SocialGlue: a Pluggable, Scalable, and Multi-Platform Service for Data Analysis

Stefano Amico, Marika Cappai, Salvatore Carta, Luca Mancosu,
Fabrizio Mulas, Maria Luisa Mulas, Paolo Pilloni, and Giordano Sini.
Dip.to di Matematica e Informatica
Università di Cagliari
Email: stefano.amico@domoslab.com, marika.cappai.unica@gmail.com,
{salvatore, luca_mancosu, fabrizio.mulas, marialuisamulas, paolo.pilloni, giordano.sini}@unica.it

***Abstract*—The growing use of Social Network Sites (SNS) and the exponential growth of social data is opening new research challenges in several scientific domains, such as: Recommender Systems, Data Mining, Sentiment Analysis, and Human-Computer Interaction, just to name a few. In this scenario, researchers from different fields are facing several obstacles given the lack of highly customizable frameworks able to let them perform a large number of ad hoc studies of big data flows coming from SNSs. Trying to overcome these challenges, this paper sets out *Social Glue (SG)*, a pluggable, scalable, and multi-platform social analysis service. SG is designed to easily enable the connection to potentially any SNS allowing scientists to plug and manage the execution of their algorithms against connected SNSs seamlessly from SG with the minimum effort. In this way, researchers can focus more on the design of algorithms rather than in the software infrastructure needed to set up their experiments.**

Keywords–Social Web; Computational Services; Framework Architecture.

I. INTRODUCTION

Being able to collect and process data is essential in all the fields of science and engineering [1]. Moreover, most fields nowadays are interdisciplinary and collaborative [2]. This means that in the so-called Social Web (also known as Web 2.0), *web communities* can generate and share large amounts of data. However, due to privacy and commercial reasons, the data generated on one environment (e.g., the e-commerce transactions) is not passed to other environments. Therefore, to generate services to a web community by analyzing and filtering the data generated by the users, each environment and system has to collect its own data. However, if a single environment to run multiple services without disclosing of sensible data to the service providers existed, the problem would be overcome and each service could run on much more data, improving its effectiveness.

As Koch and Lacher highlight [3], computer-based systems should support knowledge transfer and the exchange of information generated by the community, in order to integrate different services provided to the community. In this matter, they propose a set of guidelines to create a common infrastructure that enables this knowledge transfer. Karacapilidis et al. [4] proposed a web-based collaboration support platform for the biomedical domain, to strengthen collaborative data analysis and decision making in web communities.

As both the research works mentioned above clearly point out, web communities tend to generate large amounts of data that are recognized as a valuable source, and *the development of a flexible and domain-independent framework that allows researchers and developers to exploit and analyze these data and provide services to the users represents an open issue in the current literature*.

To overcome the aforementioned issues, in this paper we present *Social Glue (SG)*, a pluggable, scalable, and multi-platform framework, that enables the providing of any service to a web community. Indeed, our proposal allows researchers and developers to plug, run, and use their algorithms on a connected Social Network, with the minimum programming effort. Therefore, the data integration highlighted in the literature as a requirement for a system that provides services to a web community is made possible by *Social Glue*, since the system is able to process the plugged algorithms on the data generated by the web community. In other words, the data generated in one environment, can be exploited in many different contexts. The proposed framework has been designed to exploit modularity and employ state-of-the-art technologies.

The main scientific contributions coming from this paper are the following:

- the experimentation of a novel domain-independent framework to provide services in Social Web environments is proposed in the literature; each algorithm can run on our framework, by employing different types of data generated by the web community while using a social network;
- thanks to *Social Glue*, researchers and developers themselves are provided with services offered by the framework, i.e., algorithms upload, data and user base availability, back-end interfaces to plug the algorithms, and so on.

The rest of the paper is organized as follows: Section II provides an overview of the state of the art in the field of computational services for web communities, Section III provides a high level description of the architecture of *Social Glue*, while Section IV sets out a detailed description of each layer composing the system architecture; Section V contains conclusions and future work.

II. RELATED WORK

This section presents related work on platforms that support web communities to provide services to them.

In [4], the authors present Dicode, a framework to support a biomedical research community at collaborative analyzing data and enable decision making processes. The platform is flexible, in the sense that heterogeneous workloads can be processed by the framework. However, no layer that enables external contributions to process data in different ways is presented, so the system can only be employed by the community to run a fixed set of services.

 ISBN: 978-1-61208-459-6

Tiwana and Bush [5] presented an architecture to enable social exchange in distributed web communities. Again, with social exchange the authors intend the transfer of information among the users, but differently from SG, the possibility to employ the social data to run different types of services on a flexible architecture is not available.

Koch [6] presented an approach to support interoperable community platforms in the university domain, by also allowing the members of the community to manage their identity. The approach, however, assumes that different platforms for different services and communities exist, while in our approach we aim at developing a unique framework and environment to provide different types of services and exploiting all the data generated by the web community. In [7], the author considers also aspects such as the modularization of the provided services and the use of ubiquitous user interfaces.

The capability to provide different types of services has been developed in several systems as a form of context-awareness (i.e., to provide services according to the context in which the system operates). Zafar et al. [8] present an architecture of these types of systems. This type of frameworks provide ad-hoc services, like location aware tour-guides [9], [10]. Therefore, a system is devoted at providing a single service, and there is no chance to integrate other types of algorithms to run on these frameworks.

As this analysis shows, our proposal has some peculiarities that set us apart from other similar frameworks proposed in the literature. Our solution provides a way to integrate different types of services in a flexible framework to provide the knowledge transfer and data integration highlighted in the Introduction.

III. High Level Architecture

This section introduces the architecture of the proposed framework, by giving an overview of the layers that compose it.

The *Social Glue* framework is characterized by a multi-layered architecture composed of three macro-layers: the *Clients* layer, the *Back-end* layer, and the *Modules* layer. Each layer is in turn composed of sub-modules called *Plugins* in the Clients layer, *Adapters* in the Back-end layer, and *Modules* in the Modules layer. As can be seen in Figure 1, at the top of the architecture sits the *Clients* layer which exploits low level layers to give users the possibility to extend the framework by adding customized elements called plugins. Plugins are the tools users have to develop in order to interconnect with real-world social networks, to create custom web communities and, more important, to manage the execution of their algorithms against available communities. As an example use case, Social Glue natively implements two plugins built exploiting a famous social network engine called *Elgg* [11]: the *Simulator Movie* and the *Social Monitor* plugin. The former implements the functionalities needed for the creation and the management of a web community of movie lovers, the set up of algorithms to be executed against the community, and the management of the output of executed algorithms. The latter, instead, is in charge of implementing an administration interface to allow users to control the lifecycle of all available algorithms.

All the functionalities offered by a plugin are supported by the *Back-end* layer. This layer exposes a set of REpresentational State Transfer (REST) Application Programming Interfaces (APIs) that are responsible for the management of the algorithms. The Back-end layer is composed of a variable number of "Adapter" classes that manage the loading of the algorithms plugged in the system through the lower level of the architecture. In general, there must be an adapter for every available algorithm. The functionalities of the adapters are exposed to the higher level by means of a standard API in order to give clients full control on the execution of algorithms.

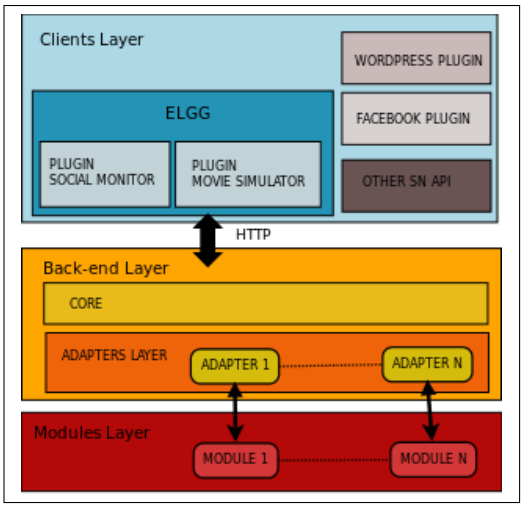

Figure 1. Social Glue Framework

Finally, the *Module* layer actually contains the source code of the algorithms researchers plugged in the system. Every algorithm inserted constitutes a module that has to expose a standard interface to let the corresponding adapter in the Back-end layer expose all the functionalities the plugins in the Clients layer need to control the execution of the algorithms.

The low level layers have been designed to allow users to deploy system configurations able to scale horizontally in cases where multiple and potentially resource-intensive computations have to be performed in parallel. In fact, the the web service is able to correctly handle multiple databases sources and multiple modules/adapters provided that the logic flow of the program is designed to work in a stateless way and a web proxy is properly configured to route the HTTP traffic.

IV. Architecture Layers

The following section provides a detailed description of the previously presented layers composing Social Glue.

A. Clients Layer

As highlighted in Section III, at the top of the architecture sits the *Clients* layer. The entire system has been designed keeping in mind the modularity it has to offer to the final users and this layer is no exception. Indeed, a user can develop and plug into this layer her/his social plugins implementations to exploit public APIs to interact, for example, with a real world social network or even with a user-created social network.

In general, in addition to the implementation of the plugin that handles the communication with the social network, it is possible to implement also a custom administration plugin in order to provide a user-friendly web area where administrators can manage the APIs exposed by the Back-end layer (see

 ISBN: 978-1-61208-459-6

Subsection IV-B). By default, the framework provides an administration area to let users manage the lifecycle of the loaded algorithms. In cases where it is needed a finer control over the management of the algorithms, it is possible to implement custom administration functionalities by exploiting the APIs exposed by the lower level.

In the following, we will provide a more detailed description of the use case previously introduced (see Section III). This use case has been developed to set up a series of preliminary tests to investigate how the entire system behaves in terms both of reliability and performance during the execution of popular recommendation algorithms against a social community. As mentioned before, for our use case we decided to opt for a well-known and open source framework called Elgg that offers the possibility to build web communities with the minimum development effort. The framework is very powerful and in its base configuration is able to support blogging and microblogging functionalities, groups management, and several common social actions such as: comments on posts, tags, likes, and so on.

Elgg's architecture is highly modular and extensible. Its plugin system allows the user to easily develop and integrate new plugins inside the framework in order to provide her/his customized functionalities. As previously mentioned, for our use case we exploit Elgg's flexibility to create a social network for movie lovers that we populated using a well-known dataset called Movielens [12]. The community is managed by means of a custom plugin called *Simulator Movie*. This plugin supports the insertion of new users and movies, the population of the community by means of a dataset and so on. In addition to the community-related features, the plugin gives administrators the possibility to load algorithms to be executed against the community by exploiting the lower layers of the architecture. In our case, we loaded a recommendation algorithm to build tailored movies recommendations by exploiting users' preferences contained in the dataset we used to populate the community. In this way, we were able to provide users with tailored movies recommendation directly inside the social network (see Section IV-C).

The execution and management of the algorithms is demanded to another custom Elgg plugin called *Social Monitor*. Social Monitor implements all the functionalities related to the management of the algorithms the administrators intend to execute against a certain community. Figure 2 shows a screenshot of the Social Monitor. Starting from the top of the screen, the user has an overview of all the algorithms she/he can launch. In particular, it is indicated the name of the algorithm, whether or not some form of input data is required, and the possibility to launch the execution by means of the "Execute" button. Just below the "Algorithms" menu is shown the "Thread Pool" view that reports the current state of the low level threads that are in charge of executing the algorithms. In this particular case, it reports that there are two threads currently in execution and both of them are waiting for their inputs (see the "Waitdata Instances" label). The "Thread Pool Information" shows low level details about the current state of the Thread Pool. In particular, starting from the top of the view, the following information is shown: the current number of busy threads, the total capacity of the queue (i.e., the maximum number of algorithms that can be enqueued for execution), and the maximum number of threads the pool can contain. Finally, at the bottom of the screen, is shown the "Running Algorithm Status" that reports general information about the algorithms currently in execution. In this particular case, there are two running algorithms both waiting for their input data.

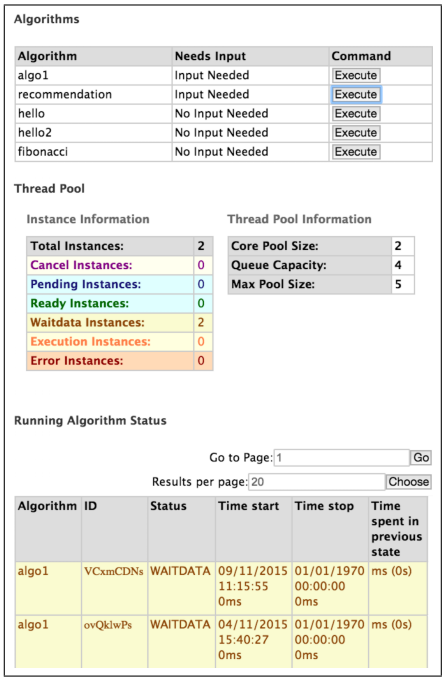

Figure 2. Social Monitor plugin

B. Back-End Layer

The Back-end layer is in charge of providing the core services to the higher layer. From a technological point of view, we decided to adopt a well-known and widely used software stack. The core of the Back-end is written in Java and implemented by exploiting the Spring framework [13] and the MySQL [14] database to manage the persistence of the data. The entire application runs on the Apache Tomcat [15] web application server. Thanks to this high flexible software architecture we implemented both the core logic and the set of RESTFull APIs invoked by the higher layer. This layer is responsible of managing the run-time load and unload of the user/researcher provided algorithms and, most important, it implements the control logic that handles the entire lifecycle of the algorithms plugged in the system by means of the lower layer (see Section IV-C).

The control logic is instructed on how to execute a certain algorithm by means of a simple configuration file that the user/researcher is requested to provide for every algorithm she/he intends to load. The following string represents an example of an entry of the configuration file:

{*"name":recommendation,"class":RecAlgoAdapter,"needInputs":true*}

The first attribute, called "name", indicates the name of the algorithm, "class" reports the actual Java class (Adapter) in charge of handling the execution of the algorithm loaded from the lower layer, and "needInputs" is a boolean value that specifies whether or not it is necessary to manually upload the input for the algorithm.

 ISBN: 978-1-61208-459-6

The most important component of the Back-end, in charge of handling all the steps needed to execute the algorithms, has been implemented by means of a finite state automata (see Figure 3). We opted for this tool because we needed a

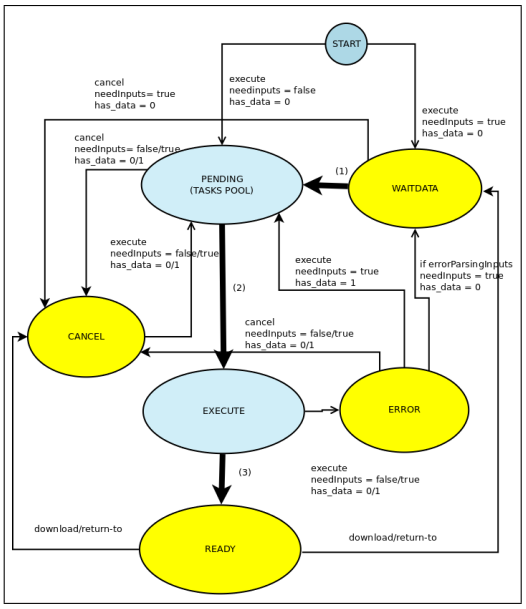

Figure 3. Back-end finite state automata

relatively simple and, at the same time, robust mechanism able to handle the concurrent execution of potentially very heterogeneous algorithms. Indeed, the entire system must be able to correctly execute algorithms that can differ a lot in terms of computational complexity, input/output dimension, memory consumption, and so on.

Let us now describe in detail the automata and, in particular, its states and the possible transitions between states that can take place during the execution of an algorithm. A client, through an ad hoc graphic control panel that in turn invokes the Back-end APIs, can issue the commands for a particular algorithm in order to trigger the transitions between states. As can been seen in Figure 3, the initial state is called "START" and this is the state that is associated to the invocation of the "execute" command. According to the type of algorithm being executed (i.e., according to the configuration provided by the user by means of the configuration file), the automata can transit to two different states. In particular, the state changes to "WAITDATA" if the current algorithm needs to be provided of input data (i.e., the label "needInputs" is equal to true). When an input is provided (i.e., the label "hasData" is set to 1), the current active state becomes "PENDING". The same state is reached directly when the algorithm being executed does not need input data (i.e.,"needInputs" is equal to false). At this point, the algorithm is ready to be executed and the control passes to the execution logic that has been implemented using a common thread pool. When one of the threads in the pool becomes idle (i.e., it can execute the current algorithm), the active state becomes "EXECUTE" (i.e., the system starts the execution of the algorithm). Eventually, the system reaches the "READY" state when the execution of the algorithm finishes correctly. On the contrary, a transition to the "ERROR" state signals that the execution has been terminated because of an unexpected error. In this case, the execution of the algorithm gets stopped and the system waits for the intervention of the user. From "READY", the automata can pass to "CANCEL" when the user downloads the results and then again to the "PENDING" state if the user triggers another "execute" action. From the "READY" state the control can pass to the "WAITDATA" state in cases when the user, after downloading the results, chooses to provide new input data for the current instance of the algorithm in order to wait for a new execution by following the same state transitions described above.

Let us now describe in more detail the "CANCEL" state previously introduced. An algorithm can be in this state when:

- the user has executed the "download and cancel" command after a correct execution of the algorithm;
- the user can decide to cancel the execution because she/he deems that the current algorithm is waiting in the "PENDING" state for a too long time (i.e., the entire system could be busy).

It is important to note that the system, when the current state is "CANCEL", is able to save the provided input and configurations in order for the user to restart the execution at a later time. Finally, as we saw before, the "ERROR" state is reached when the execution of an algorithm terminates abnormally. From this state, there are three possible transitions towards the following states:

- "CANCEL": the user decides to cancel the execution of the current algorithm;
- "PENDING": the user decides to reschedule the execution of the algorithm using the same configurations;
- "WAITDATA": the user decides to relaunch the execution by providing different inputs to the algorithm.

C. Modules Layer

The *Modules Layer* is designed to provide researchers and developers the opportunity to integrate their algorithms on the framework. Each algorithm can be seen as a module that runs on the framework and is a service provided to the researcher/developer. The integration of the source code of each module (that can also be in a different programming language, like C or C++), is made possible by the Java Native Interface (JNI), which enables Java to use the native code of the operating system.

We will now provide an example of a module integrated in our framework, which is a *user-based Collaborative Filtering recommendation algorithm*. The choice to integrate this type of module in our framework was made because it lends itself well to a web community domain, characterized by social interactions. Indeed, the algorithm is able to analyze the different items a user had an experience with, and provide recommendations for items she/he has not considered yet, but that might interest her/him. In general, with the term *item* we refer to something generic, which in a social domain might be anything, from a Youtube video a user has posted, to a restaurant she/he visited and evaluated on Facebook.

In order to provide the recommendations, a *user model* like the one in Table I is considered. A model contains for each *item* i_n that the user u evaluated, a *rating* r_{un} that expresses with

TABLE I. EXAMPLE OF *USER MODEL*

	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8
u	r_{u1}		r_{u3}				r_{u7}	r_{u8}

TABLE II. EXAMPLE OF *RATINGS MATRIX*

	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8
u_1	r_{11}		r_{13}	r_{14}			r_{17}	r_{18}
u_2	r_{21}	r_{22}		r_{24}	r_{25}		r_{27}	
u_3		r_{22}	r_{33}	r_{34}		r_{36}		
				...				
u_n	r_{n1}	r_{n2}	r_{n3}		r_{n5}	r_{n6}		r_{n8}

a numerical value how much the user likes the corresponding item.

All the user models, like the one previously described take the form of a matrix, usually called *rating matrix*, like the one in Table II.

The missing ratings for each user are predicted with a widely-used User-Based Nearest Neighbor Collaborative Filtering algorithm, presented in [16]. The algorithm predicts a rating p_{ui} for each item i that was not evaluated by a user u, by considering the rating r_{ni} of each similar user n for the item i. A user n similar to u is called a *neighbor* of u. To indicate that we are dealing with a user-based approach, the set of neighbors of this algorithm will be indicated as $neighbors^{uu}$. Equation 1 gives the formula used to predict the ratings:

$$p_{ui} = \overline{r}_u + \frac{\sum_{n \subset neighbors^{uu}(u)} userSim(u,n) \cdot (r_{ni} - \overline{r}_n)}{\sum_{n \subset neighbors^{uu}(u)} userSim(u,n)} \quad (1)$$

Values $\overline{r}_u$ and $\overline{r}_n$ represent, respectively, the mean of the ratings expressed by user u and user n. Similarity $userSim()$ between two users is calculated using the Pearson's correlation [17], a coefficient that compares the ratings of all the items rated by both the target user and the neighbor. Pearson's correlation between a user u and a neighbor n is given in (2). I_{un} is the set of items rated by both user u and user n.

$$userSim(u,n) = \frac{\sum_{i \subset I_{un}} (r_{ui} - \overline{r}_u)(r_{ni} - \overline{r}_n)}{\sqrt{\sum_{i \subset I_{un}} (r_{ui} - \overline{r}_u)^2} \sqrt{\sum_{i \subset I_{un}} (r_{ni} - \overline{r}_n)^2}} \quad (2)$$

The values of the metric range from 1.0 (complete similarity) to -1.0 (complete dissimilarity). Negative correlations do not increase the prediction accuracy [18], so they are discarded.

The output of the algorithm is a ranked list of top-n items with the highest predicted rating, which is recommended to each user of the community.

In order to evaluate the recommendation algorithm's accuracy, we need to choose the number of neighbors for each user (parameter $neighbors_{uu}$). The experiments that run the algorithm with different values of the parameter are now presented. Figure 4 shows the RMSE of the prediction algorithm for increasing values of $neighbors^{uu}$; this is the common way to choose the value [19]. Our results reflect the trend described by the authors, i.e., for low values of the parameter, great improvements can be noticed. As expected, RMSE takes the form of a convex function (Figure 5 shows a detail of Figure 4), which indicates that after a certain value improvement stops. In the experiment that value is 100.

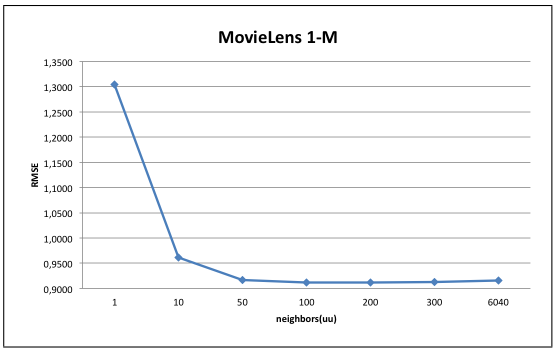

Figure 4. RMSE values for increasing number of $neighbors^{uu}$ in the MovieLens-1M dataset

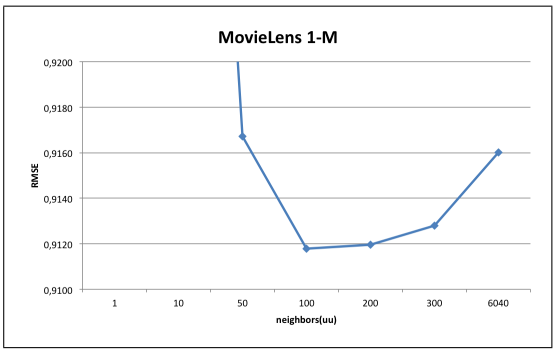

Figure 5. RMSE takes the form of a convex function in the MovieLens-1M dataset

Independent-samples t-tests to evaluate the difference between the results obtained between 100 and the other numbers of neighbors, are presented in Table III. Results show that there is no difference between choosing 100 and 200 neighbors. Since it is faster to compute predictions considering 100 neighbors instead of 200, $neighbors^{uu} = 100$ is the value chosen for the algorithm.

V. CONCLUSION AND FUTURE WORK

In this work, we presented Social Glue which is a scalable, pluggable, and multi-platform social analysis service designed to overcome many of the obstacles researchers face during the analysis of huge amounts of data coming for example from social networks sites. Social Glue is able to easily integrate with any kind of social network site allowing scientists to execute any kind of algorithm by means of its advanced plugin mechanism. In this way, researchers can both launch

TABLE III. STUDENT'S T-TESTS - VALUES OF THE $NEIGHBORS^{UU}$ PARAMETER.

MovieLens-1M	**p**
$RMSE_1$ **vs.** $RMSE_{100}$	0.00
$RMSE_{10}$ **vs.** $RMSE_{100}$	0.00
$RMSE_{50}$ **vs.** $RMSE_{100}$	0.00
$RMSE_{100}$ **vs.** $RMSE_{200}$	0.82
$RMSE_{100}$ **vs.** $RMSE_{300}$	0.33
$RMSE_{100}$ **vs.** $RMSE_{6040}$	0.24

 ISBN: 978-1-61208-459-6

and monitor their algorithms focusing on the experiments rather than wasting a considerable effort on the software infrastructure needed merely to set up the experimentations.

The usage scenario we described clearly shows the potential benefits the platform can provide. As a future work, we are planning a large scale experimentation based on data collected from multiple real big data providers in order to evaluate how the service performs with different and well-known analysis algorithms.

ACKNOWLEDGMENT

This work is partially funded by Regione Sardegna under project SocialGlue, through PIA - Pacchetti Integrati di Agevolazione "Industria Artigianato e Servizi" (annualità 2010).

REFERENCES

[1] "Challenges and opportunities with big data," Tech. Rep., Spring 2012. [Online]. Available: http://cra.org/ccc/wp-content/uploads/sites/2/2015/05/bigdatawhitepaper.pdf

[2] E. S. Lee, "Facilitating collaborative biomedical research," in GROUP '07 Doctoral Consortium Papers, ser. GROUP '07. New York, NY, USA: ACM, 2007, pp. 5:1–5:2. [Online]. Available: http://doi.acm.org/10.1145/1329112.1329117

[3] M. Koch and M. Lacher, "Integrating community services-a common infrastructure proposal," in Knowledge-Based Intelligent Engineering Systems and Allied Technologies, 2000. Proceedings. Fourth International Conference on, vol. 1, 2000, pp. 56–59 vol.1.

[4] N. Karacapilidis, S. Christodoulou, M. Tzagarakis, G. Tsiliki, and C. Pappis, "Strengthening collaborative data analysis and decision making in web communities," in Proceedings of the 23rd International Conference on World Wide Web, ser. WWW '14 Companion. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2014, pp. 1005–1010. [Online]. Available: http://dx.doi.org/10.1145/2567948.2578845

[5] A. Tiwana and A. A. Bush, "A social exchange architecture for distributed web communities," J. Knowledge Management, vol. 5, no. 3, 2001, pp. 242–249. [Online]. Available: http://dx.doi.org/10.1108/13673270110401220

[6] M. Koch, "Interoperable community platforms and identity management in the university domain," International Journal on Media Management, vol. 4, no. 1, 2002, pp. 21–30. [Online]. Available: http://dx.doi.org/10.1080/14241270209389977

[7] ——, "Requirements for community support systems - modularization, integration and ubiquitous user interfaces," Behaviour & Information Technology, vol. 21, no. 5, 2002, pp. 327–332. [Online]. Available: http://dx.doi.org/10.1080/0144929021000048484

[8] M. Zafar, N. Baker, B. Moltchanov, S. L. João Miguel Goncalves, and M. Knappmeyer, "Context Management Architecture for Future Internet Services," in ICT Mobile Summit 2009, Santander, Spain, Jun. 2009.

[9] S. Yasuyuki et al., "C-MAP: building a context-aware mobile assistant for exhibition tours," in Community Computing and Support Systems, Social Interaction in Networked Communities [the book is based on the Kyoto Meeting on Social Interaction and Communityware, held in Kyoto, Japan, in June 1998], ser. Lecture Notes in Computer Science, T. Ishida, Ed., vol. 1519. Springer, 1998, pp. 137–154.

[10] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou, "Developing a context-aware electronic tourist guide: Some issues and experiences," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ser. CHI '00. New York, NY, USA: ACM, 2000, pp. 17–24.

[11] "Elgg," https://elgg.org/, retrieved: 01-2016.

[12] "Movielens," https://movielens.org/, retrieved: 01-2016.

[13] "Spring framework," http://projects.spring.io/spring-framework/, retrieved: 01-2016.

[14] "Mysql," https://www.mysql.com/, retrieved: 01-2016.

[15] "Apache tomcat," http://tomcat.apache.org/, retrieved: 01-2016.

[16] J. B. Schafer, D. Frankowski, J. L. Herlocker, and S. Sen, "Collaborative filtering recommender systems," in The Adaptive Web, Methods and Strategies of Web Personalization, ser. Lecture Notes in Computer Science, vol. 4321. Springer, 2007, pp. 291–324.

[17] K. Pearson, "Mathematical contributions to the theory of evolution. iii. regression, heredity, and panmixia," Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, vol. 187, 1896, pp. 253–318. [Online]. Available: http://rsta.royalsocietypublishing.org/content/187/253

[18] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," in Research and Development in Information Retrieval, American Association of Computing Machinery. American Association of Computing Machinery, 8/1999 1999.

[19] C. Desrosiers and G. Karypis, "A comprehensive survey of neighborhood-based recommendation methods," in Recommender Systems Handbook. Berlin: Springer, 2011, pp. 107–144.

 ISBN: 978-1-61208-459-6