Informatics in Education, 2011, Vol. 10, No. 2, 149–162

Distributed Social Bookmarking Web Service Architecture. SOAP vs. iCamp FeedBack

Andrej AFONIN
Kaunas University of Technology, Research Laboratory of e-Learning Technologies
Studentų 67-513, LT-51392 Kaunas, Lithuania
e-mail: andrej.afonin@ktu.lt

Received: December 2010

Abstract. Social bookmarking services became very popular recently. Easy of use, possibility to share and discover in addition to accessibility though the Internet, turns social bookmarking systems into powerful repository of shared knowledge. Obviously this attracts attention of educational institutions and recently such systems started to appear under their domains. However, usually these systems stay separate and limit their users by their bounds. It means that separate systems' students could reach each other and use knowledge base, aggregated in other systems. On the other side, institutions usually want to own this assembled data and do not give away collected knowledge base to third side. This issue does not allow building social bookmarking systems that can be used by multi-institutional users. An idea is to develop distributed system where every institution will have their own database, but, on the other hand, will allow exploring and using data from other network sources. This article overviews possible distributed system architecture models and suggests a solution that will eliminate such service issues. Moreover, two different approaches towards distributed services communication are evaluated in this article: SOAP vs. iCamp FeedBack. SOAP is a lightweight XML based protocol for exchanging structured information between distributed applications. FeedBack is another model that uses plain RSS feed to transmit data. Both models are tested and evaluated in this article.

Keywords: distributed system, software architecture, web 2.0, social bookmarking system, SOAP, iCamp FeedBack.

1. Introduction

Social bookmarking systems kicked of the web 2.0 craze in 2003. Joshua Schachter found the website called del.ico.us in late 2003. He proposed a different way of storing bookmarks with what was used before. The innovative idea was to move bookmarks from browser's preferences to online server, where they would become available from any computer that has internet access 24 hours a day. Additionally, a new social way of information organization and tagging system benefits provide broader possibilities to explore and access them. Since that time, hundreds of social bookmarking services were launched, with the most popular (in no particular order): Google Bookmarks, Digg, Reddit, Shashdot, Furl, Spurl, Newswine, Yahoo MyWeb, StumbleUpon, Technorati, LinkaGoGo, Twine, Diigo and many more.

Stunning popularity of bookmarking sites was quickly noticed by educators, they found it attractive and started using it in learning process (D'Souza, 2006). Online community, tagging system and online sharing provided wide possibilities for research, share and students collaboration (D'Souza, 2006).

In Icamp project (`www.icamp.eu`) social bookmarking services were admitted into student's portfolio as a part of networking tools group. Social bookmarking tools were used to help different countries students, working on the same topic, to aggregate and share useful internet resources. Interestingly, that first Icamp trial showed that students from different countries prefer their own local services instead of global. Additionally, project mentors had raised data ownership problem: why university has to give away students' generated data to a third part services? Finally, exactly users' generated data and their activeness are most important aspects defining web 2.0 startup price.

The problem solution was to develop distributed social bookmarking system that will satisfy universities in their wish to keep data and users on one side, and also, provide distributed solution benefits, such as bigger amount of data and wider networking possibilities to students. That was the first challenge in system development.

Another challenge was a suitable distributed architecture meeting bookmarking service performing needs. Working on possible distributed patterns evoked four different architectural designs of distributed social bookmarking systems: federation search (through middleware), replication (through middleware) or managed synchronization, peer-to-peer synchronization, pull/push and aggregate. All these models are presented in this paper with a special attention to selected model.

And the last challenge was a cross-site scripting problem. Due to security reasons it is not allowed to communicate with remote database directly. This problem is well known and is usually solved implementing communication using SOAP protocol. But the performance of SOAP is rather low because it needs extra process parsing and building message to process an XML document (Takeuchi *et al.*, 2005; Davis and Parashar, 2002). Especially, that parsing SOAP message causes high CPU load (Takeuchi *et al.*, 2005). Also, when using multiple system calls to send one message, it is a large source of inefficiency in SOAP (Davis and Parashar, 2002). Typical high performance systems use RMI or CORBA technologies instead (Takeuchi *et al.*, 2005). On the other hand, XML use in SOAP protocol helps to archive high interoperability between distributed systems (Chiu *et al.*, 2002). This is more important, if considered that cross domain communication in this particular case is going to perform between two social bookmarking systems, that have been build in RSS feed delivery mechanisms. Therefore, it makes sense to use iCamp FeedBack mechanism that is dealing with plain RSS feeds and light-weight RPC-calls (Wild *et al.*, 2007). Initially it was developed to run communication between separate blogs to push posting updates to all subscribers. iCamp FeedBack is based on RSS feed use, so it might work faster, that heavyweight SOAP protocol.

In this article, these two approaches will be compared trying to decide which one is more suitable for developing distributes social bookmarking system.

2. Building Distributed Architecture Design

Common system architecture vision in combination with reviewing state of the art conduced to develop several designs of developing distributed solution. There are two main approaches to designing back-end (i.e., database layer) of distributed database system. The database could be replicated, partitioned or combination of both. A replicated database is one, where full or partial copy of database exists on different platforms. A partitioned database is one, when part of database is stored on one platform and part on another(s) (Beverage, 2002).

A research on distributed system architectures brought four main models widely used in distributed systems development. George Couloures (2003) defines client-server model, services provided by multiple servers, proxy servers and cashes and peer processes model as main models (Couloris *et al.*, 2003).

A review on most popular distributed systems architecture and back-end design approach allows to define four most suitable models to build distributed social bookmarking system. Defined models allow analyzing advantages and disadvantages of each schema and select which model meets requirements best. Prototype analysis could give a brief view, on which schema will require less changes in existing software that is going to be reused. Every solution was named upon key features of each model: joint central middleware service, pull/push & aggregate, synchronize and federate. Advantages and disadvantages will be discussed subsequently.

But before starting working on patterns, definition should be done on what design requirements are raised for distributed system. Coulouris *et al.* (2003) defines a list of requirements, which have to be evaluated during design process. These are performance issues, quality of service, use of cashing and replication, dependability issues, fault tolerance and security.

2.1. *Pattern 1: Federated Search Through Middleware (Joint Central Middleware Service)*

A design of the first pattern is based on partitioned database distribution model. In this case, we have a federated system of local systems that work independent (services provided by multiple servers architecture) and are accessed by others using middleware layer (proxy servers architecture). The basic pattern's idea is, that every local social bookmarking system is an independently working system without true "knowledge" of the existence of other systems.

Figure 1 illustrates architecture of this model. Data managing process for local user is the same as in undistributed system. Local system user works with a local version of the system and database, performing actions with data on the same local machine. The difference comes, when service user wants to make a "distributed" search. In this case, local server is calling another server, which is called mediator, forming query with a search keywords. Mediator is a web service (proxy server) that works as a middleware layer between query source and search target and "knows", where other social bookmarking

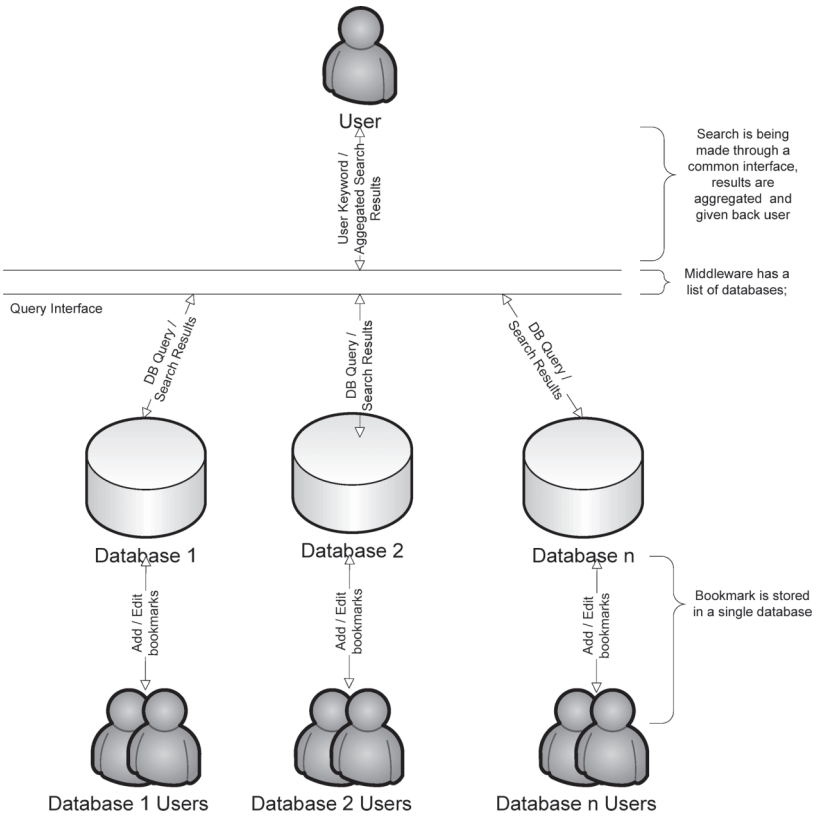

Fig. 1. Joint central middleware service model.

services are located. Middleware is a layer that is responsible for providing transaction management, messaging, distributed data, web serving, and other services (Beverage, 2002). An important aspect of middleware is the provision of location transparency and independence from the details of communication protocol, operating system and computer hardware (Couloris *et al.*, 2003).

Mediator is querying all known federated social bookmarking services with keywords, forming response list with wanted data and gives it back as RSS feed to search initiator – local bookmarking system's search portlet. The last one performs grouping, sorting or any other actions with received data and shows it to the end-user.

The main advantage of this model is that every stored data piece is unique and is not duplicated anywhere else. Also it does not require database changes of queered databases. The whole shared data discovery work lies on a middleware and query/retrieve algorithms. Likewise, the model has a small dependency on nodes availability, because one node down will not affect the whole system work, but will decrease a number of data retrieved from distributed nodes. But on the other hand, this model causes more problems than gives benefits. First of all, such model requires a person, who will be administrating mediator, which will be adding new and removing old service's addresses. Second,

problems with server response time and results rendering on a user machine comes along, that is when system does not know, if there are some more results coming from "slow" servers. These problems can be solved by making additional data flow management on slow node respond, but it will require additional user interface change and that is better to voide. Besides, it is important to evaluate special social bookmarking features' work, such as "tag cloud". A tag cloud is a visual depiction of user-generated tags, used to describe the use and importance of a certain tag. It means that in addition to stored bookmarks information, every node has to be asked to return the list of all used tags. If some particular server is not responding, it will affect tag cloud visualization and tag 'importance' status. Finally, it will work slower than other models, because of different distance to physical server location that could affect system performance and decrease shared data discovery time.

2.2. *Pattern 2: Replicate Through Middleware or Managed Synchronization (Replicate)*

This system design derives from federated model pattern described previously. Basically, a replicated version of distributed system is a mirrored version of federated model described as *Pattern 1*. Figure 2 illustrates replication model design principles.

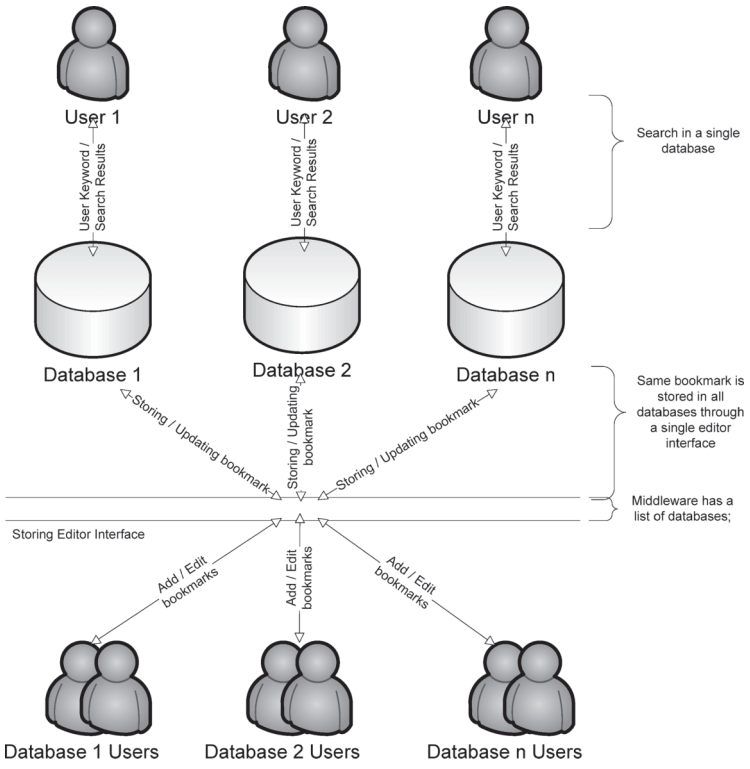

Fig. 2. Replicate through middleware model.

The same concept of middleware layer is taken, its role changed from “data harvester” to “data replicator”. Every user of the local system passes data to middleware layer that is responsible for delivering data to replicated version of the database, after performing action on data calls. Middleware layer has a list of replicated databases. After every user’s action, whether it is create, edit or delete, middleware is forming a package with new data and sending it to every copy of the database, updating its information. In the end, there are identical copies of databases in every system node. There is no more need for middleware, during the search action. Database replication allows searching only local node of the system that owns the same information as every other node. In this case, data discovery action is performed much faster, then in federated model. Besides, tag cloud construction is no more dependent on all nodes response.

However, this model faces the same problem of nodes response during create/edit process as in *Federate* pattern. If one of the nodes is not responding during add/edit action, it will not get bookmarks update information from middleware. This problem could be solved developing scheduled processes in middleware layer that is tracking down nodes state and sending those changed packages, when they start to respond. Nevertheless, this solution reduces the risk of non delivered data, but the problem still remains: if nodes are down for a longer period of time or additional changes are made to the particular data that is held re-sending. This may require additional synchronization algorithms in middleware layer, what would add additional complexity to this model.

2.3. *Pattern 3: Peer-To-Peer (Point-To-Point) Synchronization (Synchronize)*

Peer-to-peer synchronization model is based on peer processes architectural model and uses replicated database model as a back-end. This model does not require middleware layer to perform synchronization actions. Every node of this system could receive requests for services from other system nodes and respond to them. As well, every system node could request every other node on the net to ask for particular data. The only condition is that every node needs to have a mapping scheme of other nodes, where information has to be sent.

Figure 3 illustrates peer-to-peer synchronization model. The main advantage of this model is that it requires only small changes of existing social bookmarking software. Basically, create/edit functions as well as search are being performed the same as in single solution model. It means that this solution brings all advantages of non-distributed version, such as high data collection speed. The only change that needs to be done is change during information create or edit process. Every time when system user commits changes to local database, a package has to be formed and sent to every other node of the system. Basically, there is a replicated version of a database in every node. This kind of solution allows having a distributed solution of independent databases. It means that, if one of the nodes is not responding it is not affecting other nodes performances.

However, in this case replicated pattern disadvantage appears: if some of nodes are down during update time, it has to be tracked and update action has to be performed later. Besides the problem with unsent and edited data and change tracking remains in this

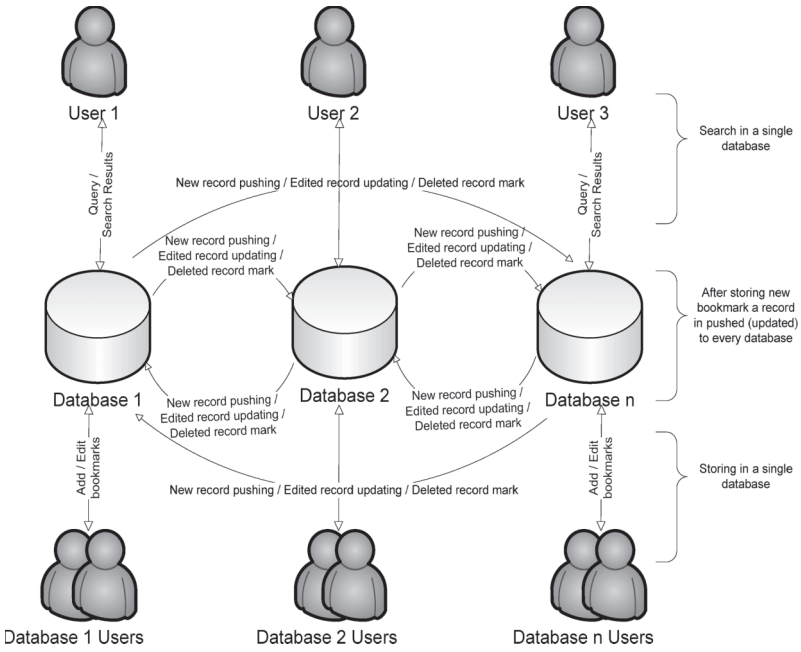

Fig. 3. Peer-to-peer synchronization model.

model too. The main model disadvantage could be named high data traffic, when a node has to update information in every other node. And this traffic is rising exponentially, adding every new node to the system.

2.4. *Pattern 4: Push/Pull and Aggregate*

This pattern is based on a client-server model, id est a particular case of client-server model, when every server node is using one defined server node as a data storage. This model is a case of both federated and replicated database models, so it lets to employ both models' advantages.

Figure 4 illustrates push/pull and aggregate model. This model has a set of independent databases that are used for store information. Every time a user creates/edits information, changes are committed in a local node of the system. After that, every change is committed to central database that is later used to get all information aggregating it from other nodes. Committed information could be either pushed to central database or pulled by the request of central database after getting 'change of status' message. This allows decreasing data replication in every separate node and avoids synchronization issue between distributed databases. In turn, central database provides collective information retrieval from every node, when local system user performs search action. User queries not a local database there, but central one. This model's quality of service depends on central database reliability, so a special attention has to be paid to insure its permanent

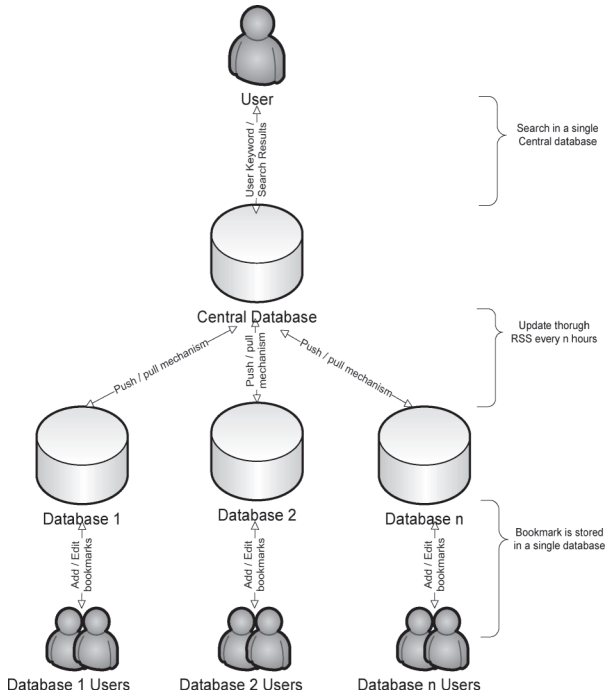

Fig. 4. Push/pull and aggregate model.

work. Local nodes do not affect the whole system work, what decreases whole system's dependability on particular nodes.

In a case when, central service is down for some reason, distinct nodes could change central database search to local one, what will increase the whole system's fault tolerance. Besides, such system does not need a middleware service, that will allow avoiding problems related to it.

2.5. *Evaluation and Solution Selection*

Designed patterns comparison and models overview show that every solution has its weaknesses and strengths. It is even more important to have in mind that distributed system is designed for social bookmarking service, what puts some limitations and raises problems that could be omitted in other vase. These results are shown in the Table 1. There are taken design requirements raised by George Coulouris (Couloris *et al.*, 2003), as a measuring system. The discussion about the security issue is excluded, because it mostly depends on coding implementation. By defining grades, total evaluation is calculated: $+/+-$ if there are more advantages and minor or none disadvantages, $+/--$ if the model has equally advantages and disadvantages, and $-/--$ if the model, obviously, has more disadvantages than advantages.

Table 1
Model comparison according to George Coulouris requirements

	Federate	Replicate	Synchronize	Pull/Push & aggregate
Performance issues	$+/-$	$+/-$	$+/-$	$+/+$
Quality of service	$+/-$	$+/+$	$+/+$	$+/+$
Use of cashing and replication	$+/+$	$+/-$	$+/-$	$+/-$
Dependency issues	$-/-$	$+/-$	$-/-$	$+/-$
Fault tolerance	$-/-$	$-/-$	$+/-$	$+/-$
Total	$+:4/-:6$	$+:5/-:5$	$+:5/-:5$	$+:7/-:3$

Performed research allows conduction that designed "Pull/Push& Aggregate" (Fig. 4) design is the most suitable model for applying it to develop distributed social bookmarking system.

It combines pluses of both federated and replicated database models, has high performance issues, provides good quality of service, avoids a high level of replication, is not dependable on every system node and is tolerant on faults. However, it is important to mention that a special attention has to be paid to central server development and its stability. Besides, it is necessary to implement fault tolerant algorithms that will help reduce damage, when central node is not responding.

3. System Implementation: SOAP vs. iCamp FeedBack

3.1. *SOAP Approach*

SOAP (Simple Object Access Protocol) is a lightweight XML based protocol for exchanging structured information between distributed applications over native web protocols, such as HTTP (SOAP Version Protocol). A SOAP protocol consists of three parts: an envelope which describes the contents of message; a set of rules for serializing data exchange between applications; a procedure to represent remote procedure calls (Tsenov, 2006).

That is how such architecture defines Tsenov (2006), distinguishing next steps. First web application makes a procedure call on the WSDL file and SOAL Service client. Then SOAP client takes data, builds XML container for them and sends it over HTTP as SOAP request. After getting SOAP message from the client, server parses that XML container and passes it to appropriate method, executes it and returns a result. This result is packaged to XML container and returned to the client over the POST request, which in turn parses returned result and decides what to do next.

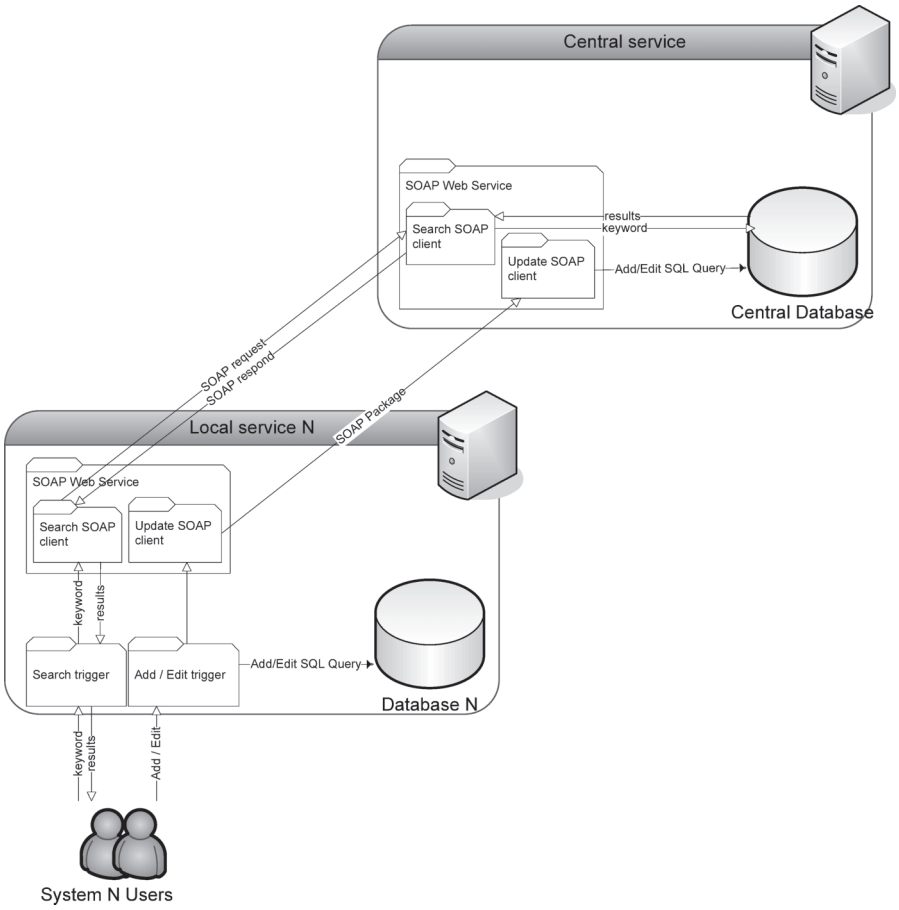

Fig. 5. Pull/push and aggregate model using SOAP web servers.

3.2. *FeedBack Approach*

There are three main steps when managing feeds with FeedBack (Wild *et al.*, 2007). First, central server has to be informed about opportunity to pull data from node server. Therefore, central server fetches node feed, parses it for xml-rpc endpoint, and calls feedback.offer() remote procedure.

Next, central server has to acknowledge whether it wants to get updates from node server, i.e., confirm the subscription. To ensure that next time it will be exactly the same client, central server generates security token and passes it back to node server, executing feedback.request (token) procedure. Finally, node server has to ping central server every time when updates are ready to be taken and node is ready to perform synchronization – using the feedback.notify(token) procedure. After getting ping, central server retrieves RSS feed with latest updates from the node.

Detailed specification of FeedBack communication process is presented in Fig. 7.

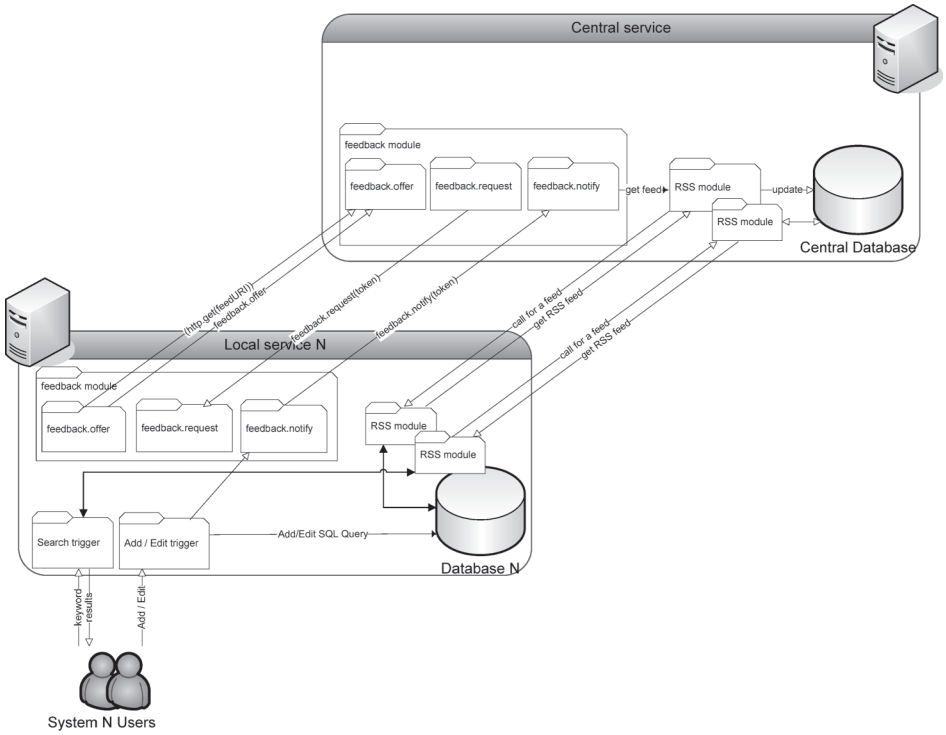

Fig. 6. Pull/push and aggregate model using FeedBack approach.

3.3. *Performance Evaluation*

To find out which implementation is more preferable we tested both implementations using real bookmarking data. To test SOAP implementation a 'SOAP for PHP' extension was selected. In turn FeedBack module is available on sourceforge.net portal (`http://sourceforge.net/projects/icamp/files`). The results are shown in the Table 2.

Table 2

SOAP and FeedBack evaluation

Number of records	SOAP, latency [ms]	FeedBack, latency [ms]
1	1.5	1.2
10	12	10
100	90	60
1000	800	500
10000	6000	2000

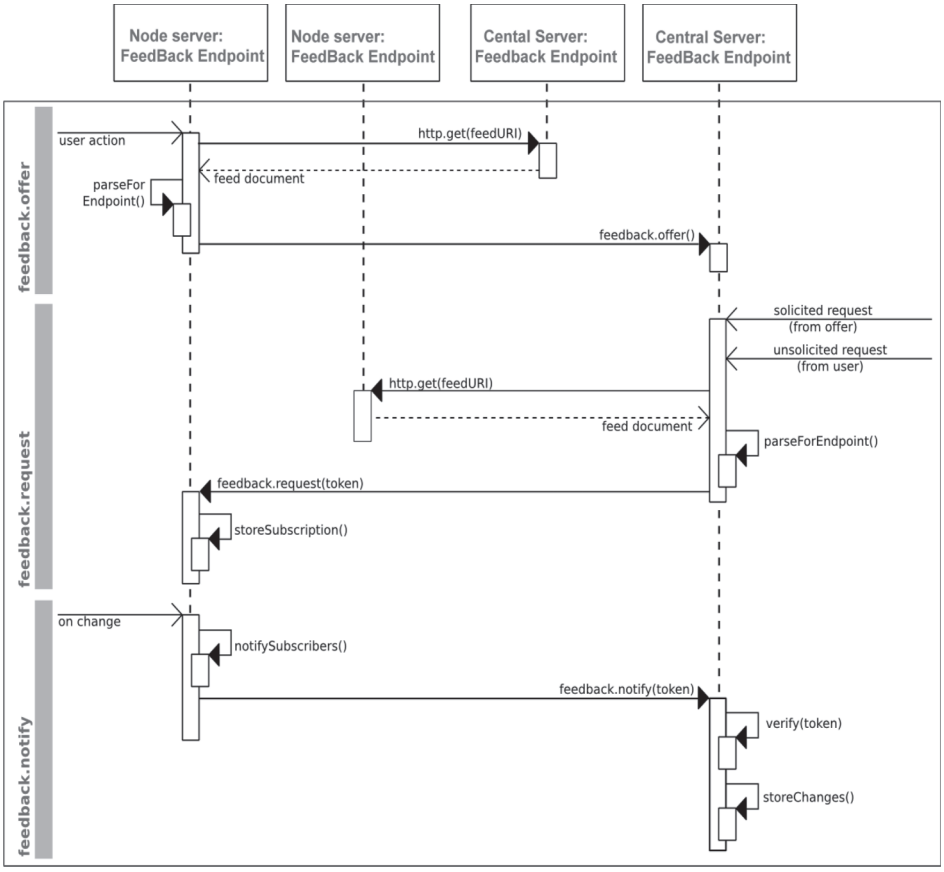

Fig. 7. The communication process in 'FeedBack' (Wild*et al.*, 2007).

As it was expected, iCamp FeedBack module showed better results transmitting various amount of data. SOAP web service delays part of working time wrapping data to SOAP envelope and performing additional tasks, defined by specification; in turn FeedBack works with pure RSS feed and spends time only for sending notification to central server. However, it is important to stress, that SOAP protocol ensures higher data delivery level because of its call back mechanism, which could re-transmit lost data in case of non-delivery.

4. Conclusions

Four different architectural models for building distributed social bookmarking systems were presented in this article, and they are: federated search through middleware model, replicated through middleware model, peer-to-peer synchronization model and pull/push and aggregate model. Considering basic requirements for distributed system

raised by George Coulouris and specific requirements, raised for social bookmarking system. Pull/push and aggregate model is most suitable for this particular case. Besides, the system was implemented using two approaches: SOAP specification and iCamp FeedBack module. Performance evaluation showed that FeedBack module is more suitable in this particular case. However, it is important to mention, that SOAP approach is more fault tolerant, because it has build-in mechanism to check if data transmission completed successfully.

Acknowledgements. I gratefully acknowledge the help of Fridolin Wild and Steinn Sigurdarson from Vienna University of Economics and Business.

References

Beverage, T. (2002). *Guide to Enterprise IT Architecture: A Strategic Aproach*. Secaucus, NJ, USA, Springer-Verlag New York.

Couloris, G. *et al.*. (2003). Distributed Systems: Concept and Design. 3rd edn. China Mashine Press.

Chiu, K., Govindaraju, M., Bramley, R. (2002). Investigating the limits of SOAP performance for scientific computing. In: *Proceedings of 11th IEEE International Symposium*, 246–254.
Davis, D., Parashar, M.P. (2002). Latency performance of SOAP implementations. Cluster computing and the grid. In: *The 2nd IEEE/ACM International Symposium*, 407–407, 21–24.

iCamp Codebase. `http://sourceforge.net/projects/icamp/files/`.

SOAP Version 1.2 Specification Assertions and Test Collection. `http://www.w3.org/TR/2003/REC-soap12-testcollection-20030624/`.

D'Souza, Q. (2006). Web 2.0 ideas for educators. A guide to RSS and more. Version 2.0. In: *Meeting of the K12*. Online, 2006.11.06. Available online at `http://www.teachinghacks.com/files//100ideasWeb2educators.pdf` [Accessed 2010.02.22].

Takeuchi, Y., Okamoto, T., Yokoyama, K., Matsuda, S. (2005). A differential-analysis approach for improving SOAP processing performance. In: *Proceedings IEEE'05 . International Conference on e-Technology, e-Commerce and e-Service*, 472–479.

Tsenov, M. (2006). Web services example with PHP/SOAP. In: *International Conference on Computer Systems and Technologies*. CSREA Press, USA.

Wild, F., Sigurðarson, S.E., Sobernig, S., Stahl, C., Soylu, A., Rebas, V., Górka, D., Danielewska-Tulecka, A., Tapiador, A. (2007). An interoperability infrastructure for distributed feed networks. In: *Proceedings of the 1st International Workshop on Collaborative Open Environments for Project-Centered Learning, COOPER-2007*. Sissi, Lassithi, Crete, Greece.

A. Afonin is seeking doctor degree at Kaunas University of Technology. He is a junior scientific employee at Research Laboratory of e-Learning Technologies, Department of Multimedia Engineering, Kaunas University of Technology, Lithuania. His research interests include web 2.0 technologies, distributed systems architecture, interoperability, development of e-learning software, personal learning environments and collective mind usage in educational software.

Paskirstytos socialinių nuorodų sistemos architektūra. SOAP prieš iCamp FeedBack

Andrej AFONIN

Straipsnyje aprašomas paskirstytos socialinių nuorodų sistemos projektavimas. Straipsnyje pateikiamos keturios tokios paskirstytos architektūros modeliai ir jų tolimesnė analizė. Taip pat nagrinėjami du skirtingi paskirstytų sistemų realizavimo būdai: pirmas, naudojant gerai žinomą *SOAP* protokolą ir *XML* formatą, ir antras, naudojant *iCamp FeedBack* mechanizmą ir *RSS* formatą. Straipsnio pabaigoje pateikiami abiejų būdų duomenų perdavimo analizės rezultatai.