Exploiting Cloud Computing and Web Services to Achieve Data Consistency, Availability, and Partition Tolerance in the Large-Scale Pervasive Systems

https://doi.org/10.3991/ijim.v15i15.22517

Ashraf Ahmed Fadelelmoula(✉)
Prince Sattam Bin Abdulaziz University, Alkharj, Saudi Arabia
asahfaab@gmail.com

Abstract—This article presents a new comprehensive approach to realize a sufficient trade-off between the CAP properties (i.e., consistency, availability, and partition tolerance) in the large-scale pervasive information systems. To achieve these critical properties, the capabilities of both cloud computing and web services were exploited in developing the components of the proposed approach. These components include a cloud-based replication architecture for ensuring high data availability and achieving partition tolerance, a web services-based middleware for maintaining the eventual consistency, and a data caching scheme to enable the mobile computing elements to conduct update transactions during the disconnection periods. The evaluation of the performance aspects revealed that the proposed approach is able to achieve a load balance, lower propagation delay, and higher cache hit ratio, as compared to other baseline approaches.

Keywords—CAP properties, cloud computing, data replication, distributed systems, pervasive information systems, web services

1 Introduction

The recent decade has witnessed a tremendous spread of large-scale distributed systems (LSDSs) in a wide variety of industries. The crucial factors behind this spread include the rapid advancements in the wide area network technologies and computational resources, the intensive use of networks for running a broad diversity of applications, and the fast development of the modern industry [1–3]. Categories of these systems include large-scale distributed computing systems (e.g., grid and cloud computing systems), large-scale distributed information systems, and pervasive systems (e.g., mobile computing systems and sensor networks) [4,5].

The characteristics of LSDSs can be represented into several dimensions, including the size and heterogeneity. With respect to their size, LSDSs consist of a large number (usually millions) of highly geographically dispersed nodes. The heterogeneity characteristic indicates that LSDSs are heterogeneous in nature at the levels of hardware, software, and network technologies [1]. In addition to these dimensions, data in LSDSs are shared extensively over wide areas and associated with intensive concurrent read/write operations.

These characteristics make the data management in LSDSs more challenging than in the ordinary systems. An important aspect of this challenge is developing scalable data management solutions that enable reliable and fast access to massively distributed data over wide areas. One feasible solution to provide such access is through implementing the data replication. Substantially, data replication is a widely used data management technique for decades in many systems [6]. It is defined as the process of maintaining multiple copies of data objects, called replicas, on separate sites [7,8]. The mostly cited benefits of data replication include increasing data availability, enhancing reliability, and improving performance.

Despite of these benefits, there are many issues associated with the data replication. A major issue is the impossibility of achieving strong consistency along with other two desirable properties of replication systems, which are availability and partitioning tolerance. This impossibility was indicated by the CAP theorem, which states that it is not possible for a distributed system with data replication to guarantee more than two of these three properties (i.e., consistency, availability, and partition tolerance) at the same time [9]. In this theorem, consistency implies that all replicas of a data item should have the same value (i.e., state) at the end of each update operation. Availability indicates that the read/update requests for data items can be processed successfully. Partition tolerance encompasses the ability of the system to continue operating even if a network fault resulting in several partitions occurs [9,10].

To respond to the impossibility consequence of this theorem, a sufficient trade-off between these properties is required. This involves finding weaker consistency guarantees that contribute to enabling both availability and high performance [11]. Many systems (e.g., Amazon's Dynamo [12]) have implemented a relaxed consistency guarantee (i.e., other than strong consistency) called eventual consistency, which its idea encompasses performing local updates on each replica and afterward propagating these updates to other replicas asynchronously to reach an eventual convergence [13]. Accordingly, eventual consistency allows the replicas of a data item to temporarily diverge, as long as they eventually converge to a global consistent state (i.e., having the same value) [14–16].

However, there are several challenges should be addressed for eventual consistency, especially when the number of replicas increases. These include providing fast dissemination (i.e., propagation) of updates between the data item's replicas to eliminate their state divergences, resolving update conflicts consistently, and ensuring that updates are implemented in the same order at each replica.

This paper handles these issues in an important class of LSDSs, which is the Large-scale Pervasive Information Systems (LSPISs). This class represents a hybrid of both large-scale distributed information systems and pervasive systems. It is identified here as a category of systems consisting of a large number of diverse pervasive computing elements, massive data, and heterogeneous networks that are distributed over wide areas. In addition to the general characteristics of LSDSs, LSPISs have specific traits, including the popularity of using mobile computing elements, sensors, and actuators for conducting a huge number of distributed transactions, the commonality of wireless communications, and the wide diffusion of computing elements in diverse environments to sense, manipulate, and store data [5,17,18]. Examples of these systems include the pervasive variants of the healthcare information systems [19,20], environmental

monitoring systems [21], warehousing and logistics management systems [17], traffic control systems [22,23], crimes recording systems, and financial transaction processing systems. Despite their importance and commonality in vital industries, LSPISs are often overlooked in research studies devoted to handling replication issues in the information systems area.

There are intrinsic limitations associated with the popularity of wireless networks and mobile computing elements in LSPISs, including poor bandwidth, communication latency, unreliability and unavailability of the wireless connectivity in many areas, frequent disconnections, and limited battery duration [24–27]. These limitations aggravate the aforementioned replication issues by hindering the realization of a sufficient trade-off between the CAP properties and the achievement of the overall system scalability. Consequently, effective replication solutions considering these aspects are highly needed in LSPISs. To act in accordance to this need, this paper focuses on proposing a scalable replication approach that makes a sufficient trade-off pertaining to the achievement of the three desirable CAP properties in LSPISs. As such, the specific objectives of the study are:

1. Develop a scalable replication architecture that provides a robust basis for improving the availability of read/write operations and achieving partition tolerance in LSPISs.
2. Propose a service-oriented middleware to conduct and automate the essential operations of the replication process, including the automation of updates propagation between the replicas in the replication architecture in a manner that maintains the eventual consistency of the replicated data.
3. Propose a method for detecting and resolving update conflicts that occur due to the concurrent write operations on many replicas.
4. Develop a data caching scheme for mobile computing elements that enables them to conduct update operations during the disconnection times.

To achieve these objectives, the proposed replication approach exploits the capabilities of both cloud computing and web services.

2 Background and related work

This section provides a background to the categories of the data replication approaches, and presents the related work.

2.1 Data replication approaches

Replication is a crucial process for effectively support shared data in a variety of distributed environments (e.g., grid, mobile, cloud, and P2P environments) [25,28–30]. It intends to provide several positive outcomes in these environments, including improve data availability, enhance fault tolerance, minimize bandwidth consumption, reduce data access latency, and decrease data transfer cost and time [28,31,32,54]. There are many replication approaches attempted for such environments. These approaches can be classified into synchronous and asynchronous replication solutions.

In the synchronous replication (also called as eager replication), the updates occur on one replica are immediately propagated to the other replicas before the completion of the transaction that performed these updates (i.e., the update transaction). Accordingly, the updates are implemented at each replica as a part of a single transaction. When this transaction commits, all the replicas will have the same value. This makes the synchronous replication an effective mechanism for providing a strong consistency guarantee. However, updating all the replicas before the termination of the update transaction negatively affects the transaction's response time, and consequently hinders the applicability of this replication mechanism in environments consisting of a large number of replicas [10,33]. Moreover, it requires a reliable communication between replicas. Thus, this mechanism cannot act in accordance to the characteristics of LSPISs, especially the commonality of unreliable wireless connectivity and frequent disconnections of mobile computing elements, as well as it cannot cope adequately with the scalability requirement of these systems.

A widely cited synchronous replication approach is the read-one/write-all (ROWA) protocol, which requires the execution of the write operation on all the replicas of a data item and performing the read operation on any replica [34].

In contrast, in the asynchronous replication (also known as lazy replication), the updates are propagated to the other replicas sometime after the update transaction commits. This propagation occurs through independent transactions, generally known as refresh transactions. The main advantage of this replication mechanism is that it has lower response times for update transactions, because an update transaction can immediately commit once it has updated one replica [10]. Additionally, it improves the availability in systems having unreliable communication, increases the throughput, and enhances the overall system performance. Hence, it is widely used as a scalable replication solution in the distributed environments [35,36]. However, the delay in updating the other replicas leads to inconsistencies, and thus inhibits this replication mechanism from providing a strong consistency guarantee [25,37,38]. Consequently, a weak guarantee is provided by such mechanism, which is the eventual consistency.

Therefore, most of the existing asynchronous replication approaches tend to perform the tradeoff between the CAP properties through decreasing consistency for improving availability and partition tolerance.

2.2 Related work

Representatives of asynchronous replication approaches in a variety of large-scale distributed environments are presented next. This variety is mainly due to the lack of approaches devoted specifically to LSPISs. More focus is given to the mobile environments because they represent a foundation for running the pervasive information systems.

In a mobile environment, the Cedar system [39] focuses on trading off consistency in order to enable mobile data access (i.e., availability) with high performance over wide-area networks. It adopts a simple client/server based asynchronous approach in which the updates occur on the server (i.e., hosting the master copy) are implemented on the clients at infrequent intervals. It relies on the usage of the stale client replicas for improving data access, and accordingly achieving such trade-off. However, keeping the master replica on

a certain node (i.e., the server) hinders the implementation of this approach in large-scale distributed systems having massive number of updatable replicas.

By concentration on providing a scalable asynchronous replication strategy for improving data availability in mobile environments, a system called ROAM was proposed with a characteristic that any replica can serve operation requests [40,41]. In this system, nearby replicas are grouped into domains called wards. The members of each ward can directly synchronize and communicate with each other in a P2P fashion. Each ward has a specific member designated as a ward leader that is responsible for maintaining consistency with the other wards. The updates propagation occurs within each ward (i.e., between its replicas) and among wards (i.e., between ward leaders) through a ring topology. Although ROAM seeks to achieve high scalability, the mechanisms of handling and committing large numbers of updates on highly distributed replicas were not addressed in details in this system.

Mohana and Jaykumar [42] proposed a cluster-based hierarchical replication scheme for mobile database systems. The hierarchical structure of this scheme consists of a database server, cluster heads, and cluster members (i.e., clients). The server sends the replicated data to the cluster heads to respond to the queries issued by the members. The heads can also fetch the desired data for satisfying the queries from each other. The updated data in each cluster are sent to the server through the heads. With respect to maintaining consistency, a detailed procedure for handling enormous concurrent updates on the same data items in different clusters is needed in this scheme.

Bsoul et al. [43] considered a grid distributed environment to implement a hierarchy-based replication strategy. In this strategy, the network structure consists of several regions. Each region has a header and a set of nodes that are located closely. On the top of the region headers, there is a master site in which the replicas are stored and further distributed to the headers. The region header ensures that the replica requests are satisfied in its region. The node can request a replica from the other nodes in the same region. The methods of handling the replica updates and maintaining the consistency among the replicas were not discussed in this strategy.

Tos [6] proposed a replication strategy that concerns on several aspects in cloud computing environments, including the satisfaction of the query performance. It considers a cloud environment that comprises a set of geographical regions with each region encompassing a set of data centers, which each of them in turn contains a number of servers. These servers hold the replicas of the data fragments based on a placement heuristic. In this strategy, a minimum number of replicas is maintained to fulfill a least availability required level for the data fragments. Updating these replicas and maintaining their consistency were not included in the scope of this strategy. Moreover, it limits its focus to dealing with queries pertaining to OLAP applications.

In a similar vein, the proposed strategy of Limam et al. [29] considered a hierarchical cloud topology for supporting data replication. It aims at calculating the minimum number of replicas needed to realize a high data availability. A new replica is created only when this minimum number is not reached or when the response time goal is not achieved. The limitations of this strategy include its focus on the replication of read only data. Accordingly, it is used for OLAP purposes.

In sum, the extant asynchronous replication approaches in the various distributed environments are not dealing well with the characteristics of LSPISs, especially the

carrying out of a massive number of concurrent update transactions by a wide variety of distributed computing elements, including mobile clients. Accordingly, new replication solutions that adequately cope with the traits of these systems while perceiving a reasonable trade-off between the CAP properties are highly needed.

3 The proposed data replication approach

To cope well with the characteristics of the LSPISs and realize a sufficient trade-off between the CAP properties, the components of the proposed approach are specified as follows. The first component is a scalable replication architecture to provide a robust foundation for improving data availability and achieving partition tolerance in the LSPISs. The cloud computing concepts were employed to determine the specifications of this architecture and enable fast access to the replicated data. The second component is a web services based service-oriented middleware to perform and automate the crucial operations of the replication process. The third component is a method for detecting and resolving update conflicts that occur due to the concurrent write operations on multiple replicas. The last component is a data caching scheme to enable the mobile computing elements to conduct update operations during the disconnection periods.

3.1 The replication architecture

The replication model of the LSPISs is structured in our proposed approach as a scalene triangle that includes a variety of computing elements from an internal cloud and one element from a public cloud (see Figure 1). Accordingly, the triangle encompasses a hybrid of both internal and public clouds. The internal cloud consists of two main categories of computing elements, which are local servers and update sources (i.e., located in the triangle's right and bottom sides, respectively). Both categories are physically distributed among wide areas. An additional category that includes specialized servers is found in the left side. Regarding the public cloud, the triangle includes only the needed computing element from this cloud (i.e., in its upper vertex), and referred to here as external server. The roles of these elements are provided next.

- **Local servers:** They are placed in one or more internal data centers, which are administrated by the owning organization. They are the only computing elements in the internal cloud that hold data replicas, thereby they are called here as replica servers. Each of these servers receives read/update requests from many update sources in the replication architecture. Consequently, all transactions in the entire replication system are committed in the replicas hosted in these servers. The exact content of any of these servers is a set of replicated data domains. The data domain is the replication unit of the proposed approach, and is defined as a collection of data that can be changed by a large number of dispersed update sources. Accordingly, it can be a data file, a database table, or a part of such units. Every domain is distinguished by a unique identifier and type, and is represented as an ordered triple: *D(Identifier, Type, Data)*.

- **Update sources:** This category encompasses a variety of computing elements, including fixed and mobile elements (e.g., user clients, sensors, and actuators). They are called here update sources because one of their major functions is to access the data replicas to generate update requests. The mobile elements can issue update requests only upon their connections with the replica servers, while their fixed peers can request updates at any time due to their reliable connectivity with these servers. Some of the mobile elements (i.e., mobile clients) need to have a cached copy of the replicated data in order to be able to perform local tentative updates during the disconnection periods. Once the connection takes place, these elements request the replica servers to commit their tentative updates permanently (i.e., making the changes permanent on the copies stored in these servers). This is because the only updatable copies admitted by the proposed approach are those stored in the replica servers. For mobile sensors, they do not need to store a cached copy for conducting tentative updates. This stems from their nature as merely data generators. Hence, their produced data represent stable updates.
- **External server:** Each of the replica servers propagates the updates committed on its replica to the external server in the public cloud. The purpose of using this server is to maximize the availability of the recent updates to all replicas. Its exact responsibilities include receiving the updates committed in each replica in the internal cloud, reconciling the received updates through conducting the conflicts detection and resolution processes, and propagating the reconciled updates to all replica servers for reaching the eventual consistency in that cloud. Accordingly, the updates occurred in any internal cloud's replica are distributed to the other replicas through the external server. This server holds a replica for only performing these responsibilities. Therefore, it does not have any interactions with the update sources in the internal cloud in terms of receiving updates on its replica or committing them. These interactions are undertaken locally in the internal cloud by the replica servers. Accordingly the update transactions processing is performed reliably on the internal cloud, while the updates reconciliation process is conducted by an element (i.e., the external server) from a certified public cloud. This indicates that the update transactions are only carried out in the internal cloud.
- **Specialized servers:** The data center in the left side of the triangle contains three servers with certain functionalities. The first one is responsible for managing the admission process in the replication system. It registers the details of: (a) the replicas and their distribution among the ordinary data centers in the right side of the triangle and (b) the computing elements that are authorized to update these replicas (i.e., the update sources). Hence, before joining the system, each computing element should initially contact this server in order to be admitted as an authorized node for interacting with the replicated data (i.e., performing read and update operations). Also, this server is contacted upon needing to get an updated list of the replicas. Moreover, it is contacted by the external server to obtain the details of the new replicas. It is called here as an admission server. The second server is acting as a backup facility for periodically maintaining a recent copy of the entire replicated data hosted in the external server. The last server includes analytical tools (e.g., OLAP and data mining packages) for analyzing vast amounts of replicated data, and providing the stakeholders of the owning organization with valuable

insights and predictions for better decision making. It regularly interacts with the external server for obtaining the most recent data to be analyzed.

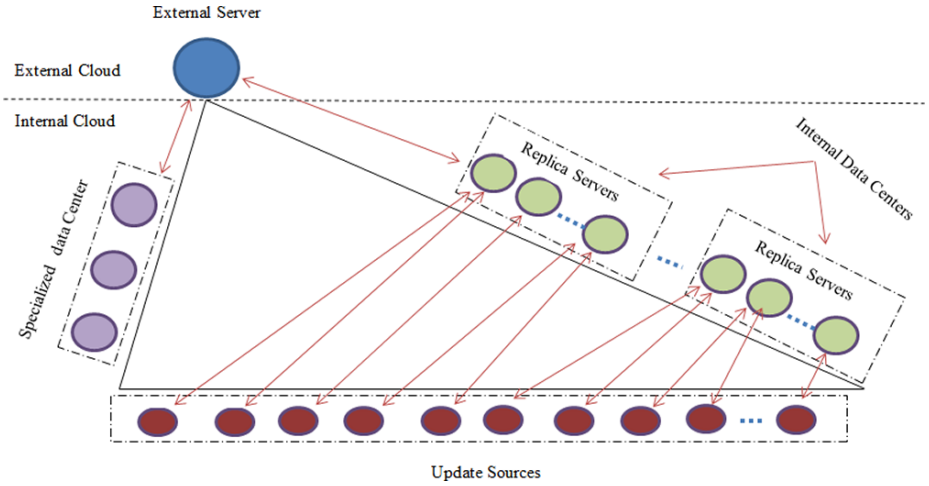

Fig. 1. The replication architecture

For very large-scale systems, the proposed replication architecture can be extended by adding one or more triangles. This results in having multiple external servers in the hybrid cloud. In this case, one of the these servers will be elected for reconciling the recent updates received by it and its counterparts in the public cloud, and then propagating the reconciled updates to them in a P2P fashion. In turn, each peer propagates these updates to its underlying replica servers in the internal cloud in order to be available for the transaction requesters.

In this architecture, the replica servers are the only computing resources along the path from the update sources (e.g., user clients, sensors) to the external cloud server. Each of these servers represents an edge in the internal cloud that eliminates the needs of the update sources to access the external server. This is because the various read/write requests are satisfied through these servers. Thus, the crucial computation pertaining to updating and retrieving data is performed on the level of the edge of the internal cloud network. Therefore, such type of processing comprises the core concept of the edge computing, which represents a form of the cloud computing. Augmenting this concept through conducting the essential computation in the internal cloud aims to reach several consequences, including communication costs reduction, fast responses to read/write requests, load balance realization, and better response to the LSPISs' challenges.

3.2 The web services-based service-oriented middleware

The proposed replication architecture is augmented by specifying a service-oriented middleware for managing the crucial replication operations in the LSPISs and

facilitating the interactions (i.e., communications) between the heterogeneous components of these systems. The middleware consists of a set of web services that enable the automation and rapid performance of the essential operations of the replication system, including the requisition, execution, reconciliation, and propagation of updates. Each computing element in the replication architecture has its own web service (see Figure 2). The crucial categories of these services are those reside in the update sources, the replica servers, the external server, and the admission server. They are denoted as US-WS, RS-WS, ES-WS, AS-WS, respectively. For instance, US-WS indicates the web service that resides in an update source. The vital roles of these categories are as follows.

- **US-WS:** It enables its update source to request update transactions on the replicated data domains hosted on the replica servers. It generates the details of the update request (such as *request-ID*, *update-type*, and *requester-ID*) and sends them to the RS-WS of the replica server.
- **RS-WS:** It executes the update requests, reconciles the updates received from the mobile computing elements (i.e., those conducted during the disconnection periods), and propagates all updates received within a certain time interval to the external server.
- **ES-WS:** It performs a global reconciliation for all updates received from the replica servers, and propagates a set of recent reconciled updates to these servers (i.e., updates propagation from the external cloud to the internal one).
- **AS-WS:** It handles the replicas registration process, and provides the web services of the other computing elements with a list of active replicas. Moreover, it attaches a web service to each new computing element joining the system based on its type (e.g., replica server or update source).

As such, these web services characterize the replication architecture as a service-oriented architecture. This is because the aforementioned operations of the replication system are provided as services by several providers, which are the web services of the computing elements involved in the replication process. The provider of the updates requisition service is the US-WS. The provider of the updates execution (i.e., conducting the updates on a replica) service is the RS-WS. Both updates reconciliation and propagation services are provided on different levels by the RS-WS and ES-WS. Thus, in this architecture, both the requestors and providers of a replication service are web services that reside in heterogeneous computing elements.

With respect to the implementation of these services, the core standards of the web services mechanism, which are the Web Services Description Language (WSDL), the Simple Object Access Protocol (SOAP), and the Universal Description, Discovery, and Integration (UDDI) protocol [44,45,55,56], are utilized in the proposed approach as follows. The functionalities of these services (i.e., their operations) are described using the WSDL, which represents an XML-based standard for describing the web services. The excerpt given in Figure 3 shows the usage of WSDL to define the update request operation (i.e., provided by the US-WS). As shown, the <portType> element of the WSDL document has been used to define one operation with request and response messages (i.e., input and output messages).

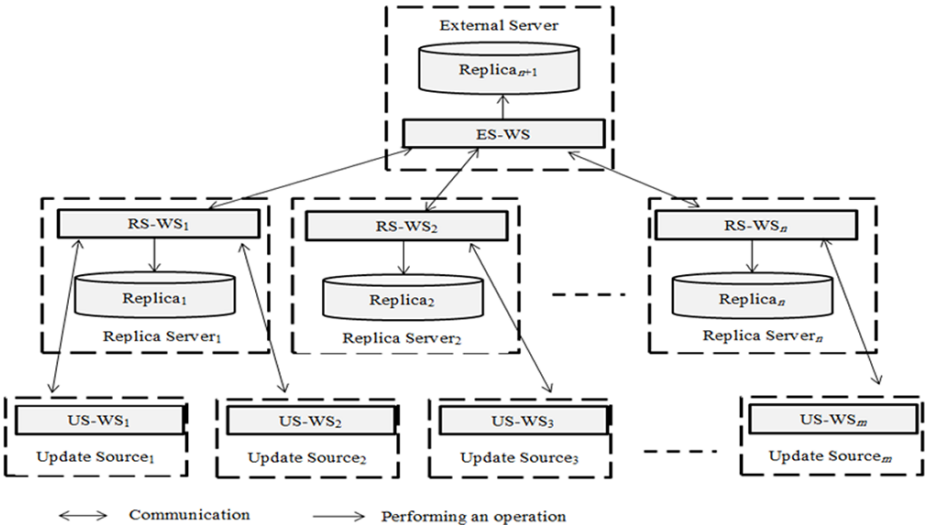

Fig. 2. The web services-based service-oriented middleware

```
<wsdl :portType name="UpdateRequestPortType">
    <wsdl :operation name="UpdateRequest" parameterOrder="RequestID
    UpdateType UpdateStatement UpdateTimestamp…">
        <wsdl :input message="tns : DataDomianUpdateRequest " />
        <wsdl :output message="tns : DataDomianUpdateResponse " />
        …
    </wsdl :operation>
</wsdl :portType>
```

Fig. 3. An excerpt that shows the usage of WSDL to define the update request operation

The interaction between these web services occurs through using the SOAP, which represents an XML-based messaging protocol for transferring data [46]. The SOAP is generally recommended for distributed environments. It supports two different communication patterns: Remote Procedure Call (RPC) and message-oriented (or document). The message-oriented pattern is widely used in the modern SOAP engines as the default communication style [47]. Accordingly, the proposed strategy relies on using the message-oriented pattern of the SOAP communication to enable the interactions among the identified web services. A sample of this usage is sending a SOAP message for updates requisition (i.e., US-WS sends the update details to a RS-WS). As shown in Figure 4, this SOAP message is XML-formatted. The essential element in this message is the body one, which encompasses the details of the update request. These details are represented as XML data. The optional element of this message (i.e., the header one) contains data that aid in handling the message.

As such, using the SOAP, each web service can communicate with the others, with the exception that US-WS and ES-WS do not have direct communication with each other.

With respect to the flow of the SOAP messages (i.e., the exchange style), for updates propagation, these messages are sent in two directions as follows. A RS-WS propagates the reconciled updates in its replica server through these messages to the ES-WS in a bottom-up fashion, while the ES-WS follows the top-down direction to propagate the universally reconciled updates to the RS-WSs. For updates requisition and queries, the SOAP messages are sent from the US-WSs to the RS-WSs according to the request/response messaging style, which requires sending a reply back to the requester (i.e., in a form of acknowledgment or data retrieval).

```
<E:Envelope ...>
  <E:Header>
    <t:Transaction
       xmlns:E="TheSpecifiedTransactionURI"
       E:mustUnderstand="1">
     </t:Transaction>
    …
   </E:Header>
  <E:Body>
    <UR:UpdateRequestDetails>
      <UP: RequestID > Auto-Generated-ID </ UP: RequestID >
      <UP: UpdateType > Insert </ UP: UpdateType >
      <UP: UpdateStatement > Statement-Syntax </ UP: UpdateStatement >
      <UP: AffectedDomain > Data-Domain-ID </ UP: AffectedDomain >
      <UP: RequestTimestamp > System-Time </ UP: RequestTimestamp >
       …
   </ UR:UpdateRequestDetails >
  </E:Body>
</E:Envelope>
```

Fig. 4. A sample SOAP message format for updates requisition

The registration and locating of these web services are enabled using a local registry in the internal cloud. This registry acts as the one provided by the UDDI standard, which is implemented for setting up a service registry [44]. As such, all services recorded in this local registry are belonging to one provider (i.e., the owning organization of the internal cloud). In its essence, this registry represents a directory in the admission server that facilitates the handling and discovery of the currently registered web services in the replication system, registering the specifications of new services, and providing the new computing elements joining the system with the web services they need to conduct the replication operations. Respecting the case that a new computing element joins the system, the content of this directory are searched by the AS-WS in order to decide which web service should be reside in the new member (e.g., replica server or update source). Then, it attaches the relevant service to this member.

Each of these web services can be implemented as a class that comprises a set of methods. The essential methods associated with these services include

Generate-Update-Request (), Detect-Conflicted-Update-Requests (), Execute-Update (), Reconcile-Conflicted-Update-Requests (), Propagate-Updates-To-Higher-Level (), Query-Satisfaction (), Perform-Universal-Reconciliation (), and Propagate-Updates-To-Lower-Level ().

The steps of the crucial operations of the web services: The crucial operations conducted by the web services in the proposed middleware encompass those pertaining to the updates request generation and execution. The steps that are carried out for performing these operations are provided in Figure 5.

In this object, the *Execution-indication* attribute holds the value "*True"* in the case that the update was executed in a replica server, and "*False"* otherwise.

Step1. Upon generating an update request, the US-WS conducts the following:
1.1 Set the values of the attributes of the update request, including Request-ID, Update-Type, Update-Statement, Affected-data-Domain-ID, Requester-Update-Source-ID, Recipient-Replica-Server-ID, and Request-Send-Timestamp. / These attributes are stored on an object called Update-Requests-Object (URO) */*
1.2 Send the request to the RS-WS of the destination replica server.

Step2. Upon receiving the request, the RS-WS performs the following actions:
2.1 Set the value of the Request-Received-By-Replica-Server-Timestamp attribute in the URO.
2.2 Retrieve the request details written by the US-WS from the URO.
2.3 Execute the request on the affected data domain.
2.4 Set the value of the Execution- indication attribute in the URO to "True".
2.5 Send a snapshot of the results of the executed request to the requesting US-WS.

Fig. 5. The steps of the updates request generation and execution processes

The schema of the *URO* is given in Figure 6.

Object Name	Attributes
URO	*Request-ID, Update-Type, Affected-data-Domain-ID, Requester-Update-Source-ID, Recipient-Replica-Server-ID, Request-Send-Timestamp, Request-Received-By-Replica-Server-Timestamp, Execution-indication*

Fig. 6. *URO* schema

Another crucial operation of the web services in the replication system is supporting the dynamic change of the used replicas. The proposed approach implies that the number of replicas in each internal data center is dynamically changed on the basis of the *Access Pattern* (*AP*), which is classified for each replica as *very very low* (*VVL*), *very low* (*VL*), *low* (*L*), *medium* (*M*), *high* (*H*), *very high* (*VH*), and *very very high* (*VVH*) access. These *APs* are determined based on the *Average Number of Accesses* (*ANA*) that occurred in all replicas during a *Prefixed Time Period* (i.e., pre-decided by the admission server), which is denoted by *PTP*. The values of *APs* range from 0 to 0.14× ANA for *very very low*, …, 0.86× *ANA* to 100 for *very very high*. The focus in our explanation will be on the two exceptional *APs*, which are the minimum (*very very*

low) and maximum (*very very high*) ones. The steps of calculating the *ANA* and using it to dynamically change the number of replicas are given in Figure 7. The involved web services in these steps are RS-WS and AS-WS.

Step1. Calculate the ANA
1.1 When PTP is over, each RS-WS reports the number of accesses that occurred in its server during this period to the AS-WS.
1.2 The AS-WS calculates the ANA and disseminates the result to all RS-WSs.

Step2. Each RS-WS determines the AP on its server based on the received ANA and decides the required actions as follows.
2.1 In the case that AP is VVL, it sends a request to AS-WS to change the status of its server to Inactive. / removing it temporarily from the list of servers that accept requests */*
2.2 In the case that AP is VVH:
2.2.1 It sends a request to AS-WS to balance the load.
2.2.2 The AS-WS will activate one of the inactive replica servers to share the load of the heavy-loaded server, and inform the requesting RS-WS.
2.2.3 The RS-WS of the heavy-loaded server will forward the upcoming requests to its peer in the activated server.
2.2.4 In the case that no inactive server, the AS-WS will nominate a server with a VL pattern to its peer in the heavy-loaded server.
2.2.5 The step 2.2.4 will be repeated for any further request until all servers reach the VL pattern. In this case, the AS-WS will issue an alarm indicating that a new replica server is needed.

Fig. 7. The steps of calculating the ANA and its usage

The steps of the operations pertaining to the updates conflict detection and resolution as well as updates propagation are presented in the next sub-section.

3.3 The conflicts detection and resolution method

Maintaining the eventual consistency among all replicas involves conducting the updates reconciliation process, which encompasses two sub-processes, called update conflicts detection and resolution. These sub-processes are carried out on both the internal and external clouds. In the internal cloud, each replica server performs these processes on the updates that it receives from its underlying update sources, and then propagates a set of reconciled updates to the external server. In turn, this server performs these processes on all updates received from the underlying replica servers, and propagates the totally reconciled updates to them.

The conflicts detection sub-process relies on the timestamp attribute of the received updates. The value of this attribute is assigned immediately to each generated update based on the global system time. Maintaining this unified time is not an issue in the replica servers and external server due to the reliable communications among them. But, for mobile computing elements, such maintaining cannot be guaranteed due to their frequent disconnections. Consequently, the proposed approach includes the steps shown in Figure 8 to ensure the matching of the update timestamps in these elements with the global system time:

Step1 At the beginning of each connection session with a replica server, the web service of the mobile computing element checks the accuracy of the local time by comparing it with the global system time available in the replica server.
Step2 If a difference is found, the web service will correct the timestamps of all updates occurred since the last connection with a replica server.

Fig. 8. The steps of matching update timestamps

The following equation is used for the correctness process based on the difference from the system time:

$$CUT = IUT \pm D$$

Where *CUT*≡ Corrected Update Timestamp, *IUT*≡ Incorrect Update Timestamp, *D*≡ Difference between the system time and local time of the mobile computing element, + sign is used when the difference is negative (i.e., the local time of the element is less than the system time) and the – sign is used otherwise.

This equation acts as a method for enforcing the assignment of the update timestamps on the basis of the global time, and accordingly ensuring a unified fair timing for all updates occurred in the replication system. The corrected update timestamp is used by the RS-WS and further the ES-WS in detecting and resolving updates conflicts. It is necessary to include that the implementation of this equation by the fixed update sources is not required because they perform the updates directly on the replica servers, which always maintain the accurate global system time.

In sum, the proposed approach relies on the update timestamps, which are assigned based on the global time, for conflicts detection and updates ordering. The web services in the mobile computing elements are responsible for ensuring the accuracy of these timestamps by maintaining the same global time as in the replica servers.

With respect to the conflicts resolution sub-process, it is conducted in this approach based on the types of the conflicted update operations (i.e., insert, delete, and modify). The insert operation cannot conflict with the modify and delete operations and vice versa. Therefore, if the insert operation shares the same timestamp with one of these operations, the resolution will be implemented on the basis of several measures, including their send-times to the higher level (i.e., a replica server or external server) and arrival-times at this level. As such, the resolution of such case is implemented in terms of updates reordering process.

If a delete operation conflicts with a modify one on the same portion of a data domain, the latter is given a priority to be executed in the higher level, because it indicates the validity of the modified data, and accordingly the former is aborted. In this case, a notification is sent to the deletion requester indicating the validity of the data to be deleted to other update sources. To handle the case of receiving a modify operation on a deleted data (i.e., the timestamp of the delete operation is less than its counterpart for the modify operation), such data are not removed permanently. Instead, they are kept in an object (i.e., acts as a recycle bin) for a temporary period that is specified according to the application requirements. As such, the upcoming modify operation can restore the validity of these data during the reconciliation interval.

If two modify operations on the same data domain are conflicted (i.e., having the same timestamp), the resolution can be conducted by assigning priority levels to

the types of the modifications. These levels are determined based on the business rules that govern the data validity. A simple example is that deducting a health insurance premium from an employee gross pay can occur only after registering him in the service. In this case, the registration modification is given a higher priority level than the deduction one (i.e., will be executed before the latter).

To conduct the reconciliation process, the external server implements the concept of the *reconciliation time interval* (*RTI*) for each data domain, which is denoted by $RTI_{external\text{-}server}$ and defined here as a regular time interval that the web service of the external server (i.e., ES-WS) must wait before ordering and propagating the received updates on the data domain to replica servers. This is in order to ensure that all recent updates carried out during $RTI_{external\text{-}server}$ are received, which enables mobile clients to catch this interval. The length of *RTI* varies among the data domains based on the *Inconsistency Time Period* (*ITP*) affordable by each data domain, which is specified according to the application requirements.

As such, for each data domain, these intervals have an equal length, indicating that $\text{Length}(RTI_{external\text{-}server-i}(x)) = ITPx$, where $i = 1, \ldots, n$, and n is total number of *RTIs* of a data domain x. They are numbered serially on the basis of the day (i.e., 24 hours), such as *Domain1-RTI-SUN-02-march-20-1*, *Domain1-RTI-SUN-02-march-20-2*, etc. Upon elapsing of each interval and propagating the reconciled updates to the replica servers, the divergence between the replicas will be bounded and both read(x) and write(x) operations will interact with consistent data, where x is a replicated data domain. To clarify this point, the sequence of $RTI_{external\text{-}server}$ intervals can be represented mathematically by the following function:

$$\rho\ (i,date) = RTI_{external\text{-}server-i}^{Date},\ 1 <= i <= 24/\ RTI_{external\text{-}server}$$

Each item in this sequence corresponds to a convergence status for the replicated data domain in the replica servers. This indicates that the serial elapsing of the intervals in this sequence contributes to reaching consecutive convergences to consistent states by all replicas (i.e., the eventual consistency). As such, the following limit is assured by the behavior of this sequence:

$$\text{Lim}\ (RTI_{external\text{-}server-i}^{Date}) = \text{eventual consistency}$$

If a set of recent updates belong to the current interval (e.g., *Domain1-RTI-SUN-02-march-20-10*) have been reconciled and propagated to the replica servers, and some updates belonging to this interval have come on a next interval (e.g., *Domain1-RTI-SUN-02-march-20-11*) due to a late propagation from mobile clients, then the web service will implement these updates on its replica of the data domain, and propagate a correctness report along with the reconciled updates of the *Domain1-RTI-SUN-02-march-20-11* to the replica servers. The purpose of this report is to inform the replica servers to correct the execution of the pervious received updates on the basis of the timestamps of the late received updates.

The same notion of *RTI* is applied at the level of the replica servers. All these servers implement $RTI_{replica\text{-}server}$ for each data domain to propagate the recent updates to the external server. Two conditions are identified for this interval as follows. The first condition is that all replica servers should have the same $RTI_{replica\text{-}server}$ for each data domain in order to enable consistent propagation of the updates that occur on the domain.

The second condition is that the logical expression $RTI_{replica\text{-}server} < RTI_{external\text{-}server}$ should be true in order to ensure a fast propagation to the external server where the ultimate reconciliation will be done. Another reason for this condition is that the updates performed on a replica server are always less than those received by the external server.

The conceptual object schemas pertaining to the reconciliation process include the ones depicted in Figure 9.

Object Name	Attributes
Reconciliation-Intervals-Object	*Data-Domain-ID, $RTI_{external\text{-}server}$, $RTI_{replica\text{-}server}$*
Reconciliation-Interval-Limits-Object	*Interval-Code*, Data-Domain-ID, Interval-Type-ID**, Upper-Limit, Lower-Limit, Interval-ID-Within-24Hours**** Notes: **Interval-Code* is added to link this object with the *Reconciliation-Details-Object*. ***Interval-Type-ID* is either 1 for $RTI_{external\text{-}server}$ or 2 for $RTI_{replica\text{-}server}$. ****Interval-ID-Within-24Hours* varies from 1 to 24/ $RTI_{replica\text{-}server}$. For example if $RTI_{replica\text{-}server}$ is 2 hours, there will be 12 intervals of this type.
Reconciliation-Details-Object	*Reconciliation-ID, Server-Type-ID*, Server-ID, Interval-Code, Date* Note: **Server-Type-ID* is 1 for an external server and 2 for a replica server.
*Current-$RTI_{external\text{-}server}$-Received-Updates-Object**	*Interval-Code, Data-Domain-ID, Update-ID, Update-Type, Update-Timestamp, Updates-Source-ID, Replica-Server-ID, Late-Received-Update-Flag*** Notes: *Once the current interval is elapsed, the contents of this object are transferred to the *Data-Domain-Updates-Tracking-Object* for permanent saving. ***Late-Received-Update-Flag* is either 0 for an update occurred during the current reconciliation interval or 1 for one occurred during a past interval.
Data-Domain-Updates-Tracking-Object	*Interval-Code, Data-Domain-ID, Update-ID, Update-Type, Update-Timestamp, Updates-Source-ID, Replica-Server-ID, Late-Received-Update-Flag*

Fig. 9. Object schemas pertaining to the reconciliation process

Reconciliation steps in the external server: The steps shown in Figure 10 are carried out by the web service of the external server for reconciling the updates received from the replica servers.

Step1. Start a new reconciliation session
1.1 Retrieve the $RTI_{external\text{-}server}$ *from the Reconciliation-Intervals-Object*
1.2 Retrieve the Lower-Limit and Upper-Limit from the Reconciliation-Interval-Limits-Object
1.3 Insert the details of the current reconciliation in the Reconciliation-Details-Object

Step2. Upon receiving an update from a replica server during the current $RTI_{external\text{-}server}$
2.1 Determine the affected data domain
2.2 Insert the details of the update in the Current- $RTI_{external\text{-}server}$ *-Received-Updates-Object*
2.3 If the timestamp (update) within the current $RTI_{external\text{-}server}$ *then*
2.3.1 Mark this update as a current by setting the value of the Late-Received-Update-Flag attribute to 0

Else
2.3.2 Mark this update as an old by setting the value of the Late-Received-Update-Flag attribute to 1

Step3. Upon elapsing of $RTI_{external\text{-}server}$
3.1 Detect the conflicts of the current updates
3.2 If there are any conflicts then
3.2.1 Resolve the conflicts based on the time-stamp values

Else
3.2.3 Next step
3.3 Propagate the reconciled updates to the replica servers
3.4 Transfer the details of the reconciled current updates from the Current- $RTI_{external\text{-}server}$ *– Received-Updates-Object to the Data-Domain-Updates-Tracking-Object*
3.5 Detect the conflicts of the old updates against those stored in the Data-Domain-Updates-Tracking-Object
3.6 Perform step 3.2
3.7 Send the correctness report to the replica servers
3.8 Transfer the details of the reconciled old updates from the Current- $RTI_{external\text{-}server}$ *– Received-Updates-Object to the Data-Domain-Updates-Tracking-Object*
3.9 Remove the contents of the Current- $RTI_{external\text{-}server}$ *-Received-Updates-Object to save storage space*

Fig. 10. The steps of updates reconciliation

With minor modifications, this algorithm is applied on the level of the replica servers to reconcile the received updates from their sources before propagating them to the external server. These modifications include replacing $RTI_{external\text{-}server}$ with $RTI_{replica\text{-}server}$ and making the external server as the destination of the propagated reconciled updates.

3.4 The caching scheme

Data caching is essentially required for mobile computing elements in order to continue functioning in the disconnected modes and tolerate the network partitioning. As such, the proposed approach enables each mobile client to cache a recent copy of its data domain of interest and perform tentative updates on it in the disconnected mode. When it connects to a replica server, its web service performs the following functions: (a) captures all updates that occurred in the cached data domain since the last connection with a replica server (i.e., represent tentative updates), and (b) sends a request to the replica server for the permanent commitment of these updates. Hence, the cached copy

of the data domain in the mobile client is treated as a virtual replica that can receive tentative updates, which should be committed later on its corresponding domain in the replica server.

Accordingly, the purpose of this caching is only to record the tentative updates occurred on the cached data domain during the disconnection periods (i.e., tracking these updates). Such updates will become stable upon their commitment in a replica server. This is because the only updatable copies in the replication architecture are those stored on the replica servers. Hence, the tentative updates of the mobile client are not considered in the whole replication system unless they are committed in a replica server.

The validity of the cached data domain in the mobile client (i.e., its consistency with its peer in a replica server) is verified by the web service (i.e., US-WS) at each connection with a replica server. This verification involves comparing the *Timestamp of the Last Connection* (*LCT*) with a replica server with the *Timestamp of the Last Update* (*LUT*) that occurred on that data domain. *LCT* is stored in an object called *Connections_Tracking_Object* in the mobile client, while *LUT* is maintained in another one called *Data_Items_Updates_Tracking_Object* in the replica server. The schemas of these objects are depicted in Figure 11.

Object Name	Attributes
Connections_Tracking_Object	*Mobile-Client-ID, Replica-Server-ID, LCT*
Data_Items_Updates_Tracking_Object	*Data-Domain-ID, LUT*

Fig. 11. The schemas of the objects that contain the timestamp attributes

The *Data_Items_Updates_Tracking_Object* stores only one tuple for each data domain. This tuple is frequently replaced by another one when the temporal data item (i.e., *LUT*) is changed. At each connection with a mobile client, the RS-WS makes this tuple available to the US-WS for checking the validity of the cached data domain (i.e., comparing the value of *LUT* from this tuple with the value of *LCT* from the *Connections_Tracking_Object*). The system time of the replica servers is considered for tracking both *LCT* and *LUT*. This is because a unified global timing can be maintained easily in these servers as they represent a part of a fixed network. On the other hand, whenever the mobile clients disconnect from these servers, their local system times may change and become inconsistent with the unified global time.

If the cached data domain is valid (i.e., *LCT* > *LUT*), the US-WS will request the RS-WS to commit the tentative updates on the replica server. Otherwise, the US-WS will conduct the steps depicted in Figure 12.

Step1. Update the cached data domain based on the current state of its peer in the replica server. This update is done by importing such state (i.e., the current data) from the replica server.
Step2. Perform the tentative updates again on the updated cached data domain. / This is because they were previously performed on the invalid cached data domain*/*
Step3. Send a request to the RS-WS to commit these updates.

Fig. 12. The steps carried out in the case that the data item is invalid

Accordingly, the requests to commit the tentative updates are sent to the replica server only if the cached data domain is valid.

An obvious characteristic of our caching scheme is that the replica server does not broadcast an invalidation report (i.e., contains timestamps and recent modifications) to the mobile client. This is in contrast to the previous schemes that rely on periodically broadcasting such report from the server to the mobile clients for maintaining the consistency of the cached data [38,48]. Instead, the invalidation report broadcasting in our caching scheme is replaced by the invalidation verification, which is performed by the US-WS of the mobile client through comparing the temporal details in the *Connections_Tracking_Object* and *Data_Items_Updates_Tracking_Object* objects (i.e., *LCT* and *LUT*) upon connecting to a replica server. Based on this comparison, the US-WS will import only the current state of the data domain from the replica server in the case that *LUT* exceeds *LCT*. This reduces the overheads associated with the frequent broadcasting of the invalidation reports (e.g., communication costs and bandwidth consumption).

Another characteristic is that the cached data domain can receive updates that make it more recent than its peer in the replica server. As such, the request sent by the US-WS to the RS-WS for committing these updates can be thought as an invalidation method that indicates the invalidity of the current state of the corresponding data domain in the replica server (i.e., it is not up-to-date).

4 Performance aspects

This section presents the assessment of widely used performance metrics for evaluating replication approaches.

4.1 Average load balance (ALB) for the web services

Distributing the load among the web services participating in the replication system is a crucial aspect for augmenting the overall system's performance. Such distribution is assessed here using the ALB metric, which is identified as the average number of computing elements that the web service may interact with in order to accomplish a certain process in the replication system. As such, this metric can have the following instances: ALB-Propagation (i.e., the ALB for the propagation process), ALB-Requisition (i.e., the ALB for the updates/queries requisition process), ALB-Requests-Satisfaction (i.e., the ALB for the update/query requests satisfaction process). Table 1 includes the values of these instances along with their justifications. These values indicate that the average load is distributed among the web services of the servers in both internal and external clouds.

Table 1. ALB instances and their values

ALB Instance	Involved WS	Value	Justification
ALB-Propagation	RS-WS	1	Each RS-WS propagates its updates to the external server
	ES-WS	*N*	ES-WS propagates the overall reconciled updates to *N* replica servers
ALB- Requisition	US-WS	1	Each US-WS sends a query/update request to one replica server
ALB-Requests-Satisfaction	RS-WS	*M*	Each RS-WS satisfies the requests of *M* update sources

4.2 Update propagation delay (UPD)

The updates propagation represents a vital process in the replication system for achieving the eventual consistency. It is evaluated here on the basis of the UPD metric, which is measured based on the total number of hops required for propagating an update from a replica server to another one. This is because measuring the exact time that is consumed in updates propagation depends on many complicated factors, including connectivity aspects (e.g., bandwidth and network delays) [49]. In our strategy, the total number of hops equals 1. This is because the update committed in each replica server will be propagated to the others through the external server. For the other strategies, this total number varies according to their propagation protocol.

For instance, in Roam [40,41], propagating an update from a replica R_i to a replica R_j in a different ward involves using a ring topology, and requires: (a) R_i sends the update to its ward master, (b) R_i's ward master forwards the update to R_j's ward master, and (c) R_j's ward master forwards the update to R_j. Accordingly, for such case, the minimum number of hops in Roam is 2. However, this number will vary as the number of wards changes.

In the tree-based hierarchical strategy [50,51], the updates propagation relies on an *N*-ary tree. The root of this tree represents the owner of the data and the other nodes (i.e., residing in the tree's levels) store replicas of these data. Based on the tree characteristics, a top-down updates propagation is assumed, indicating that the updates are propagated to these replicas through following the tree's levels. As such, the number of hops required to propagate an update from the tree's root to all replicas in the last level equals L - 2, where L is the total number of the levels in the tree. Hence, the number of hops increases as the number of levels increases. In sum, the proposed strategy enables each replica to receive the recent updates in at most one hop. Thus, as compared to other strategies, this implies that it has the lowest updates propagation delay.

4.3 Performance metrics for evaluating the proposed caching scheme

Two widely performance metrics in the literature of the data caching were used to compare the proposed caching scheme with two baseline approaches (i.e., pull and push-based approaches). These metrics are number of hops and cache hit ratio.

Number of hops: The first comparison with the pull and push-based approaches was performed by considering the average number of hops required to validate the cached data item in the mobile clients. In the pull-based approach, which is called aggregate cache based on demand (ACOD) scheme, when a query request is generated, the mobile client broadcasts the request packet to the server or other mobile clients (i.e., closer to the server) for validating its cached data item before using it for satisfying the request. In the push-based approach, which is called modified timestamp (MTS) scheme, the server periodically broadcasts an invalidation report to the mobile clients. After the mobile client receives this report, it validates its cached data item accordingly, and forwards the report to the nearby clients [52].

In our proposed caching scheme, the number of hops is fixed, which equals one. This is because the cached data domain is validated when connecting to a replica server. Adversely, in the baseline approaches, this number may vary due to the possibility of having multiple hops between the client and the server. Figure 13(a) summarizes the average number of hops for the three schemes, and indicates that the proposed caching scheme has the lowest average. The average number of hops for the two baselines approaches was calculated based on the results of Lim et al. [52], which are pertaining to effects of multiple factors on the number of hops.

Regarding the effect of the cache size on the number of hops, this number is not impacted by varying the cache size in the proposed caching scheme (i.e., its value remains fixed as 1). In contrast, it varies in the baseline caching schemes according to the changes occur in the size. Figure 13(b) shows that the proposed caching scheme has the minimum number of hops. For the two baseline schemes, the number of hops was included in this figure according to the results of Lim et al. [52]. The cache size was measured by the number of the cached data items in the mobile client, which varies from 10 to 100.

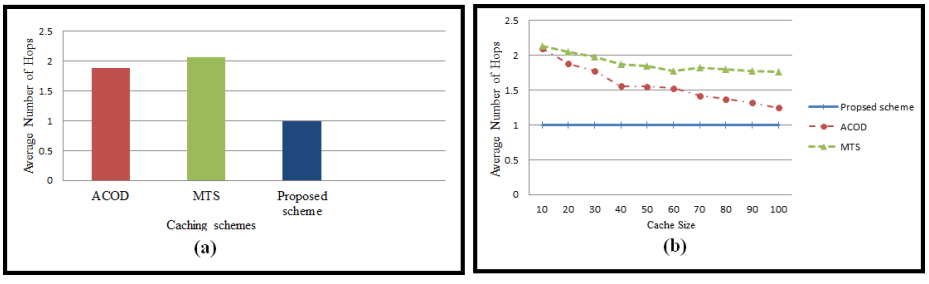

Fig. 13. (a) The average number of hops for three schemes. (b) The effects of the cache size on the average number of hops

Likewise, the number of hops of the proposed scheme is not influenced by varying the number of mobile nodes in the network. This contradicts the results of many studies, such as [53]. Figure 14 depicts that the caching approach of [53], which is called Adaptive Cooperative Caching Strategy (ACCS), has different hop counts for different numbers of mobile nodes, while the proposed scheme has a fixed count (i.e., one). The values included in this figure are obtained from the results of [53], which were produced using a fixed cache size (i.e., 1600 KB).

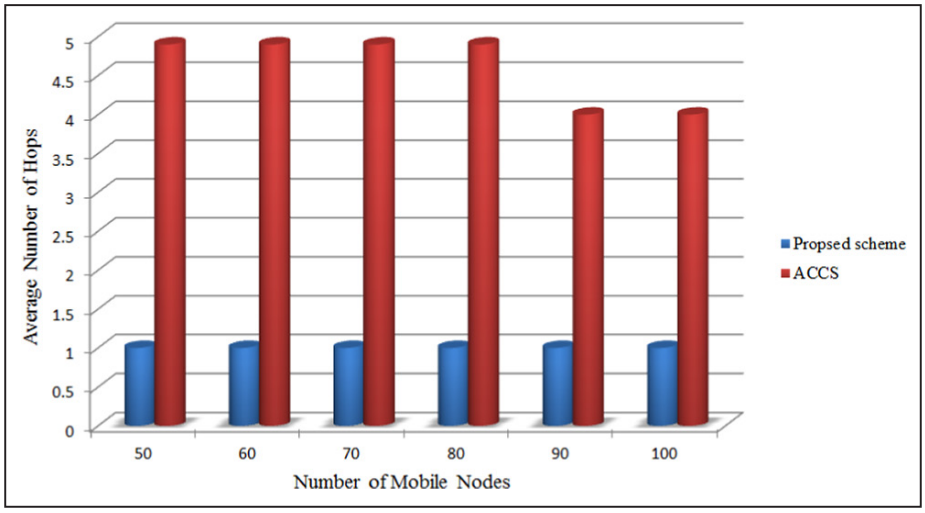

Fig. 14. The effects of the number of mobile nodes on the average number of hops

Cache hit ratio: The second comparison with the two baseline approaches was conducted on the basis of cache hit ratio, which can be decomposed into local and remote ratios. With respect to the local cache hit ratio of the proposed caching scheme (i.e., the ratio that an update request is satisfied locally in a mobile client), all requests of tentative updates are fully fulfilled in the mobile client via exploiting its local cached data domains. This indicates that the hit ratio is 100% (i.e., the maximum ratio) for such requests. On the other hand, the requests for committing these updates permanently are completely satisfied in a replica server. This satisfaction can be regarded as equivalent to the remote cache hit. Accordingly, the ratio of such hit is 100% for the requests pertaining to the permanent commitment of tentative updates. In the baseline caching approaches as well as the others, the requests are of the same type (i.e., satisfying certain queries), and both local and remote hit ratios for these requests may vary. Figure 15(a) and Figure 15(b) depict such variation as well as the fixed ratios of the proposed scheme. These figures were produced through considering the effects of the mean cache update interval on these ratios and setting the cache size to 100 (i.e., as included in the results of Lim et al. [52]). As shown, the local and remote hit ratios of the proposed scheme are not affected by the variation of the update interval.

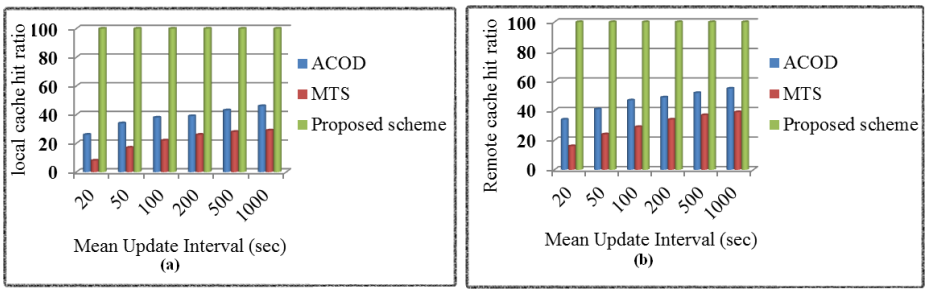

Fig. 15. (a) Local cache hit ratio and (b) Remote cache hit ratio by considering the effects of the mean update interval

Regarding the effect of the cache size on the cache hit, both the local and remote hit ratios (LHR and RHR) of the proposed scheme were compared to three types of hits identified by [57]. These types are local cache hit (i.e., requests are serviced locally), cache node hit (i.e., requests are satisfied by intermediate nodes), and server hit (i.e., the server satisfies the requests). Figure 16 (i.e., produced based on the results of [57]) shows that these types are affected according to the variation of the cache size (i.e., varied between 100 KB and 600 KB). However, for the proposed scheme, both the local and remote hit ratios remain fixed (i.e., 100%). As aforementioned, this is due to having two types of requests: one should be satisfied locally (i.e., tentative update) and the other must be remotely fulfilled by a server (i.e., permanent commitment).

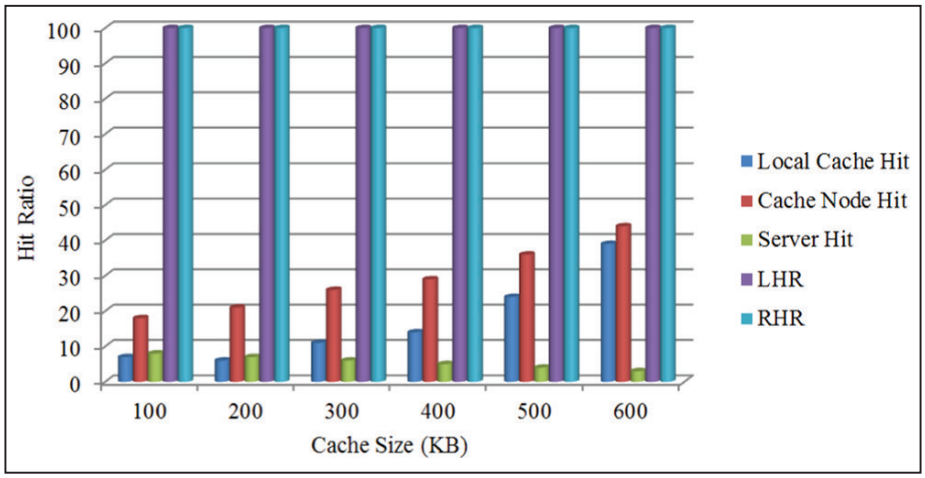

Fig. 16. Comparing the local and remote hit ratios (LHR and RHR) of the proposed scheme with those of the three types of hits identified by [57]

Similarly, the hit ratios of the proposed scheme are not affected by varying the number of mobile nodes. This is in contrast to many other caching approaches, including ACCS [53] and ACCC [58].

5 Discussion

Towards achieving a sufficient trade-off between the CAP properties in the LSPISs, the components of the proposed approach have been specified as follows. The first component is a replication architecture that combines elements from both internal and external clouds for ensuring high availability of read/write operations and achieving partition tolerance in the LSPISs. The second component is a web services based service-oriented middleware to perform and automate the crucial operations of the replication process, such as automating the updates propagation between the replicas in a manner that maintains the eventual consistency. The third component is a method for updates conflicts detection and resolution. The last component is a data caching scheme to enable the mobile computing elements to conduct update operations during the disconnection periods.

The proposed approach performs two main actions to make such trade-off. The first one is allowing the update transactions to be performed on many replicas in the internal cloud. This contributes to achieving both data availability and fault tolerance. The second one is ensuring the availability of recent updates to all replicas via exploiting the capabilities of the external cloud. As such, this action contributes to achieving the eventual consistency. Accordingly, the proposed approach focuses on relaxing one of CAP attributes, which is the consistency, in order to realize a great deal of such trade-off. This relaxing is conducted by adopting the eventual consistency instead of the strong consistency, which cannot be achieved in the presence of distributed updatable replicas.

Some of the features and characteristics of the major components of the proposed approach are outlined as follows. The replication architecture component, which comprises a set of cloud elements, supports the system scalability in terms of accommodating the future changes easily, such as covering more areas and encompassing vast numbers of new heterogeneous computing elements. This feature is highly needed in the large-scale systems as one of their design goals is developing scalable architectures that contribute to realizing higher availability levels for the system. The service-oriented middleware component acts as an operating system for the proposed approach by conducting a set of essential replication operations, including replicas registration, updates request, update conflicts detection and resolution, and updates propagation. To support the scalability of the replication system, this middleware does not enforce any restrictions about the quantity and type of the data being replicated, indicating the ability to deal with a large number of diverse data domains. This stems from the cooperative work of the middleware's web services, which facilitates the carrying out of the aforementioned replication operations on these data, regardless of their quantities and types.

With respect to the caching scheme component, the widely used cache invalidation scheme (i.e., broadcasting of invalidation reports) in the previous caching schemes has been replaced here by a verification process that is performed by the web service of the mobile client upon connecting to a replica server. This process involves comparing temporal data items, and importing only the current state of the data domain from the replica server in the case that the cached copy in the mobile client is invalid. As such, this process reduces the downlink traffic (i.e., the transferred data from a replica server to a mobile client) by eliminating the need to

broadcast periodic or on-demand invalidation reports. Regarding the uplink traffic (i.e., the transferred data from a mobile client to a replica server), it is reduced by sending only requests to the replica server to implement the same updates that occurred in the mobile client. Hence, the proposed scheme contributes to the reduction of the bandwidth usage.

The evaluation of the performance aspects revealed that the proposed approach achieves a load balance in conducting crucial processes in the replication system, including updates propagation and requests satisfaction. This balance has been realized through distributing the load among the web services of the servers in both internal and external clouds. The requests satisfaction is the responsibility of RS-WSs, while the updates propagation is conducted by these services and ES-WS. With a focus on the updates propagation process, the proposed approach achieves lower propagation delay than other replication strategies. Such delay was represented by the total number of hops required to propagate an update from a replica to another one.

Regarding the effectiveness of the proposed caching scheme, a comparison with several existing approaches was conducted. The results indicated that the proposed scheme is having a minimum fixed number of hops (i.e., one). In this regard, Saleh [53] included that the reduction of the energy consumption and the request satisfaction delay requires that the number of hops between the source and the destination of the request to be as small as possible. Also, the comparison characterized the proposed scheme as having maximum local and remote hit ratios. Accordingly, it highly improves the local satisfaction of the tentative update's requests in the mobile clients. The maximum value of the remote hit ratio is not avoidable because the requests for permanent commitment of updates should be fulfilled by a server.

6 Conclusion

This article has proposed an asynchronous replication approach to realize a sufficient trade-off between the CAP properties (i.e., consistency, availability, and partition tolerance) in the large-scale pervasive information systems. An obvious uniqueness of this research effort is that the proposed approach is having four new scalable components (i.e., replication architecture, service-oriented middleware, update conflicts detection and resolution method, and data caching scheme). The previous approaches are limited in having only some of these crucial components. Another uniqueness dimension is that the proposed approach exploits the capabilities of both cloud computing and web services to realize the trade-off among the CAP properties. The cloud computing concepts were applied to develop the scalable replication architecture for improving the availability and partition tolerance. The web services were implemented to facilitate the interactions and shared processes among the heterogeneous computing elements of the replication architecture, including supporting updates propagation and data caching. Exploiting these two web-based technological solutions has a significant role in advancing the design approaches for large-scale systems. This is because both of them support the desirable characteristics of large-scale systems, including scalability and flexibility. Such exploitation represents a uniqueness aspect of this paper in both data replication and pervasive system areas.

The future work encompasses developing the required tools to implement the components of the proposed approach and testing them in a practical environment of a large-scale pervasive system.

7 References

[1] A. Salkenov and S. Bagchi, "Cloud based autonomous monitoring and administration of heterogeneous distributed systems using mobile agents," *Future Generation Computer Systems*, vol. 99, pp. 527–557, 2019. https://doi.org/10.1016/j.future.2019.04.047

[2] S. Ghosh, Distributed systems: an algorithmic approach, 2nd ed., Boca Raton, FL, USA: CRC press, 2014.

[3] D. Zhang, W. A. Zhang, Z. G. Wu, K. Liu, H. Zhang, and Y. B. Zhao, "New advances in distributed control of large-scale systems," *Mathematical Problems in Engineering*, vol. 2015, 2015. https://doi.org/10.1155/2015/102469

[4] J. M. Pierson, Large-scale Distributed Systems and Energy Efficiency: A Holistic View, Hoboken, NJ, USA: John Wiley & Sons, Inc., 2015. https://doi.org/10.1002/9781118981122

[5] M. van Steen and A. S. Tanenbaum, "A brief introduction to distributed systems," *Computing*, vol. 98, no. 10, pp. 967–1009, 2016. https://doi.org/10.1007/s00607-016-0508-7

[6] U. Tos, "Data replication in large-scale data management systems," Doctoral Dissertation, Université Paul Sabatier-Toulouse III, France, 2017.

[7] C. Coronel and S. Morris, Database Systems: Design, Implementation, and Management, 13th. ed., Boston, MA, USA: Cengage Learning, 2019.

[8] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Computing Surveys (CSUR)*, vol. 37, no. 1, pp. 42–81, 2005. https://doi.org/10.1145/1057977.1057980

[9] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, 7th. ed., Hoboken, NJ, USA: Pearson, 2016.

[10] M. T. Özsu and P. Valduriez, (2020). Principles of distributed database systems, 4th. ed., Cham, Switzerland: Springer, 2020. https://doi.org/10.1007/978-3-030-26253-2

[11] P. Bailis and A. Ghodsi, "Eventual consistency today: Limitations, extensions, and beyond," *Communications of the ACM*, vol. 56, no. 5, pp. 55–63, 2013. https://doi.org/10.1145/2447976.2447992

[12] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin,..., and W. Vogels, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007. https://doi.org/10.1145/1294261.1294281

[13] S. Burckhardt, "Principles of eventual consistency," *Foundations and Trends in Programming Languages*, vol. 1, no. 1–2, pp. 1–150, 2014. https://doi.org/10.1561/2500000011

[14] D. Frey, A. Mostefaoui, M. Perrin, P. L. Roman, and F. Taïani, "Speed for the elite, consistency for the masses: differentiating eventual consistency in large-scale distributed systems," in *Proc. of 2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, pp. 197–206, 2016. https://doi.org/10.1109/SRDS.2016.032

[15] T. Y. Hsu, A. D., Kshemkalyani, and M. Shen, "Causal consistency algorithms for partially replicated and fully replicated systems," *Future Generation Computer Systems*, vol. 86, pp. 1118–1133, 2018. https://doi.org/10.1016/j.future.2017.04.044

[16] M. Perrin, Distributed systems: concurrency and consistency, 1st. ed., UK: ISTE Press and Elsevier, 2017.

[17] P. E. Kourouthanassis, G. M. Giaglis, and D. C. Karaiskos, "Delineating 'pervasiveness' in pervasive information systems: a taxonomical framework and design implications," *Journal of Information Technology*, vol. 25, no. 3, pp. 273–287, 2010. https://doi.org/10.1057/jit.2009.6

[18] P. E. Kourouthanassis and G. M. Giaglis, "Toward pervasiveness: four eras of information systems development," in *Pervasive Information Systems.* Advances in Management Information Systems, vol. 10, P. E. Kourouthanassis and G. M Giaglis, Eds. M.E. Sharpe, Armonk, NY. pp. 3–25, 2008.

[19] F. Kitsios, T. Papadopoulos, and S. Angelopoulos, "A roadmap to the introduction of pervasive Information Systems in healthcare," *International Journal of Advanced Pervasive and Ubiquitous Computing*, vol. 2, no. 3, pp. 21–32, 2010. https://doi.org/10.4018/japuc.2010070102

[20] J. Sligo, R. Gauld, V. Roberts and L. Villa, "A literature review for large-scale health information system project planning, implementation and evaluation," *International journal of medical informatics*, vol. 97, pp. 86–97, 2017. https://doi.org/10.1016/j.ijmedinf.2016.09.007

[21] R. Buyya and A. V. Dastjerdi, Internet of Things: Principles and paradigms, Cambridge, MA, USA: Morgan Kaufmann, 2016.

[22] R. Srivastava and K. Ahuja, "Pervasive computing and its application to traffic collision and congestion control," *International Journal of Computer Applications*, vol. 100, no. 5, pp. 7–11, 2014. https://doi.org/10.5120/17519-8081

[23] A. M. De Souza, C. A. Brennand, R. S. Yokoyama, E. A. Donato, E. R. Madeira, and L. A. Villas, "Traffic management systems: A classification, review, challenges, and future perspectives," *International Journal of Distributed Sensor Networks*, vol. 13, no. 4, 2017. https://doi.org/10.1177/1550147716683612

[24] S. K. Madria, M. Mohania, S. S. Bhowmick, and B. Bhargava, "Mobile data and transaction management," *Information Sciences*, vol. 141, no. 3–4, pp. 279–309, 2002. https://doi.org/10.1016/S0020-0255(02)00178-0

[25] T. Connolly and C. Begg, Database systems: A practical approach to design, implementation, and management, (6th. ed.). Harlow, UK: Pearson, 2015.

[26] G. Biegel and V. Cahill, "Requirements for middleware for pervasive information systems," In *Pervasive Information Systems.* Advances in Management Information Systems. vol. 10, P. E. Kourouthanassis and G. M Giaglis, Eds. M.E. Sharpe, Armonk, NY. pp. 86–102, 2008.

[27] V. Kumar, Fundamentals of pervasive information management systems, Hoboken, NJ, USA: John Wiley & Sons, 2013. https://doi.org/10.1002/9781118647714

[28] S. Souravlas and A. Sifaleras, "Trends in data replication strategies: a survey," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 34, no. 2, pp. 222–239, 2017. https://doi.org/10.1080/17445760.2017.1401073

[29] S. Limam, R. Mokadem, and G. Belalem, "Data replication strategy with satisfaction of availability, performance and tenant budget requirements," *Cluster Computing*, vol. *22*, pp. 1199–1210, 2019. https://doi.org/10.1007/s10586-018-02899-6

[30] F. Xhafa, A. D. Potlog, E. Spaho, F. Pop, V. Cristea, and L. Barolli, "Evaluation of intra-group optimistic data replication in P2P groupware systems," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 4, pp. 870–881, 2015. https://doi.org/10.1002/cpe.2836

[31] N. Dogra and S. Singh, "A survey of dynamic replication strategies in distributed systems," *International Journal of Computer Applications*, vol. 110, no. 11, 2015. https://doi.org/10.5120/19357-9898

[32] F. Xie, J. Yan, and J. Shen, "Towards cost reduction in cloud-based workflow management through data replication," in *Proc. of 2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*, pp. 94–99, 2017. https://doi.org/10.1109/CBD.2017.24

[33] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database system concepts, 7th. ed., New York, NY, USA: McGraw-Hill Education, 2020.

[34] P. A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency control and recovery in database systems, 1st. ed., Boston, MA, USA: Addison-Wesley, 1987.

[35] A. Gao and L. Diao, "Lazy update propagation for data replication in cloud computing," in *Proc. of 5th International Conference on Pervasive Computing and Applications*, pp. 250–254, 2010. https://doi.org/10.1109/2010.5704107

[36] K. Daudjee and K. Salem, "Lazy database replication with ordering guarantees," in *Proc. of 20th International Conference on Data Engineering*, pp. 424–435, 2004. https://doi.org/10.1109/ICDE.2004.1320016

[37] M. Santana, J. E. Armendáriz-Inigo, and F. D. Munoz-Escoi, "Evaluation of database replication techniques for cloud systems," *Computing and Informatics*, vol. 34, no. 5, pp. 973–995, 2015.

[38] R. K. Ghosh, "Caching and Data Replication in Mobile Environment," in *Wireless Networking and Mobile Data Management*, pp. 443–474, Springer, Singapore, 2017. https://doi.org/10.1007/978-981-10-3941-6_14

[39] N. Tolia, M. Satyanarayanan, and A. Wolbach, "Improving mobile database access over wide-area networks without degrading consistency," in *Proc. of 5th international conference on Mobile systems, applications and services*, pp. 71–84, 2007. https://doi.org/10.1145/1247660.1247672

[40] D. H. Ratner, "Roam: a scalable replication system for mobile and distributed computing", Doctoral Dissertation, University of California, Los Angeles, USA, 1998.

[41] D. Ratner, P. Reiher, and G. I. Popek, "Roam: a scalable replication system for mobility," *Mobile Networks and Applications*, vol. 9, no. 5, pp. 537–544, 2004. https://doi.org/10.1023/B:MONE.0000034707.26695.e8

[42] M. Mohana and C. Jaykumar, "Hierarchical replication and multiversion concurrency control model for mobile database systems (MDS)," *Wireless Networks*, vol. 23, no. 5, pp. 1401–1411, 2017. https://doi.org/10.1007/s11276-015-1190-y

[43] M. Bsoul, A. F. Otoom, A. E. Abdallah, and N. Hamadneh, "An enhanced bandwidth hierarchy based replication strategy for dynamic replication in data grid," *Multiagent and Grid Systems*, vol. 13(2017), pp. 163–175, 2017. https://doi.org/10.3233/MGS-170266

[44] R. Fox and W. Hao, Internet infrastructure: networking, web services, and cloud computing, Boca Raton, FL, USA: CRC press, 2018.

[45] L. H. Etzkorn, Introduction to Middleware: Web Services, Object Components, and Cloud Computing, Boca Raton, FL, USA: CRC press, 2017.

[46] M. Papazoglou, Web services: principles and technology, Harlow, UK: Pearson, 2008.

[47] H. Y. Paik, A. L. Lemos, M. C. Barukh, B. Benatallah, and A. Natarajan, Web service implementation and composition techniques, Cham, Switzerland: Springer, 2017.

[48] H. Safa, H. Artail, and M. Nahhas, "A cache invalidation strategy for mobile networks," *Journal of Network and Computer Applications*, vol. 33, no. 2, pp. 168–182, 2010. https://doi.org/10.1016/j.jnca.2009.08.003

[49] A. Ahmed, D. D. Dominic, and A. Azween, "A novel replication strategy for large-scale mobile distributed database systems," *Journal of Engineering Science and Technology*, vol. 6, no. 3, pp. 268–299, 2011.

[50] T. Hara, M. Nakadori, W. Uchida, K. Maeda, and S. Nishio, "Update propagation based on tree structure in peer-to-peer networks," in *Proc. of 3rd ACS/IEEE International Conference on Computer Systems and Applications,* 2005. https://doi.org/10.1109/AICCSA.2005.1387037

[51] T. Watanabe, T. Hara, Y. Kido, and S. Nishio, "An update propagation strategy for delay reduction and node failure tolerance in peer-to-peer networks," in *Proc. of 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07),* vol. 1, pp. 103–108, 2007. https://doi.org/10.1109/AINAW.2007.92

[52] S. Lim, W. C. Lee, G. Cao, and C. R. Das, "Cache invalidation strategies for internet-based mobile ad hoc networks," *Computer Communications*, vol. 30, no. 8, pp. 1854–1869, 2007. https://doi.org/10.1016/j.comcom.2007.02.020

[53] A. I. Saleh, "An adaptive cooperative caching strategy (ACCS) for mobile ad hoc networks," *Knowledge-Based Systems*, vol. 120, pp. 133–172, 2017. https://doi.org/10.1016/j.knosys.2017.01.005

[54] M. Banane and A. Belangour, "Towards a new scalable big data system semantic web applied on mobile learning," *International Journal of Interactive Mobile Technologies*, vol. 14, no. 1, pp. 126–140, 2020. https://doi.org/10.3991/ijim.v14i01.10922

[55] M. Hammoudeh and A. Al-Ajlan, "Implementing web services using PHP SOAP approach," *International Journal of Interactive Mobile Technologies*, vol. 14, no. 10, pp. 35–45, 2020. https://doi.org/10.3991/ijim.v14i10.14391

[56] F. Mdarbi, N. Zahir, N. Afifi, and I. Hilal, "Web service to automate bibliographic research – case of dependability ontology," *International Journal of Recent Contributions from Engineering, Science & IT*, vol. 8, no. 2, pp. 21–35, 2020.

[57] L. Selvin and Y. Palanichamy, "Push-pull cache consistency mechanism for cooperative caching in mobile ad hoc environments," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 24, no.5, pp. 3459–3470, 2016. https://doi.org/ 10.3906/elk-1406-178

[58] S. El Khawaga, A. Saleh, and H. Ali, "An administrative cluster-based cooperative caching (ACCC) strategy for mobile ad hoc networks. *Journal of Network and Computer Applications*, 69, 54–76, 2016. https://doi.org/10.1016/j.jnca.2016.05.003

8 Author

Ashraf Ahmed Fadelelmoula received his Ph.D. in Information Technology from Universiti Teknologi PETRONAS, Malaysia. He joined University of Khartoum, Sudan as a teaching staff after his graduation. He is currently an assistant professor in the Department of Management Information Systems, College of Business administration, Prince Sattam Bin Abdulaziz University, Saudi Arabia. His teaching and research interests include Management Information Systems, Distributed Database Systems, Mobile Databases, Cloud Computing, and ERP Systems.

Article submitted 2021-03-11. Resubmitted 2021-05-31. Final acceptance 2021-06-09. Final version published as submitted by the authors.