IBIMA Publishing
Journal of Cloud Computing
http://www.ibimapublishing.com/journals/JCC/jcc.html
Vol. 2014 (2014), Article ID 545850, 9 pages
DOI: 10.5171/2014.545850

Research Article

Integrating Cloud Computing and Mobile Applications: A Comparative Study Based on *Icloud* and *Sanscode*

Christian Moss

Cyberdesign Works, Sydney, Australia

Correspondence should be addressed to: Christian Moss; christian@cdwsydney.com.au

Received date: 5 September 2013; Accepted date: 27 January 2014; Published date: 31 December 2014

Abstract

Cloud computing is widely used to share and synchronize data between mobile devices such as smartphones and tablets. *iCloud* is Apple's cloud system for use with *iOS* and *Mac OS* devices and applications. However, a major limitation of iCloud is that it is not compatible with other mobile platforms such as Google's *Android* and *Windows Mobile*. This paper discusses *Sanscode*, a cross platform cloud computing system developed by *Cyberdesign Works* to address the underlying interoperability problems. We note that Sanscode could provide a practical alternative solution which supports much of iCloud's functionality with several additional features.

Keywords: Cloud Computing, Data storage, iCloud, iOS, Mobile Devices

Introduction

Data can no longer be seen to be in any one place at any one time, nor can it be seen as belonging to any one person. The 'Cloud' as it is commonly known, can be considered as a collective system of data storage with network connections that can be accessed by any device utilizing a standard Internet connection. Although we have witnessed an increase in the popularity of cloud computing systems in recent years, many argue that the technologies underpinning such systems have existed since the birth of the web. In fact, the Cloud is often akin to the web which has become more versatile and ubiquitous as communication technologies continue to improve. Furthermore, one can argue that the syncing and storing of data between devices in a cloud system (also known as 'cloud storage') should not be classified as cloud computing, but rather 'distributed storage', as true cloud computing implies some parallel/distributed data processing across large-scale networks. Nevertheless, this paper will use the term *cloud* to refer to the data storage aspect of cloud computing rather than the latter (more general) data processing implementation.

This paper is motivated by the potential impacts of cloud computing on mobile devices. In particular, it investigates the practical limitations of Apple's *iCloud* system (Apple Inc. 2013) in the context of mobile application development. It describes how some of the shortcomings of iCloud could be overcome using *Sanscode* - an interoperable cloud based data repository solution developed for mobile devices and web-based content management systems. To facilitate its comparison with iCloud, a metric was used with criteria selected to evaluate mobile applications based on the following set of minimal/client requirements:

- The system should work over multiple devices and across different OS platforms e.g. iOS/Android/Windows Mobile etc.
- The system should be capable of handling the syncing of data in the background.
- The system should be automated thus requiring minimal input from the user; c.f. *usability*.

As with most cloud based systems, the system should be scalable and able to handle a large number of devices.

The rest of the paper is organised as follows. Section 2 examines the advantages and limitations of the iCloud system and its evaluation against the criteria described above. In Section 3, Sanscode, a proprietary cloud based solution developed at Cyberdesign Works, is described. Section 4 examines the limitations and future revisions of Sanscode in terms of the criteria set. Section 5 presents a summary of this paper, with conclusion remarks and future directions of work

iCloud

With the increasing sales of smartphones and mobile network enabled devices such as tablets and notebooks, the need for decentralized data storage has emerged. In the case of the Apple *iPhone*, for example, it was sufficient in the past to use a central PC as the main storage unit for user data, as users would simply synchronize their phone with their PC in order to maintain data consistency. However, many users now posses multiple devices and, as a result, syncing every such device manually is no longer a feasible option. Apple's introduction of the iCloud system in 2011 was an attempt to address this and other issues.

The history of iCloud is well documented in literature; for example, see discussions in (Rocchi, 2013). The main function of predecessors to the iCloud system was to enable cloud computing services amongst indigenous Apple applications, the official iCloud API (Application Programming Interface) which allowed third party application developers, to utilize Apple's iCloud services was unavailable until the use of iOS5 in 2011. However, despite the introduction of iOS6 in late 2012, iCloud has been criticised by many third-party developers for bugs that rendered some features unusable, particularly in *Core Data* - Apple's database and data handling system (Apple Inc. 2013). Additionally, storing and syncing larger amounts of data between user devices also caused several major problems (Hamburger, 2013).

According to the recently held *Apple Worldwide Developers Conference 2013*, iOS 7 would feature *iCloud Keychain* as a future update. This would function as a secure database, allowing information such as a user's website login details, Wi-Fi network passwords, credit/debit card details and other account data to be stored securely. This would allow the data to be quickly accessed; for example, using the *auto-fill* on a web page. However, the most important advantage of Keychain is that it uses the relatively strong 256-bit AES encryption technology (Dobbertin, 2005) for data storage on Apple's devices, or when such data is pushed from iCloud between a user's trusted devices[1].

Advantages of iCloud

Among iOS developers, iCloud is rapidly becoming the standard cloud service. A principal strength of iCloud is its inbuilt database management system (Core Data) which forms the basis of iCloud and is often used as a foundational data storage framework. The Core Data system can be compared to *Structured Query Language* (SQL), as both are persistent data storage systems which allow data to be managed via statements and queries. The main difference between the two is that SQL is strictly a relational database management system (RDBMS), whereas Core Data is an object-oriented database management system (OODBMS) (Paterson, 2006).

The main advantage of Core Data over SQL is that Core Data can store complex data types as custom objects, allowing attributes of these objects to be accessed and queried directly. SQL is limited in this respect; although it is possible to store custom objects (e.g. as binary data strings), this can often lead to deterioration in performance and adds complexity to the application. Furthermore, once an object has been encoded as a binary string, its attributes and properties can no longer be directly accessed.
Other advantages of iCloud include:
After the initial set-up, iCloud handles the majority of the data synchronization in the background. For example, if a user downloads an application onto his/her smartphone it will automatically be copied onto their other iOS based devices.
Application data can automatically be saved or backed up to iCloud. For example, an application that allows a user to create a MS Word document can also save that document to iCloud. If the device and (hence) the application is lost, the data or the document can still be restored.
iCloud is useful for e-commerce applications; a user can have a single user id - in Apple's case an *iTunes id*. It can then use this id to authenticate and purchase a range of services across platforms and devices. With the future release of iCloud Keychain, this may become a more prevalent feature.
For application developers, iCloud is particularly attractive as it is relatively easy to use and requires little interaction from the user; everything is authorized with the single iTunes' *id* and handled in the background.

An application featured on the *App Store* is likely to have a positive effect on its sales. An application is more likely to be featured in App Store (solely at Apple's discretion) if it adopts Apple's technological frameworks. This has encouraged developers to choose iCloud over other platforms (Counsell, 2013).

Limitations of iCloud

One of the major causes for concern with cloud services in general is the privacy of users' data. Many users often feel uncomfortable with their data being stored in the cloud. For example, Google has recently been accused of violating many privacy laws with users' data, when it takes the view that information already available elsewhere on the Internet or in public records is not to be regarded as private or confidential (Svantesson, 2010). In the case of iCloud, application data is kept encrypted on Apple's servers, with Apple maintaining the master key for decryption at its own discretion; for example, when requested by government agencies (Foresman, 2012).

As with other cloud services, another challenge for iCloud is that whilst network connections are becoming faster and more affordable, the movement of large amounts of data can often be slow and costly. This is especially the case if the user is connecting over a 3G/4G network where many mobile service providers still charge considerably high costs for data transfer.
However, the main limitations that we have experienced with iCloud within our work are: 1) Data visibility; 2) Interoperability issues; and 3) Apps' review time. These are discussed in the following sections.

Data Visibility

A limitation with iCloud is that the back-up data is hidden from both the developer and the user. Whilst an application can configure its data to be automatically backed up in iCloud, such data is not accessible as the actual process of backing up or recovering is handled by the iOS/Mac OS system in the background. This can cause the following problems:

If some malicious or corrupt data existed which causes the application to malfunction, the user would have to delete the application and reinstall.

If an application chooses not to have its data backed up via iCloud, and the application is re-installed, then this will be treated as a fresh install and all of the user's data would be lost.

A reinstalled application whose data is backed up in iCloud might continue to work correctly, until corrupted data last saved to the iCloud is recovered.

Interoperability

Although Apple's iPhone has a strong market presence, other systems such as Android (Meier, 2012) and Windows Mobile devices are becoming more popular. Yet, a cloud service that can work between different devices is not yet fully available. For many Apps developers, the key limitation with iCloud is that it is only compatible between iOS and Mac OS enabled devices. This means that it cannot store or share data between the different smartphone systems. For this reason iCloud is not a truly generic cloud computing service. We shall return to these interoperability issues later in this paper when discussing Sanscode.

App Review Time

Another drawback concerning iOS is Apple's review process/policy for newly uploaded Apps. Here, a new App or its updates could often take a considerable length of time (weeks) to be approved and pushed live by Apple. This is due to Apple's policy on reviewing every application submitted to the App store. With almost a million active applications in the app store as of June 2013 (Costello, 2013), this lengthy review process is understandable. Even if the review process was much faster, the application still needs to be recompiled and submitted to Apple, causing further delay and/or consuming extra resources.

Sanscode Applications Framework

To address the aforementioned interoperability issues with iCloud, Cyberdesign Works have developed *Sanscode*, a custom PHP based web-application development framework which was originally developed to handle websites and online content management systems. Specifically, Sanscode has been extended with enhanced functionality to act as a cloud based data repository between mobile devices and web servers, providing a cross platform solution by adopting standard based/compliant web technologies such as SQL and JSON. Figure 1 depicts a practical realization of Sanscode, with fundamental operations of the application development framework described as follows.

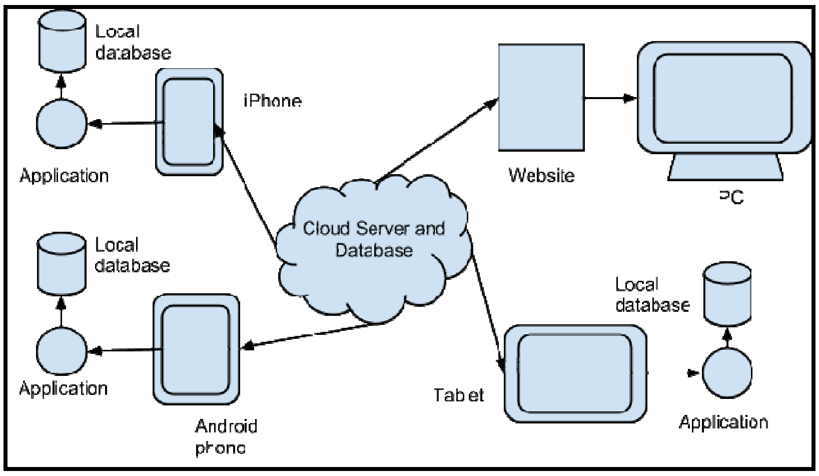

Figure 1: Mobile Apps/devices constructed using Sanscode extract data from the central repository to create their own local database.

The majority of the application data is stored on the server side database in the form of JavaScript, HTML files and SQL entries. This is depicted in Figure 1 as '*Cloud Server and Database*'.

When a user initially downloads the application, a connection is made to the server and creates a copy of the database to store on the local device. Once completed, the application has all the necessary data to function without connecting to a server. In essence, any downloaded applications can also have an "offline mode" for execution when the device is not connected to the Internet.
The application receives a version number representing the version of the server's database it has downloaded and stores this on the mobile device.

Each time the application is opened it checks if newer updates have been made available to the server side database. If so, it downloads all the new necessary data to update the local database on the mobile device.

If a change is made to the server side data, the server's version number is incremented; by comparing the version number of the application to the version number of the server, only the necessary data that is needed to update the application can be determined and downloaded rather than downloading the entire database each time.

Thus, the above discussion highlights how Sanscode enables the application data to remain consistent across all devices, by automatically syncing any changes made to the central data repository. Technically, when applications are executed using Sanscode, they are downloaded and cached (stored) in the local store, allowing them to be re/started via Sanscode as desired. This makes installing and updating applications relatively simple (and seamless) to the user. In essence, Sanscode solves the cross platform issues by having a central SQL database located on a web server that communicates between devices using JSON (see http://www.json.org/). JSON allows data to be encoded using a Key->Value format then sent and returned as POST or GET data in HTML requests. In passing, JSON is widely supported as a de facto standard and is also relatively lightweight, making it a practical method of communicating in the cloud. For transferring large files, such as audio and images, Sanscode automatically wraps them in a *ZIP* file before uploading to the device. The advantages of using a ZIP file approach are four-fold: 1) ZIP files are widely supported, 2) Multiple files/types can be bundled. 3) Data is compressed thus reducing the total data transfer size. 4) With a bundled ZIP a file transfer protocol can be used to download the data. This last point allows an application to give an indication of the download time and progress to the user.

The advantages of this solution are twofold; firstly, the application can be dynamically changed without the lengthy review process as required by Apple. Secondly, as the application stores a version of the database in the device's permanent memory, it can continue to function offline, thus separating it from a simple/common *webapp* (Freeman, 2011). In effect, the application exists in two places at any point in time; on the user's device and in the cloud. This approach is particularly beneficial when the App requires an urgent update[2]. Compared to iCloud, which requires hard coding of these changes, re-compiling the App and submitting it to Apple, the changes can be made instantly on the server side database and then synced locally by the Sanscode enabled application to the device.

Discussions and Systems Evaluation

As cloud computing demands increase, the needs for cross platform applications also increase. This adds further incentive to application developers to seek an alternative in-house solution. Viewed in this light, it is not difficult to envisage a decline in the use of iCloud for mobile applications. However, whilst Sanscode was designed to provide a platform neutral solution for general cloud systems in light of the limited interoperability of iCloud, there are still design aspects where improvements represent the focus of our current work in progress. These are described below.

Security: Although Sanscode uses hashing to store sensitive information such as passwords, it does not encrypt all of the user's data. Similarly, communications between the server and the device are, by default, unencrypted. However, the plan to incorporate such default security safeguards is underway as usage (and popularity) of Sanscode continues to grow.

Updates & Maintenance: At present, Sanscode is potentially susceptible to the (inevitable) future updates by all proprietary cloud based systems including iOS/iCloud and Android/Google-Cloud systems upon which the respective mobile devices are operated. To illustrate, one such problem we have experienced in the course of our development of Sanscode concerned the consistent parsing of JSON objects using a recently updated system class file in iOS that rendered many functions deprecated. Here, it is likely that, in the future, other functions may be changed resulting in the data that is sent from the central server being parsed incorrectly. As a platform neutral application development framework, Sanscode should be maintained as a matter of principle in accord with the updates from the (cloud based) systems that it supports to ensure correct functionality.

Usability: From an application developer viewpoint, a principal attraction of iCloud is that no explicit sign on is required of the user, as the system automatically identify the connecting mobile device using the iTunes' id of its owners. By contrast, a third party solution such as Sanscode or *Drop Box* (Error! Hyperlink reference not valid.) often necessitates the account registration by a user/owner. Such *single sign on* (SSO) requirements becomes a real issue when a user has multiple (different) accounts on several third-party (albeit platform neutral) cloud based systems for which a user may have potential different credentials for each cloud service.

Several comments are in order. First, the iCloud system is attractive as it offers many features to its users with little interaction needed. Given the imminent introduction of Keychain technology as described in Section 2, users of iCloud will continue to be assured that the transactions made with the connecting mobile devices are both easy and secure. Indeed, the recently launched iPhone

5S (in late 2012) has incorporated a well featured fingerprint scanner which helps validate the identity of its user. Second, many iOS applications are developed by individuals who may not have the resources to create their own cloud services for supporting cross platform applications. For them, iCloud naturally remains an appealing solution. Third, application development in the iCloud environment based on the well documented MVC (*Model View Controller*) design pattern as described in (Apple Inc., 2013) facilitates flexibility in design (particularly) at the services integration layer, as it decouples a model object (application/data) from its presentation on the device (view) within the cloud environment (where the controller object operates). This in turns improves services cohesion, enabling better management of web/services and data both locally and remotely in a single service abstraction framework enhanced (with exposed interface elements) to realise a business transaction. As such, it provides a usability focus that is completely at one with the SSO capability afforded by iCloud.

Criteria Evaluation

The current implementation of Sanscode is compared with iCloud using the criteria outlined in Section 1. The results are summarised in Table 1, where the major differences are highlighted.

Table 1: Evaluation Criteria

Requirements/Criteria		iCloud	Sanscode
1. Device functions	Multiple devices	Yes	Yes
	Multiple OS/browsers (Safari, Chrome, FireFox and IE)	iOS & Mac OS only. NB. IE/Vine only	iOS, Mac OS, Android, Windows - except Windows Mobile.
2. Data Syncing & Recovery		Yes (background)	Yes (background & foreground)
3. Services automation/ usability		Initial registration only	Individual Sign-On
4. Scalability		Yes NB. iOS7 required as earlier releases might cause problems	Yes NB. Tested with earlier iOS releases.

Summary and Conclusions

In this paper, we have shown how cloud computing has become essential to modern day data needs, with users having multiple devices across different platforms. We also examined the approach adopted by Apple's iCloud and compared it with Sanscode.

The paper first highlighted iCloud as a cloud based solution designed to work exclusively between iOS and Mac OS enabled systems. As such, it lacks general interoperability that a 'true cloud' solution might offer. By adopting standard based technologies such as JSON and SQL, an alternative solution that provides cross platform compatibility has been developed and illustrated using

Sanscode - specifically an OS/platform neutral data repository constructed for mobile devices and web/content management. Using iCloud as our benchmarking system, the interoperability issues were studied in some depth, highlighting some of the allied design issues including security, software updates and maintenance, and importantly, the usability concerns.

As cloud computing becomes more prevalent with widely documented problems being addressed, we envisage that its interactions within mobile computing community will continue to increase and be improved. Further, with the mobile market becoming more segmented in terms of operating systems and their support, demand for interoperable cross platform solutions such as Sanscode as described in this paper are likely to escalate. To this end, further work on improving the proposed cloud based data repository solution in the identified areas of work in progress is currently underway.

Notes

[1] Keychain is already used on Mac OS and utilizes private and public keys to validate users.
[2]JNLP solved a similar problem; see http://docs.oracle.com/javase/tutorial/deployment/deploymentInDepth/jnlp.html . However, being a Java-based technologies, it is not supported by the native iOS system.

References

1. Apple Inc, (2013), 'Modal View Controller - Cocoa Core Competencies', available at : https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/MVC.html, last accessed on 1/9/2013.

2. Apple Inc., (2013), 'iCloud for Developers', available at: http://developer.apple.com/icloud/index.php, last accessed at 1/9/2013.

3. Apple Inc., (2013), 'Introduction to Core Data Programming Guide', available at: https://developer.apple.com/library/ios/documentation/cocoa/conceptual/CoreData/CoreData.pdf, last accessed on 1/9/2013.

4. Costello, S., (2013), 'How Many Apps Are in the iPhone App Store', (*About.com Guide*), available at http://ipod.about.com/od/iphonesoftwareterms/qt/apps-in-app-store.htm, last accessed on 1/9/2013.

5. Counsell, D., (2013), 'How to get featured on the app store', available at http://www.realmacsoftware.com/blog/how-to-get-featured-on-the-app-store, last accessed o 1/9/2013.

6. Dobbertin, H. (ed), Rijmen, H., Sowa, A., (2005), Advanced Encryption Standard - AES, Proceedings of Advanced Encryption Standard - AES: 4th International Conference (AES 2004), Springer-Verlag, Berlin, Heidelberg.

7. Foresman, C. ,(2012), 'Apple holds the master decryption key when it comes to iCloud security, privacy', available at http://arstechnica.com/apple/2012/04/apple-holds-the-master-key-when-it-comes-to-icloud-security-privacy/, last accessed on 10/9/2013.

8. Freeman, E., (2011), Head First HTML5 Programming: Building Web Apps with JavaScript, O'Reilly Media LLC.

9. Hamburger, E., (2013), 'Apple's broken promise: why doesn't iCloud "just work"?' available at http://www.theverge.com/2013/3/26/4148628/why-doesnt-icloud-just-work, last accessed on 31/5/2013.

10. Meier, R., (2012), Professional Android 4 Application Development (Wrox Professional Guides), Updated edition, John Wiley & Sons.

11. Paterson, J., Edlich, S., Hörning, H., Reidar Hörning, R., (2007), The Definitive Guide to db4o, 1st Edition, Apress, Springer-Verlag, New York

12. Rocchi, C., (2013), iCloud for Developers, 1st Edition, The Pragmatic Programmers, LLC.

13. Shankland, S., (2013), CNET.com, available atError! Hyperlink reference not valid., last accessed 1/9/2013.

14. Svantesson, D. and Clarke, R., (2010), 'Privacy and consumer risks in cloud computing', Computer law and security review, 26 (4), 391-397, also available online at: http://epublications.bond.edu.au/law_pubs/347 [last accessed on 1/9/2013]