MEASURING THE USAGE OF SAAS APPLICATIONS BASED ON UTILIZED FEATURES

Ali Bou Nassif
Department of Electrical and Computer Engineering, University of Western Ontario, London, Ontario, Canada

Hanan Lutfiyya
Department of Computer Science, University of Western Ontario, London, Ontario, Canada

Keywords: SaaS pricing models, Software metering, Feature-based approach, Pay per feature.

Abstract: Software as a Service (SaaS) is an online delivery of software to customers as a service. The interest in adopting SaaS has been rapidly increasing due to the advantages of SaaS. Current SaaS vendors such as Salesforce.com charge their customers based on the type of the edition and on the number of users. Many customers are not satisfied with the current pricing model and are requesting to pay according to the actual usage of the SaaS application. SaaS vendors are looking to implement a metering solution that measures the computational resources such as the CPU, memory and transactional usage as well as the utilized features of each SaaS user. This paper focuses on implementing a crucial element of the metering system which is measuring the SaaS applications based on the features utilized by each user. This is also called the Feature-Based Approach.

1 INTRODUCTION

Software as a Service is an online delivery of software to customers (Hoch, Kerr 2001). Rather than purchasing a software license (On-Premise software) for an application such as Customer Relationship Management (CRM) which might cost thousands of dollars, customers subscribe to applications from a SaaS provider and pay fees based on the usage of the application. In a SaaS environment, software becomes a service. The service is deployed from a datacentre and accessed through the Internet on a recurring fee basis (Hoch, Kerr 2001, Chong, Carraro 2006). Demand for SaaS providers is increasing and the SaaS architecture should be improved to accommodate the substantial increase of new customers (Turner, Budgen & Brereton October 2003). The advantage of SaaS is that the SaaS provider and not the customer manages the application and the supporting infrastructure (Nassif, Capretz 2010). For the customer, the use of SaaS reduces costs of maintaining an infrastructure and updating of Software to the customers.

SaaS has many advantages over internally hosted applications. These advantages include very low cost of entry since the customer does not have to worry about software licenses, hardware and maintenance of the software/hardware. On the other hand, SaaS has some disadvantages. These include security and privacy issues.

While the demand of SaaS is increasing, many SaaS vendors such as Salesforce.com (Salesforce 2009) are still charging customers a flat rate based on the number of users that the customer has and the edition being used. While this pricing model might be convenient for some SaaS customers, there are several concerns. These concerns include (Schinkel 2006):

- SaaS customer pays for the total number of users regardless of their activities;
- The users can only utilize the features of the edition which the user is subscribed to;
- Customers are committed to paying for the total number of users during the contract period.

SaaS users as well as SaaS vendors are looking for a sophisticated metering solution so that customers pay for SaaS services the same way they pay for hydro (water and electricity) in North America. The components of this metering solution may include data about users, computational resources such as CPU and memory usage, transactions executed, storage tracking and the features used by each user (Chaudhuri 2008, Reinvented 2008, Progress 2008).

Each SaaS application has a predefined set of features or functionalities that the users can utilize. Functionality is the procedure where information about an entity is obtained, exported or stored (Nassif, Capretz & Ho 2010). Entities can represent any aspect of business. Examples of entities include customer, invoice, receipt, etc. Examples of functionalities can be seen in a Call Centre application. In a Call Centre application, a user might create a ticket, send mass email, and attach a note. In this case, the user will utilize three features. Monitoring the use of these features may show that the user with a certain phone number sent five text messages and three voice messages last month. Since each feature might have a predefined cost, the total cost can be calculated. Based on this, we introduce *Feature-Based Approach*, in which users are allowed to utilize all the features that the SaaS provider supports.

This paper focuses on the design, implementation and evaluation of a component of a metering solution that measures metrics of the features of a SaaS application utilized by each user (also known as the Feature-Based Approach). This includes providing information about the features (features names and the number of uses of each feature) utilized by each user during a period of time (for example, monthly reports). Another requirement is to determine the number of sessions and length of each session by user. By knowing the cost of each feature, the total cost based on the utilization of these features can be determined for each user.

The rest of the paper is organized as follows: Section 2 presents the related work. Section 3 demonstrates the system design while Section 4 discusses the implementation of the Feature-Based Approach. Section 5 validates the work. Section 6 proposes a general discussion about the proposed approach. Finally, Section 7 concludes the paper.

2 RELATED WORK

This section presents the current work relevant to software usage. Some of this work is academic and some is commercial. A summary of this work is described in this section.

Bronner (Bronner 2007) describes a proposal for a system measuring the usage of Open Source Software (OSS). For each user that utilizes an OSS, metrics regarding software usage such as the IP Address, User ID, and MAC Address of a user are measured locally and then transferred to a server through the Internet.

Commercial tools such as Softtrack (Softtrack 2010), Express Metrix (Express 2010) and Open It (OpenIt 2010) can be used to audit and control users' activities and can be installed locally on the user's workstation to log information such as the user's identity (user login name, IP Address, logon time, total time logged on).

In addition to offering software, many providers offer hardware and infrastructure as a service. IaaS provides the underlying operating system and the hardware needed to run any application. IaaS providers include Amazon Web Services (AWS) (AWS 2010), Microsoft Windows Azure (Azure 2010) and Google App Engine (Google 2010). IaaS vendors provide computational resources such as CPU, memory, networking, bandwidth and security on demand. For instance, Amazon AWS EC2 offers nine types of instances (Amazon 2010). Each instance has specific computational resource capacity. Customers can rent any type of these instances and pay a fixed rate per instance-hour.

The current pricing models of IaaS vendors such as Amazon EC2 and Microsoft Windows Azure cannot be adopted by SaaS providers for several reasons. First, IaaS customers can determine the amount of computational resources required to run their businesses and consequently, IaaS customers pay a fixed fee per hour to rent these resources. The amount of these computational resources might fluctuate according to the demands of a business. Customers can increase the computational resources in peak periods and pay more for these resources. Alternatively, customers can decrease the quantity of these resources when the business is slow and pay less. On the contrary, this cannot be achieved by SaaS customers. SaaS customers such as Salesforce.com's can switch to a higher edition where they pay more but they cannot switch to lesser expensive editions when their business is slow (Schinkel 2006). Another reason that IaaS pricing models is not a good model for SaaS is that IaaS customers can, for example, shutdown their EC2 instances anytime and stop paying. On the other hand, SaaS users have to pay even if they are not using the SaaS application. Furthermore, Garfinkel (Simons 2007) argued that IaaS services such as Amazon EC2 and S3 are still young and immature and they are still not ready to service e-commerce customers. On the other hand, SaaS businesses are booming. By 2011, 25% of new business software will be delivered as SaaS, and the expected SaaS revenue in 2011 is $19.2 billion (Ju, Zhijie 2010) (Cao, Zhou 2009). This concludes that the current pricing models of IaaS do not provide new insights into new pricing models for SaaS.

3 SYSTEM DESIGN

This section presents the design of the Feature-Based Approach. The Feature-Based Approach allows users to be charged according to the features that they utilize, so users who use the application more will be charged more as opposed to the current pricing model of some SaaS vendors where all users are charged the same regardless of their activities. The Feature-Based Approach also gives a SaaS customer an idea about each user's activities as a monthly report shows the features utilized by each user.

Web servers such as Apache and the HTTP protocol for communicating with web servers are stateless (Connolly 2010). Stateless means that each HTTP request has to be handled based entirely on information that comes with the request. Support for the Feature-Based Approach requires the application to be stateful so that each user's HTTP requests can be mapped and grouped together. Sessions and cookies will be used to convert the stateless environment to a stateful one. This section describes how this can be achieved.

3.1 Sessions Creation and Management

When a user starts a SaaS application, it has to authenticate itself. In a session-based application (Feature-Based Approach), when the web server receives a request, it first checks if this user has an existing valid session by checking the cookie header (if existing) in the user's request. If there is no cookie from the web server, this indicates that the user is visiting the website for the first time. If the web server found a cookie header that carries information such as session identifier (SessionID) in the user's request, it tests for a valid session. The session might be invalid if the user tries to access the application after a certain time of inactivity because the web server often destroys sessions after a certain period of time of user's inactivity to release the memory. In the case that there is no valid session, the web server prompts a login form asking the user to log in to the application. When the user logs in, the web server creates a session of session identifier SessionID, then the username and the password will be checked in a database. If the check is successful, the user will be redirected to a page where it can utilize features (for example, the member's web page). If the check is not successful, the user is prompted to re-enter its username and password and nothing will be stored in the database. When the web server creates a session, it sends the session identifier SessionID to the user through a cookie by using a Set-Cookie header. The browser then sends the cookie that holds the SessionID with each request. The web server can recognize the user by comparing the SessionID that is found in the cookie header and the SessionID that is stored in the server. By default, the expiry date of the cookie is zero. This means that the cookie will expire when the session is destroyed or when the user closes all the instances of the browser. The session will be destroyed when the user explicitly logs out or after certain time of user's inactivity. If the cookie expires, the user has to log in to the system again.

Session management is the process of keeping track of a user's activities when they access multi-user web applications. Session management and tracking can be implemented using several methods. These include User Authorization, Hidden Form Fields, URL Rewriting and Cookies. In this work, cookies will be used for session management and tracking because cookies are safe, highly customizable and easy to implement (Stein 2010).

3.2 Features Identification and Apache Log File

Apache is the web server used in this approach. Apache logs each HTTP request and thus logs every request that is related to features. Each log entry has a default set of parameters as shown in Figure 1. In this work, all features of the SaaS application are labelled as Feature1, Feature2, etc. The mapping between the labelled features and the actual names of the features (e.g. "Attached Notes" and "Send Messages") is stored in a database. In order to distinguish between the requests that are related to features and the requests that are other page hits (not related to features), the URL of each feature is edited before users can utilize the features to contain the name of the feature as well as the session identifier. This is shown in Figure 2. The name of a feature is predefined (E.g. Feature1) while the session identifier is a variable that is different for each user. This is helpful in relating each feature to its related session. For example, if the URL to access Feature1 was http://www.saas.com/sales.php, the new URL will be http://www.saas.com/sales.php?feature=Feature1&SessionID=a2223h6gfd. Eventually, the information saved in the log file is used to store information (such as the access time of each feature, the feature name and the session identifier of the session where each feature belongs) in the database.

The Apache module mod_log_config can be reconfigured so that Apache can log each request with the required parameters. An example of the parameters that Apache logs includes the IP Address of the user, the date of the request when it was sent, the Request Method, the URL and the Response Status Code.

An alternative approach considered is based on saving information about features at the same time that the users are utilizing these features and consequently without using the Apache log file. However, there are two issues with this approach. The first issue is a performance issue. Each time a user accesses a feature, a connection will be established with the database.

In a typical SaaS application where many users are using the application simultaneously, the server might become a bottleneck. In the first approach information is recorded about features through URL modification (which is done before the application is deployed) and through using a log file which is already used by web servers. The log file can be read and saved in the database once per day. The second issue of the alternative approach is that features will be stored in the database regardless if a feature was successfully being used. The first approach described addresses this issue since the status code of each request will be logged as shown in Figure 2. Although Apache logs successful and non successful requests, in the Feature-Based Approach, only successful requests will be saved in the database and consequently, customers will not be charged for unsuccessful requests.

```
127.0.0.1 [22/Apr/2009:22:48:19 -0400] "GET /saas/main.php HTTP/1.1" 200
127.0.0.1 [22/Apr/2009:22:48:21 -0400] "GET /saas/main.php HTTP/1.1" 200
127.0.0.1 [22/Apr/2009:22:48:24 -0400] "GET /saas/email.php HTTP/1.1" 200
127.0.0.1 [22/Apr/2009:22:51:46 -0400] "GET /saas/main.php HTTP/1.1" 200
127.0.0.1 [22/Apr/2009:22:51:58 -0400] "POST /saas/createdb.php HTTP/1.1" 200
192.0.0.10 [22/Apr/2009:22:54:26 -0400] "GET /saas/main.php HTTP/1.1" 200
127.0.0.1 [22/Apr/2009:22:54:30 -0400] "POST /saas/displaydb.php HTTP/1.1" 200
127.0.0.1 [22/Apr/2009:22:56:59 -0400] "GET /saas/main.php HTTP/1.1" 200
127.0.0.1 [22/Apr/2009:22:57:08 -0400] "POST /saas/selectdb.php HTTP/1.1" 200
```

Figure 1: Apache Log File before the implementation of session.

```
1. 192.168.0.10 [08/Apr/2009:18:01:38 -0400] POST /saas/ http://192.168.0.100/saas/
   HTTP/1.1 302

2. 192.168.0.10 [08/Apr/2009:18:01:38 -0400] GET /saas/index.php HTTP/1.1 200

3. 192.168.0.10 [08/Apr/2009:18:01:41 -0400] GET
   /saas/sales/features.php?feature=Feature1&SessionID=283e7d6365466aa69db9e9059a2d
   ee99 HTTP/1.1 200

4. 192.168.0.10 [08/Apr/2009:18:01:55 -0400] GET
   /saas/marketing/features.php?feature=Feature3&SessionID=283e7d6365466aa69db9e90
   59a2dee99 HTTP/1.1 200

5. 192.168.0.10 [08/Apr/2009:18:02:11 -0400] GET
   /saas/sales/features.php?feature=Feature1&SessionID=283e7d6365466aa69db9e9059a2d
   ee99 HTTP/1.1 Code=200

6. 192.168.0.11 [08/Apr/2009:18:03:26 -0400] POST /saas/ http://192.168.0.100/saas/
   Protocol=HTTP/1.1 302

7. 192.168.0.11 [08/Apr/2009:18:03:26 -0400] GET /saas/index.php HTTP/1.1 Code=200

8. 192.168.0.11 [08/Apr/2009:18:03:56 -0400] GET
   /saas/marketing/features.php?feature=Feature7&SessionID=a6a769bc297493d7337f8b5
   49c8b0861 HTTP/1.1 200
```

Figure 2: Apache Log File after the implementation of session.

In Figure 2, the first log entry of the Apache Log File shows that a user from IP address 192.168.0.10 might have logged in to a SaaS application at 18:01:38 on April 08 2009. The original URL "/saas/" would be redirected (status code = 302) to a new URL http://192.168.0.100/saas/ where 192.168.0.100 is the IP address of the server. When a user submits information using a POST request, the user might be redirected to the main page or other page. After a successful login, the page index.php is displayed. This is seen in entry 2 since the date/time of the second request was exactly the same as the first request but the Request Method of the second request was GET which is used when information is needed from the server as opposed to the first request that was POST. After the user had logged in, a session is created. The session identifier is not shown in the first two entries since the first two entries just show that the user has successfully logged in but the user has not utilized any feature yet. Entry 3 shows that this user has successfully (status code = 200) utilized Feature1 in a session of session identifier = "283e7d6365466aa69db9e9059a2dee99".

Even though each user might have a unique IP address, the user is identified by the session identifier and not by the IP address. This is because one user (one IP Address) might access the application using the same machine but through different browsers and consequently, this would be considered as many users (each browser corresponds to a user). Entry 4 illustrates that the same user (same session identifier) has successfully utilized Feature3 after 14 seconds. Entry 5 demonstrates that the same user has successfully utilized Feature1 again. Entry 6 and 7 shows that another user from IP address 192.168.0.11 has successfully logged in and a new session was created. Entry 8 illustrates that the second user has successfully utilized Feature7 in a session of identifier "a6a769bc297493d7337f8b549c8b0861". As a conclusion, on April 8, 2009 from 18:01:38 until 18:03:56, two users accessed the SaaS application. The first user utilized Feature1 twice and Feature3 and the second user utilized Feature7.

3.3 Saving Information in the Database

The information about the users who are using a SaaS application is stored in superglobal variables such as session variables and in an Apache log file. Superglobal variables can store information such as the username of the user, session start date, session identifier, IP address of the user. However, the Apache log file stores the IP address of the user, the access date of each request, the request method used, the URL, the protocol used and its version and the status code. To save this information (found in session variables and log file) permanently, a database which contains tables about the users, sessions and features will be created.

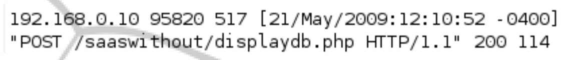
4 IMPLEMENTATION

This section presents the implementation of a Feature-Based Approach. In this work, the SaaS application will be represented as a three-tier web application where the client is a browser, the web server is Apache, the scripting language used is PHP and the database server is MySQL.

Apache is composed of modules. The Multi-Processing Module (MPM) used in this work is the Prefork which is based on the Process-Based Server. The module mod_log_config is responsible for logging all HTTP requests made to the server. Logs are written into a file where log formats, name and location of the file can be configured.

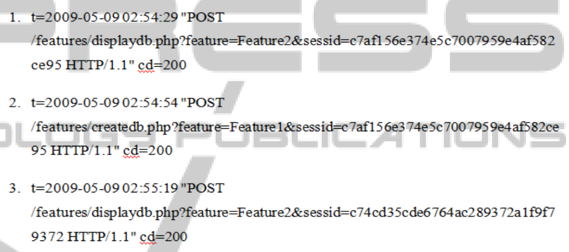
In this work, two log files exist. The first one is the default log file called access.log which logs every HTTP requests (features and non features requests) while the second log file only logs the requests that are related to features in a separate log file which is called access1.log. The goal of the file access1.log is to clean the default file (access.log) by removing all the requests which are not features related and to make this file (access1.log) ready to be read by a special script to save the required information in the database. Figure 3 and Figure 4 depict Apache log files access.log and access1.log respectively. Please note that Apache logs each entry shown in Figure 4 as a separate line without the numbers displayed at the left.

There are two types of information in the SaaS application to be saved in the database; the application-level information and the log-file information. The application-level information includes the username, IP address, session identifier, session start time and session end time. The log file stores information related to the features used in sessions. To store this information, a database *SaaS* is created. The database SaaS includes tables such as *Session*, *IP*, *Time* and *Contain*.



```
192.168.0.10 95820 517 [21/May/2009:12:10:52 -0400]
"POST /saaswithout/displaydb.php HTTP/1.1" 200 114
```

Figure 3: Apache Log File access.log.



```
1. t=2009-05-09 02:54:29 "POST
   /features/displaydb.php?feature=Feature2&sessid=c7af156e374e5c7007959e4af582
   ce95 HTTP/1.1" cd=200

2. t=2009-05-09 02:54:54 "POST
   /features/createdb.php?feature=Feature1&sessid=c7af156e374e5c7007959e4af582ce
   95 HTTP/1.1" cd=200

3. t=2009-05-09 02:55:19 "POST
   /features/displaydb.php?feature=Feature2&sessid=c74cd35cde6764ac28937 2a1f9f7
   9372 HTTP/1.1" cd=200
```

Figure 4: Apache Log File access1.log.

Listing 1 shows the algorithm for saving the session identifier, session start time, IP address and user name in the database in tables saas.Session and saas.IP.

```
1: if the login is successful then
2: establish a connection with the database;
3: sessionid = session identifier;
4: IP = IP address;
5: session_start_date = current time/date;
6: userid = user name;
7: if IP does not exist in table saas.IP then
8: insert into table saas.IP IP;
9: end if
10: insert into table saas.Session sessionid, userid, IP,
       session_start_date;
11: end if
```

Listing 1: Saving application-level information in the database

The monitoring of the session end time is critical since the user might choose to log out from the application or might leave the application for a certain amount of time. Listing 2 displays the algorithm used to implement the session end time.

```
1: establish a connection with the database;
2: user visits a page;
3: session_end_date = current time/date;
4: update table saas.Session set session_end_date to a new
        value;
5: while (user visits a page AND time between 2
        consecutive visits < 20 min)
6: session_end_date = new current time/date;
7: update table saas.Session and set session_end_date to a
        new value;
8: end while
9: destroy session and display log in page;
```

Listing 2: Session End Date.

A scripting file was created to save the information stored in the log file access1.log in the database. Listing 3 shows the algorithm of this scripting file. The information is stored in tables saas.Time and saas.Contain.

```
1: establish a connection with the database;
2: open log file with read permission;
3: while (! EOF)
4: read line;
5: if (line contains a feature AND the status code is 200)
     then
6: sessionid = session identifier;
7: feature_name = feature name;
8: feature_time = feature access time;
9: if (feature_time does not exist in table Time) then
10:          insert into table saas.Time feature_time;
11:    end if
12:         insert into table saas.Contain sessionid,
            feature_name, feature_time;
13: goto 3;
14: end if
15: end while
```

Listing 3: Read Log File and Save Information in Database

5 VERIFICATION AND VALIDATION

In this work, two SaaS applications were created, one with session management and one without. The purpose of this is to measure the overhead caused by session management using cookies. The implementation of session management has a performance penalty because of the overhead that is caused by the following: First, a cookie is sent in each HTTP request as a part of the request's header. Secondly, in each page a user hits to access features, the program checks if this user has a valid session then the program accesses the database to update the session_end_date attribute in table saas.Session with the latest date the user has accessed the application (Implementation of session-end-time as seen in Listing 3).

The overhead caused by session management in the Feature-Based Approach is measured by measuring the response time and the size of each request that is feature related before and after the implementation of session management. The response time is the time that the web server takes to service a request. The response time of each feature was measured before and after the implementation of session. The response time of each request can be measured by configuring the module mod_log_config to log the time taken to service each feature in milliseconds. For the results to be accurate and to avoid the time a process might spend in the waiting queue of the processor, a single user accessed the application and utilized each feature before and after session management. To reduce the percentage of error as much as possible, this user utilized each feature in five different times (t1, t2, t3, t4 and t5) before and after session management. The result of the response time of each feature measured at five different times before session management as well as the average of the five results are presented in Table 1. Table 2 displays the results of measuring the response time after session management. Please note that in all tables below, F1, F2, F3, F4 correspond to Feature1, Feature2, Feature3 and Feature4 respectively. R1, R2, R3, R4 and R4 correspond to Response time at t1, Response time at t2, Response time at t3, Response time at t4, and Response time at t5 respectively

Table 1: Response Time before Session Management.

	R1	R2	R3	R4	R5	AVG
F1	3112	3196	3201	3094	3223	3165
F2	3872	4050	3712	3965	3854	3890
F3	39673	40872	38416	39218	40313	39698
F4	4564	4673	4512	4498	4523	4554

Table 2: Response Time after Session Management.

	R1	R2	R3	R4	R5	AVG
F1	3309	3354	3661	3413	3384	3424
F2	4152	4212	4117	4309	4212	4200
F3	42398	43208	42109	43312	42885	42982
F4	4903	4832	4896	4965	4812	4881

Based on the results shown in Tables 1 and 2:

The overhead of management of Feature1 = (3424-3165)/3165 * 100 = 8.18 %

The overhead of management of Feature2 = 7.9 %

The overhead of management of Feature3 = 8.2 %
The overhead of management of Feature4 = 7.1 %
The average of the overhead is 7.8%.

One of the main factors that improve the performance of the application in this work is that data gathered in the log file can be saved in the database once per day as opposed to the traditional methods when data is saved each time a user is using the application and thus increases the overhead.

6 DISCUSSION

The Feature-Based Approach measures the usage of a SaaS application based on the utilized features. The rest of this section presents general discussion about the proposed Feature-Based Approach.

6.1 Feature Cost

If the cost for each feature is known, customers can be charged for the features they utilize. Determining the cost of each feature is not an easy task. There are several possible approaches that could be used. One approach is that the cost of each feature is based on the total elapsed time taken to execute each feature. For example, if the total elapsed time needed to use Feature1 is 1000 microseconds and the total elapsed time for Feature2 is 5000 microseconds, the cost of Feature2 will be five times as Feature1. The problem with this is that the time may vary depend. For example, if a feature requires data processing the time depends on the amount of data to be processed. Another approach is that the cost of each feature might be determined based on the computational resources (CPU, RAM, Bandwidth, Storage) consumed by each feature. Another possibility is that a SaaS vendor may use market forces. For example, a SaaS vendor might reduce the cost of the features that are competitive with features from different vendors or features which are in high demand might be expensive regardless of the computational resources consumed. Furthermore, the cost of each feature might vary according to the access time the feature is being accessed. For example, the feature cost might be reduced between 6:00 pm and 8:00 am. This can be easily implemented since the access date of each feature will be saved in the database. Future work will determine the best method for determining the cost of each feature for different types of applications.

6.2 Pricing and Business Models

The Feature-Based Approach can be used to support multiple business models. Lyons et al. introduced nine types of business models in emerging online services (Lyons et al. 2009). These models include the Utility Model, the Advertising Model, and the Subscription Model. Current SaaS pricing models falls under the Utility Model where customers typically pay a fee for accessing and using services. The Feature-Based Approach might be used to support alternative pricing models for SaaS. For example, if the business model is based on advertisement, then a specific ad can be associated with a feature. The advertiser would pay based on the number of times the feature is invoked. A service may be associated with one or more features. A fee could be associated with a service based on the features used. The Feature-Based Approach can also support the Advertising Model in a different way. Facebook is an example of a SaaS application that is based on the Advertising Model. If the Feature-Based Approach was implemented by Facebook, then the features of each Facebook user that are utilized the most will be stored. Knowing the features that were used the most by a Facebook user might lead to learning the user's personality or hobbies and consequently, specific ads can be sent to this user to peak their interest.

6.3 Deployment

In the Feature-Based Approach, SaaS features are labelled as Feature1, Feature2, Feature3, etc. This labelling is done by the application's provider and the mapping between the actual names of features and the labelled features is stored in the database server that is owned by the application's host. There are two main reasons for labelling features. First, the actual name of a feature might contain space characters and might be very long. Most Database Management Systems do not accept a space character in their field names where the features are stored. Moreover, the length of field names is limited to a specific number of characters. The second reason for labelling is for standardization. Learning the names of the features in advance (Feature1, Feature2, etc.) facilitates the reading from Apache log file and helps in the database design as well. Consequently, labelling features supports the Feature-Based Approach to be implemented by different SaaS applications. Future work will focus on automatically labelling and mapping features.

7 CONCLUSIONS

SaaS applications are growing rapidly and SaaS vendors are planning to build a metering solution for future billing. While a metering solution might include measuring the usage of a SaaS application based on measuring the computational resources, number of transactions and the features utilized by user, this work focused on measuring the features of a SaaS application utilized by a user.

The design of this work focused on creating a Session for each user, then measuring metrics such as the name and the number of features utilized by a session, the user name and the IP address of the user who started a session, the number and the length of each session that belong to a specific user. This work also proposed different methods to assign a cost for each feature, and showed how the Feature-Based Approach can support different business models.

The current work focused on implementing the Feature-Based Approach which is one component of the sophisticated metering system. The future work includes the following:

- The completion of the metering system such as measuring the CPU and memory usage of each SaaS user.
- Minimizing the overhead caused by session management.
- Designing an infrastructure that allows for multiple methods to be used for pricing features

REFERENCES

Amazon 2010, *Amazon Elastic Compute Cloud (Ec2)*. Available: http://aws.amazon.com/ec2/.

AWS 2010, *Amazon Web Services*. Available: http://aws.amazon.com/.

Azure 2010, *Microsoft Windows Azure*. Available: http://www.microsoft.com/windowsazure/.

Bronner, M. 2007, *Measuring the Usage of Open Source Software*, M.Sc thesis, Department of Applied Information Technology IT University of Goteborg, Sweden.

Cao, L. & Zhou, G. 2009, "Analysis of SaaS-Based Informationization in Small and Medium-Sized Logistics Enterprises", *Proceedings of the 2009 Third International Symposium on Intelligent Information Technology Application Workshops*IEEE Computer Society, Washington, DC, USA, pp. 78.

Chaudhuri, S. 2008, *SaaS Pricing and Metering*, Business and Technology Whiteboard.

Chong, F. & Carraro, G. 2006, *Architecture Strategies for Catching the Long Tail*, Microsoft Corporation.

Connolly, D. 2010, *Hyper Text Transfer Protocol RFC 2616*. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html.

Express 2010, *Express Metrix*. Available: http://www.expressmetrix.com.

Google 2010, *Google App Engine*. Available: http://code.google.com/appengine/.

Hoch, F. & Kerr, M. 2001, *Software as a Service: Strategic Backgrounder*, SIIA, Washington, DC.

Ju, J. & Zhijie, L. 2010, "Research on Key Technology in SaaS", International Conference on Intelligent Computing and Cognitive Informatics, , pp. 384.

Lyons, K., Playford, C., Messinger, P.R., Niu, R.H. & Stroulia, E. 2009, "Business Models in Emerging Online Services", *Springer Berlin Heidelberg,* vol. 36, pp. 44-55.

Nassif, A.B., Capretz, L.F. & Ho, D. 2010, "Software Estimation in the Early Stages of the Software Life Cycle", *Proc. International Conference on Emerging Trends in Computer Science, Communication and Information Technology*, pp. 5.

Nassif, A.B. & Capretz, M.A.M. 2010, "Moving from SaaS Applications towards SOA Services", *Proceedings of the 2010 6th World Congress on Services*IEEE Computer Society, Washington, DC, USA, pp. 187.

OpenIt 2010, *Open It*. Available: http://www.openit.com.

Progress 2008, *SaaS Billing and Metering*, Progress Software Corporation.

Reinvented 2008, *Software as a Service*. Available: http://www.businessreinvented.co.uk/saas.

Salesforce 2009, *Selecting the right Salesforce CRM edition*. Available: http://www.salesforce.com/crm/editions-pricing.jsp.

Schinkel, M. 2006, *One-Sided Contracts make for Unhappy Customers*.

Simons, G. 2007, "Commodity Grid Computing with Amazon's S3 and EC2", *UESNIX*, pp. 7.

Softtrack 2010, *Softtrack*. Available: http://www.softwaremetering.com.

Stein, L. 2010, *What Cookies Are*. Available: http://www.w3.org/Security/Faq/wwwsf2.html.

Turner, M., Budgen, D. & Brereton, P. October 2003, "Turning software into a service", *IEEE Journals,* vol. 36, no. 10, pp. 38-44.