Computers in Human Behavior 30 (2014) 485–490

Contents lists available at SciVerse ScienceDirect

Computers in Human Behavior

journal homepage: www.elsevier.com/locate/comphumbeh

# Automated and user involved data synchronization in collaborative e-health environments

M. Shamim Hossain [a,*], Mehedi Masud [b], Ghulam Muhammad [a], Majdi Rawashdeh [c], Mohammad Mehedi Hassan [a]

[a] College of Computer and Information Sciences, King Saud University, Saudi Arabia
[b] Computer Science Department, Taif University, Saudi Arabia
[c] EECS, University of Ottawa, Ottawa, Canada

ARTICLE INFO

*Article history:*
Available online 13 July 2013

*Keywords:*
e-Health
Collaborative environment
Data synchronization
User interaction

ABSTRACT

This paper presents a data synchronization model using automated and user involved process during execution of conflicting updates. Data synchronization is performed using three techniques, namely, (i) auto synchronization, (ii) semi-automatic synchronization, and (iii) user-involved synchronization. We have evaluated and measured users' acceptability of the proposed data synchronization approach in an e-health environment. The results show the effectiveness of the proposed approach.



## 1. Introduction

A collaborative e-health system provides an environment to enable collaborative treatments through sharing of data among the healthcare service providers (e.g., physicians, hospitals, laboratories, etc.). In such an environment service providers are autonomous, therefore, independently curate, revise, and extend the shared data. For full collaboration the service providers exchange data updates. Hence, data updates (insert, delete, or modify) in a service provider, i.e., data source may, in turn, affect the data in the other service providers.

Consider an e-health system, where family physicians, walk-in clinics, hospitals, medical laboratories, pharmacists, and other stakeholders are willing to share information about patients' treatments, medications, and test results. These sources need to coordinate their data of patients. Coordination may mean something as simple as propagating updates to each other. For example, a family physician and a hospital may want to coordinate the information about a particular patient. A hospital may store data about the patient's special treatment results while the patient was in the hospital. The family physician stores information about patient's regular treatments data. The family physician which is not storing special treatments data, would benefit by the exchange of updates made on the hospital data source. On the other hand, the hospital data source would benefit by the exchange of updates of any previous diagnosis results related to the patient in the family physician data source.

Exchanging data through update propagation in a collaborative e-health system is different from update processing in a traditional replicated database system (Masud, Kiringa, & Ural, 2009). First, a replicated database system assumes that the same data is replicated in different data sources to increase performance and availability of data. Meanwhile, the data stored in the sources in a collaborative e-health system is not replicated and may use different domains (Kementsietsidis et al., 2003). Second, an update originated in a source in a replicated system must be executed at all the sources to maintain consistency and ensure a single logical view of data throughout the network (Masud et al., 2009). In contrast, in a collaborative e-health system, an update may not need to execute at all the service providers. Rather, it is sufficient to execute the update only at the relevant sources to the update.

### 1.1. Contribution

This paper presents an approach for data synchronization where sources resolve conflicts in a collaborative fashion. Mainly data synchronization is performed using three approaches:

**Auto Synchronization (Auto-Sync):** In auto-sync, sources determine the execution order of conflicting updates for data synchronization through collaboration. In auto-sync, no user involvement is necessary to order conflicting updates. Auto-Sync synchronization is performed using the *absorption resolution* rule (Santoro, 2006).

* Corresponding author. Tel.: +966 14676189.
*E-mail addresses:* mshossain@ksu.edu.sa (M.S. Hossain), mmasud@tu.edu.sa (M. Masud), ghulam@ksu.edu.sa (G. Muhammad), majdi@mcrlab.uottawa.ca (M. Rawashdeh), mmhassan@ksu.edu.sa (M. Mehedi Hassan).

0747-5632/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.chb.2013.06.019

**Semi Automatic Synchronization (SemiAuto-Sync):** In this approach, data synchronization is performed through the collaboration of sources and with users' involvement. The SemiAuto-Sync approach is applied when two conflicting updates are executed into the same number of sources and the Auto-Sync approach fails to resolve the conflict. This synchronization is performed using *mutual resolution* rule.

**User-involved Synchronization (User-Sync):** In this approach users are directly involved to resolve a conflict of updates for data synchronization. Mainly, the conflicts are resolved by the super users. User-Sync approach is used in some critical situations where Auto-Sync process cannot reach a decision for resolving conflicts among updates.

In general, our proposed approach is applicable where systems tolerate data inconsistency for a certain period of time and update to a data in a source is not immediately important to the other sources. However, eventual consistency (Saito & Shapiro, 2005; Shapiro & Kemme, 2009) is guaranteed.

The next section presents notions of ensuring consistency and the approach of data synchronization. The notions are used to present the constraints for maintaining data consistency. Section 3 describes evaluation results. Section 4 describes the related works. Finally, Section 5 concludes the work remarking and pointing out avenues for further research.

## 2. Data synchronization

In a distributed e-health system updates are executed locally and independently. The system does not require a multi-site commit protocol (Hwang, Srivastava, & Li, 1994) (e.g., two phase commit), which leads to introduce blocking and is thus not feasible. Specifically, updates are executed locally, and then asynchronously propagated over acquainted sources. A consistent execution of updates can be obtained by ensuring the same execution order of the conflicting updates over each acquainted source in the propagation path of the updates. Authors in Masud and Kiringa (2011) describe the notion of consistent execution of conflicting updates for data synchronization. In the following, we propose an approach to achieve consistent execution of conflicting updates in automatic and user-involved process.

In order to maintain data consistency in the acquainted sources the execution order of the conflicting updates must be same in all the sources. If there is no conflict at the time updates are initiated in a source, then the updates can be executed in any order in the acquainted sources. Therefore, when two updates $\Delta_1$ and $\Delta_2$ are executed at a source and are not conflicting updates, then their different execution order in the acquainted sources of the source does not create any inconsistency.

An optimistic approach is used for executing updates in the data sources. Optimistic approaches allow continuous data access during update execution. They allow users to read or update the database while they are disconnected and synchronize the data with other sources when they reconnect (Kermarrec, Rowstron, & Shapiro, 2001; Petersen, Spreitzer, Terry, & Theimer, 1997; Terry, Theimer, Petersen, Demers, & Spreitzer, 1995). Here updates are propagated asynchronously in the background without blocking any read requests. Many major commercial database vendors support this mode of asynchronous replication. The steps to resolve conflict and synchronizing updates are given below.

*Step 1*: When a source detects a conflict between two updates that are received from other sources then the source stops execution of the updates and asks its parent about the execution order. Note that a source which forwards an update is a parent and the source which receives the update is called a child.

*Step 2*: If the parent has no information of execution, then the parent asks its parent. This process continues until a source is found that knows the order or the inquiry reaches to the update initiators. The first source which detected the conflict of the same pair of updates may have already propagated an inquiry to the initiators and the execution order is already decided. Hence, other sources which detect the same conflict may receive the result from any intermediate sources along the path to the initiator.

Now, we describe the process of selecting the execution order of two conflicting updates by the initiators or by an intermediate source who has the order information. An order is selected using three resolution protocols. The protocols are *mutual resolution*, *absorption resolution* and *user-involved resolution*. The mutual resolution is an automated process, the absorption resolution is a semi-automated process, and the third one is user-involved process.

*Mutual Resolution:* From the conflict message each initiator knows how many sources have executed the updates since the conflict message has travelled through a path from the sources where the conflict is detected to the initiators. We treat this count as execution level of an update. The execution level of an update $\Delta_i$ is denoted as $level(\Delta_i)$. If $(level(\Delta_1) = level(\Delta_2))$ then the order is determined with mutual understanding. After receiving the conflict information, initiators agree on an specific order. Consider the order $\langle \Delta_2, \Delta_1 \rangle$ is selected. After selecting the order the following process starts.

A compensate update $\Delta_1^-$ is generated and is sent to $S_i$. When $S_i$ receives the order information and the compensate update, $S_i$ execute $\Delta_1^-$ and execute the updates $\Delta_1$ and $\Delta_2$ in the order $\langle \Delta_2, \Delta_1 \rangle$.

*Absorption Resolution:* After receiving the conflict information, both the initiators agree on an order $\langle \Delta_2, \Delta_1 \rangle$ if level $(\Delta_2) > level(\Delta_1)$ else the order is $\langle \Delta_2, \Delta_1 \rangle$. The compensation process starts as described in mutual resolution.

*User-involved Resolution:* If a conflict is detected at source $S_i$ and $S_j$, the super users in the sources decide the update execution order. The users are informed about a conflict when a conflict is detected by the system. If the users cannot decide the order then the users consult with the users of the parents of the updates' propagator. After the conflict is resolved the compensation process starts as described in mutual resolution.

## 3. Evaluation

For evaluating the proposed data synchronization approach we measure the feasibility, efficiency, and users' acceptability of the system. Each data source is populated with a few hundred different relational tuples. In order to generate data, we wrote a simulator using Java. We choose integers value for the domain of the attributes in each relation. Acquaintances are built based on data items stored in each source. Each update is either insertion, deletion, or replacement of some tuple in a source. We consider SQL for update operations. All experiments were performed at least five times and we took the average of these measurements.

### 3.1. System evaluation

We evaluated the proposed update synchronization approach considering different conflict factors of updates. A *conflict factor* denotes the ratio of the number of updates that are involved in conflict and the total number of updates that are active in the system. For example, 0.2 conflict factor means that 20% of the total updates that are active in the system are involved in conflict. Note that updates are generated from different sources. The objective of this evaluation is to examine what is the effect of the conflict