, we propose a scalable architecture for user modeling, based primarily on the MapReduce paradigm [6], that: (1) Is able to efficiently generate complex user models in a timely fashion from massive amounts of data and for a large numbers of users; (2) has the capability of generating both short and long term user models (if needed), and (3) uses pluggable components where the functions that are used to compute each of the features in the user model vector can be reused from existing libraries and shared across multiple application domains. In addition, the proposed architecture is generic and hence independent from the specific application environment and can be used in a variety of typical user modeling environments such as web, student, medical, or telecommunications user modeling. Finally, we shall highlight the importance of developing scalable approaches to user modeling that are able to process an entire user base of millions of users without the need for sampling users or time periods. To the best of our knowledge, the proposed architecture is the first of its kind to address the challenges of scalability, reusability and domain independence.

The rest of the paper is organized as follows: First, we describe related work (Section 2), followed by a description of the architecture in Section 3. Section 4 presents an instantiation of the architecture in the context of telecommunications applications. Section 5 evaluates the performance of the proposed architecture on a large data set of over 20 million mobile phone users, and we conclude in Section 6.

2 Related Work

In the user modeling literature, the concept of architecture for user modeling as referred to in this paper typically consists of servers for user modeling [5, 3] that include two major functionalities: (1) Serve as a central repository of information about a user and (2) contain the functionality to transform raw data about users into user models. The work presented in this paper includes the second functionality. AHA! [5] is a good example of a server for on-the-fly user modeling specifically designed for learning environments. It does not only provide the framework for user modeling but also includes mechanisms for content adaptation. CUMULATE [3] is another example of a user modeling server that uses a distributed architecture. Note that the work presented in this paper cannot be considered a user modeling server because it is not concerned with delivering and using the user model. Nevertheless it could be part of a generic user model server.

The work most related to our research is in the field of data mining and web mining frameworks [14], where some previous efforts have also focused on high performance [8]. However, these frameworks are not generally geared towards user modeling. In this context, [12] presents WIM (Web Information Mining) for web mining prototyping and provides a model and an algebra to express web mining tasks. The work presented here 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.

Note that the approach we take with our proposed architecture is very different from updating a user model 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 problem-specific architectures. Also, when short term and long term models must be generated, the advantages of having on-the-fly user models disappear as all models have to be updated over time independently of the activity of each user. In this paper, we leverage the advances in platforms and methods for data-intensive distributed computing in order to efficiently build user models.

3 Architecture for Terabyte-Scale User Modeling

The proposed architecture for building user models, named ARBUD, is depicted in Figure 1 and has four main components (described below): (1) The *Data* available to generate user models; (2) a *User Metamodel* that describes the components of the long term (LT) and short term (ST) user models; (3) a *user modeling (UM) Library* that contains the most common functions used to generate the different dimensions of the user models, and (4) a terabyte-scale *UM Generator* that instantiates the architecture and generates the set of user models from the information contained in the metamodel using the library. The output of the architecture is a set of user models for all users.

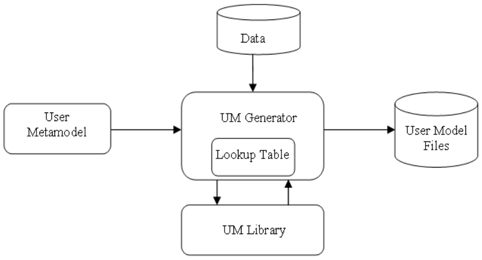


Fig. 1: Proposed architecture.

3.1 Data Input & Output

The data available to generate the user models may be in one or multiple files. It is very common that the data is collected using some time considerations, for example one file of logged user activity data 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 consist of a set of files along with a lookup table that indicates the file that contains the associated user model for each user. The user model files will be stored in a distributed file system (the Hadoop Distributed File System [2] in our case), but can be subsequently moved to a database or local file system.

3.2 User Metamodel

The metamodel contains all the parameters that describe the user model that is to be generated. This metamodel allows the architecture to factor out commonality across several application areas that use the same features or dimensions. It also allows for the easy use of pluggable components. The metamodel is a high level abstraction of the desired user models and hence it serves to clearly separate the user modeling from the model construction tasks. As a consequence, models can be built without detailed knowledge of the data layout or the distributed programming code underlying the architecture. The metamodel contains the following parameters:

1. *INPUT* is a pointer to the directory or files that contain the data to be processed. Typically, each file will contain the activity logs for all users during a given period of time. Each entry in the file will have a reference to the user ID and the user's activity in a specific moment in time.
2. LT_s, LT_e, ST_s, and ST_e correspond to the start and end times used to generate the Long Term (LT) and the Short Term (ST) models. Note that both models need not be generated, *e.g.* 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 of the N dimensions of the vector contains a specific feature derived from the user's data pointed to by INPUT. The processes for generating these features in a given context, *i.e.,* telecommunications applications or web mining, can be mostly 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 (features). Often they share many dimensions in order to measure recent deviations, but the short term model may include dimensions that only have significance over a short term period.
5. FILTERS indicates conditions that the user data must meet in order to generate a corresponding user model. One common condition used for filtering is ensuring a minimum level of user activity, thus user models are only built for users for whom there is enough data.
6. NEW-PARAMETER, contains one or more indications of what process to use to generate a feature of either the LT-UM or ST-UM for the case where the UM Library does not already contain the necessary process.

3.3 UM Library

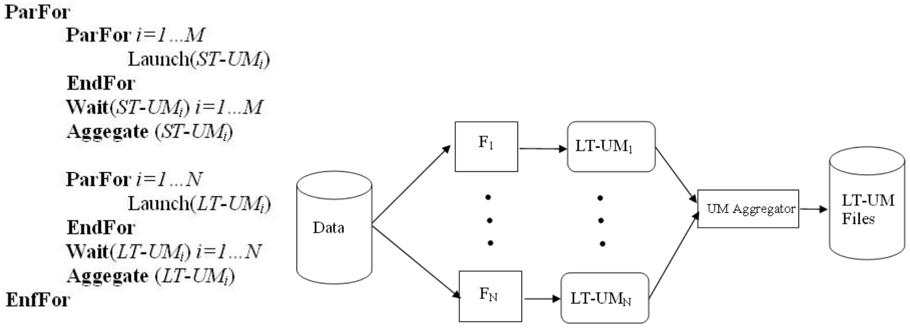
The UM Library contains the functions needed to calculate the features specified in the LT-UM and ST-UM. Note that for efficiency purposes, one or more features may be the output of a single function. For a given context, the UM Library will contain the typical elements that user models use in that environment. For example, in web mining there can be a function to identify the number of visits of a user to a web page or the number of times that a specific element of the interface has been used. In a telecommunications 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 Model Files by means of the necessary elements of the UM Library as well as any NEW-PARAMETERs. The Generator matches each of the elements of ST-UM and LT-UM to the corresponding functions of the UM Library with a lookup table which, for example in the case of LT-UM, maps each dimension LT-UM$_i$ to a function F_i. If the dimension LT-UM$_i$ is not found in the lookup table, it must be provided by NEW-PARAMETER and will subsequently be included in the UM Library and added to the table for future use. Figure 2a depicts simplified pseudocode for the UM Generator, which builds the short term and long term user models in parallel (the outer **ParFor**s). Within each user model, all features are computed in parallel (the inner **ParFor**s). The Generator waits until all features are computed, and then aggregates the results, with the

Generator ensuring that each function is only launched once for the case where multiple dimensions are generated by a single function.

Figure 2b presents an example of the workflow involved in building a LT-UM. The metamodel specifies the location of the input data, typically a set of time-stamped log files. The architecture then launches in parallel all functions $F_1, F_2, \cdots, F_N$ each of which produces a set of files containing records that contain a user ID and the associated feature in dimension LT-UM$_i$. The architecture waits for all functions F_i to complete and then launches another process to aggregate all the results. The aggregation process is responsible for merging all of the values for each of the i dimensions associated with each user. It produces as output a series of final LT-UM files where each record consists of a user ID and the full vector of N associated model features. As previously mentioned, multiple features may be computed by the same function for efficiency reasons, with the UM Generator designed to handle this.



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

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

(a) Pseudocode (b) Example Workflow for Generating LT-UM

Fig. 2: Pseudocode and Example Workflow for the UM Generator

3.5 Implementation

The proposed architecture is heavily influenced by the concepts and philosophy of the MapReduce [6] programming model and the accompanying distributed file system. 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 only in a database) which is true for many data sets used for user modeling, such as web server logs, telecommunications call logs or search engine query logs. MapReduce works with key-value pairs and consists of two basic stages: (1) A mapping stage that processes input records in parallel and emits intermediate key-value pairs, and (2) a reducing stage that gathers all values associated with a give key from the mapping stage and then reduces the set of associated values, emitting final key-value pairs. For the architecture considered here, the keys are often user IDs and the final values are typically one of the dimensions in the user model, so MapReduce is a natural choice for constructing the UM Generator upon, though a final aggregation stage is added to construct the final user models. The UM Generator has been implemented using Hadoop [1], which has two core components: an open-source Java implementation of MapReduce and HDFS [2], the associated distributed file system.

By building the system on top of Hadoop, with its near-linear scalability, the addition of more machines to a cluster allows for the construction of user models from potentially terabytes of data with no needed software changes. 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 user models.

4 ARBUD Applied to the Telecommunications Domain

In this section, we illustrate an example of how ARBUD can be used to build user models in the telecommunications domain. We describe some of the functions that are currently available in the UM Library and also illustrate a metamodel applied to churn prediction. A large number of the applications of user modeling in telecommunications focus on understanding how individuals use their phone, and how that knowledge can be used for providing better services and improve business intelligence. Common applications include churn prediction [13, 4] and fraud detection [9], all of which focus on user activity from the same data source – typically call detailed records (CDRs) which are records that indicate when users made or received phone calls, SMS messages, etc. In addition, the different user models for each application can have elements in common which can be shared using the reusable nature of the proposed architecture. In all these applications, the behavior of the user is represented by a vector that contains a set of features, and these vectors are used for training classification or regression algorithms. These are also examples of application areas where it is common to process very large datasets with tens of millions of users and where obtaining user models as quickly as possible is critical. For example, churn or fraud detection models are typically run on a daily basis in order to detect users at risk.

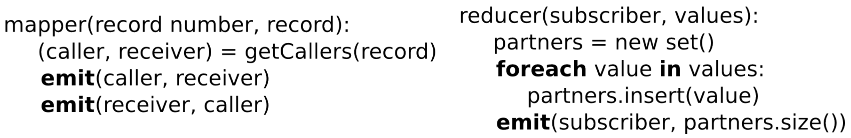
4.1 UM Library in the Telecommunications Domain

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

1. CALLS(): Represents, for each user, 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 produces multiple outputs or features. If only a subset is specified by the metamodel, the UM Generator will ignore the features that are 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, it obtains the IN-DEGREE, the OUT-DEGREE, and the TOTAL-DEGREE, where TOTAL-DEGREE is the number of unique users the user speaks with, while IN-DEGREE and OUT-DEGREE also consider the direction of the call when counting call partners.
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(): It identifies, for each user, how many times a new terminal has been used and the age of the current terminal.

Figure 3 presents a simplified example of the pseudocode for the function DEGREE() included in the telecommunications UM Library, where the calculation of the total degree for each user is implemented as a MapReduce procedure. The mapper takes each phone call log record and first parsers out the encrypted caller and receiver IDs. It then emits to the reducers two ordered pairs consisting of the call partners in both orders. Each invocation of the reducer receives all key-value pairs associated with a given key. In this case, the key is a subscriber ID and the set of values are all the talk partners for that subscriber. The reducer will build a set of all talk partners for the subscriber by iterating through all of the values it receives, and then computes the total degree for the subscriber, which is the size of the set. The subscriber ID and total degree are saved in a file for later aggregation.



```
mapper(record number, record):
    (caller, receiver) = getCallers(record)
    emit(caller, receiver)
    emit(receiver, caller)
```

```
reducer(subscriber, values):
    partners = new set()
    foreach value in values:
        partners.insert(value)
    emit(subscriber, partners.size())
```

Fig. 3: Example of MapReduce Code for Building DEGREE.

4.2 User Metamodels

In this section we describe an example of user metamodels for churn prediction built with the telecommunications UM Library. Churn is one of the main problems of any telecommunications operator and it is defined as the percentage of customers that leave the operator in a pre-defined time period. In general, the reasons for churn are varied, but there are a number of factors that are good indicators of churn, such as the number of individuals in a subscriber's social network that use another carrier, the number of complaints the subscriber makes to the operator, or a large number of phone calls to other service providers. The user model generated by our architecture will be the input to an SVM that has already been trained to identify people at risk of churning. The classification needs to happen on a daily basis, thus the need to have models of all users in a timely fashion (*i.e.,* daily in this case). 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 carrier; (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().NUM-CALLS.

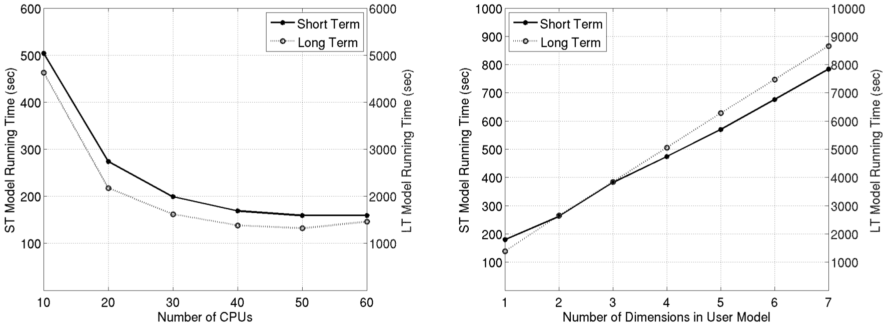
5 Performance Evaluation

In order to evaluate the performance and scalability of ARBUD, 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) to produce user models, was evaluated with respect to 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 Experimental Setup

For the purpose of the performance evaluation, anonymized phone call detailed records (CDRs) from a major mobile phone operator in a developed country were used. The information present in each CDR 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. From the telecommunications UM Library described in section 4.1, seven variables $\boldsymbol{v} = (v_1, v_2, \cdots, v_7)$ were chosen that exhibited similar performance characteristics (within one percent difference in running time). The time taken for building many variables depends mostly on data size, resulting in a large class of variables that exhibit very similar time complexity. Using the randomly chosen set of variables $\boldsymbol{v}$, 14 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 $MM_S^1, MM_S^2, \cdots, MM_S^7$, specified 3 days of CDR data comprising $2.18 * 10^8$ records and 10.1 GB with the ith short term metamodel MM_S^i denoting a user model consisting of the first i variables $(v_1, v_2, \cdots, v_i)$. The long term metamodels, denoted as $MM_L^1, MM_L^2, \cdots, MM_L^7$, were built from 30 days of CDR data, $2.17 * 10^9$ records and 103.0 GB with 20 million total users. Similar to MM_S^i, long term metamodel MM_L^i included the first i variables of $\boldsymbol{v}$.

The experiments were run on a compute cluster with five nodes. Each node consisted of 16 GB of RAM, 4 hard drives each 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 computer cluster had no other significant processes running. Version 0.20.1 of Hadoop was deployed on all nodes with several default settings changed in order to increase performance. In particular, we increased: (1) The maximum number of both map and reduce



(a) Time vs. Number of CPUs

(b) Time vs. User Model Size (40 CPUs)

Fig. 4: Time needed to build long and short term user models.

tasks run on each machine, (2) the size of the JVM for each task from 200 MB to 1 GB, (3) the size of sort buffers from 100 MB to 500 MB, (4) the sort factor from 10 to 100, (5) the I/O file buffer size from 4 KB to 64 KB and (6) the proportion of the total heap size used for retaining map outputs in memory during the reduce stage to 90%.

5.2 Experimental Results

One of the assumptions of the proposed architecture is that it is highly scalable. To test this premise, different HW configurations were used in order to disable different numbers of the cluster's CPUs, but always ensuring the same number of CPUs were enabled on each of the 5 machines in the cluster. For each configuration, one-variable long and short-term metamodels MM_S^1 and MM_L^1 were built using different numbers 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 MM_S^1 and MM_L^1. Note that 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 are added but eventually stabilizes with performance reaching a maximum at about 50 CPUs. This pseudo-exponential performance curve is in agreement with previous findings [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. All models were built multiple times, but variance in the performance was found to be minimal. Even with only 10 available CPUs, more than 2 billion CDRs were processed and MM_L^1 was built in about 1 hour and 17 minutes. Note that the memory needed to build certain dimensions surpasses the 16 GB of memory available on each machine, for example computing the DEGREE of all nodes requires about 24 GB of total memory, so it was not possible to compare these results with a traditional non-distributed approach.

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 same 7 short and long term metamodels ($MM_S^1, MM_S^2, \cdots, MM_S^7$ and $MM_L^1, MM_L^2, \cdots, MM_L^7$) were each passed 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 taken in seconds to build the short and long user models. Variance was found to be minimal and results are shown from one set of models built. 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. The correlation coefficient between the number of model dimensions and the short and long term running times are 0.9994 and 1.0000, respectively, providing strong evidence for linear scalability. It is worth noting that 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.

To test the maximum size of a dataset that ARBUD would be able to handle in order to create daily model refreshes, different 7-dimensional long term models were built using five $8 - CPU$ machines, each time increasing the dataset size. The time needed to build 1.02 terabytes of input data was found to take 24 hours and 10 minutes, just a little more than time than a daily build requires but demonstrating that the architecture can scale to terabyte-sized datasets.

6 Conclusions & Future Work

In this paper, we have presented ARBUD, an architecture for building both long and short term user models that can gracefully scale to handle massive datasets. The architecture uses metamodels to describe the desired user models, allowing for the abstraction of commonality from different user model building processes. ARBUD can build different short and long term user models from massive data sets with little to no new software development. Hence, it is generic and can be used for any application area that has record-based log files of user activity or behavior. In addition, the architecture allows for reusable components that can be applied in different domains.

In performance tests with phone call data logs of 20 million users, we have found that increasing computer resources, as measured by the number of available CPUs in a computer 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 have also verified that ARBUD's performance scales linearly with the number of dimensions in the user model. We have experimentally demonstrated that ARBUD can build a 7-dimensional user model from a 1.02 terabyte dataset in little more than 24 hours on a small 5 node computer cluster.

These results highlight the ability of the proposed architecture to take advantage of recent advances in distributed computing to produce user models efficiently, and thus is relevant for applications that need to build user models from massive datasets 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 application areas in the telecommunications domain and will create a web mining UM library, in order to automatically build 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 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*, 2005.