Fault Tolerant Platform for Application Mobility across devices

T. N. Anitha [1]
Associate Professor, Dept of Computer Science & Engineering , SJCIT, Chickballapur-562101,Karnataka,India

Jayanth. A [1]
Project Engineer,Oracle India Private Limitted
Bhannerughatta Road, Near Diary circle, Koramangala, Bangalore, Karnataka,India

***Abstract*—In the mobile era, users started using Smartphone's, tablets and other handheld devices, The advances in telecom technologies like 3G accelerates the migration towards smart phones. But still battery power and frequent change of handsets is still a constraint. They burden on user had to manually synchronize their contacts, applications they use to the new phones. Also they loss whatever they are doing when the mobile get power down. In this paper, we propose a solution to the problem discussed with a new fault tolerant platform which can provide application mobility across the devices.**

Keywords- Fault tolerance; Middle ware; Midstore Manager.

I. INTRODUCTION

Application mobility refers to the idea application can run across multiple devices seamlessly even if it fails on one device say to battery down or users move to some other device. The application does not stop and it continues doing its work across user's movement to devices. We choose smart phones as the device and laptops as the devices to consider for application mobility. Application can be anything from word editing, preparing the power point presentations etc.

We explain the case for application mobility with some scenarios. Say a user is a browsing a web page in his smartphone, his battery power is low, can he work with he work with the same page in browser in laptop automatically, or he has another phone and want to work with same page in the handheld. Say the user is watching a you tube video, he is at 2 min another 3 min video is pending, he wants to watch this video on his tablet from the 2min or his phone switches off and he powers again, can he still watch the video from 2min automatically. Currently there is no solution available in market for these problems which motivated our work. In this paper, we address these challenges and design a software solution these problems.

II. RELATED WORK

Fault Tolerance solutions usually are designed for production servers like air traffic control, distributed disaster system, railways reservation system, internet banking where a single fault may lead to huge loss of money and even human lives. Replication based technique is one of the popular fault tolerance techniques [1]. A replica means multiple copies. Replication is a process of maintaining different copies of a data item or object. In replication techniques, request from client is forwarded to one of replica among a set of replicas. This technique is used for request that do not modify state of service. Replication adds redundancy in system. In this way failure of some nodes will not result in failure in system and thus fault tolerance is achieved.

Checkpoint with rollback-recovery is a well-known technique. Checkpoint is an operation which stores the current state of computation in stable storage. Checkpoints are established during the normal execution of a program periodically. This information is saved on a stable storage so that it can be used in case of node failures. The information includes the process state, its environment, the value of registers, etc. When an error is detected, the process is roll backed to the last saved state [2].

Although replication method is widely used as a fault tolerance technique but number of backups is a main drawback. Number of backups increases drastically as coverage against number of faults increases. As the number of backup increases management of these backups is very costly. Fusion based techniques overcome this problem. It is emerging as a popular technique to handle multiple faults. Basically it is an alternate idea for fault tolerance that requires fewer backup machines than replication based approaches. In fusion based fault tolerance a technique, back up machines is used which cross product of original as fusions is corresponding to the given set of machines [3]. Overhead in fusion based techniques is very high during recovery from faults. Hence this technique is acceptable if probability of fault is low.

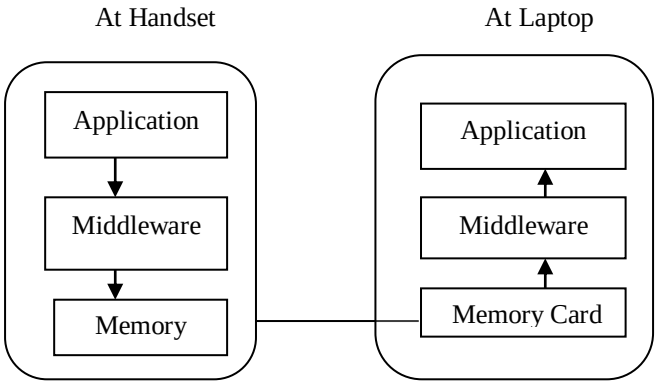
All these fault tolerance solutions address only for the server systems. We cannot use these solutions directly for our problem; all these solutions are designed for applications running in homogenous platform and the platform on which application run are same across devices.

But in our problem we need to work across devices with different platform. We need to provide application mobility for application running in Android platform to the Application running in Symbian platform or to windows on a laptop. So the fault tolerance solution for application mobility becomes even more complex.

III. PROPOSED SOLUTION

The proposed solution consists of designing a middleware. The middle ware will provide fault tolerant application mobility. The middleware provides API's for application to synchronize essential information.

The middleware can write the synchronization information into a memory card and it can also read this synchronization information and start the applications. We could have used cloud storage for synchronization of information, but relaying on cloud will sometimes become a problem like network not available. Using memory card has its advantages. The read and write operations for synchronization becomes very much faster. The memory card can be used to stat the application seamlessly on another laptop or other handheld easily, just insert the memory card and start the middleware.



So proposed system now standardizes the protocols and the standards used for Application Mobility.

A. Middleware

There are two approaches for the design of middleware. In first case, the application has to manually call the API's to set the synchronization information. In the next case, middle ware is dynamic and abstract the device API's.

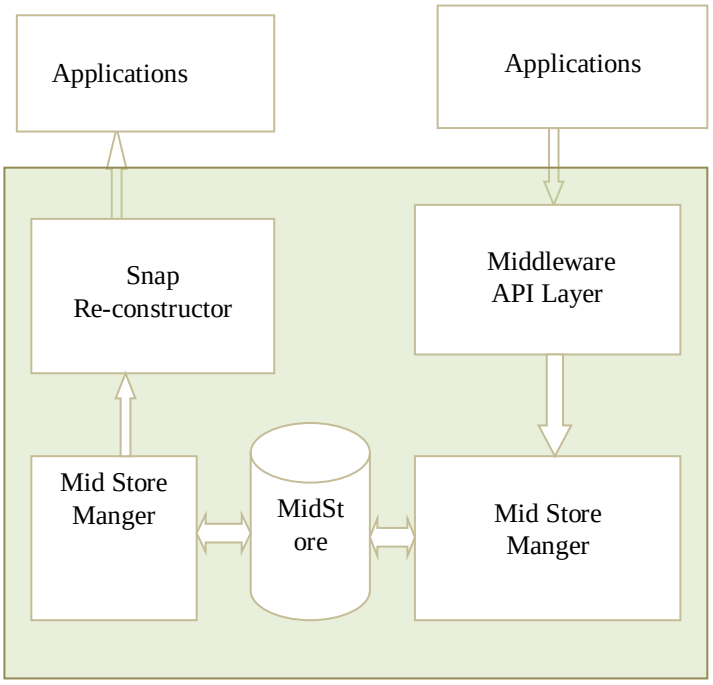
In this work , we limit our scope to application has to manually call the middle's API.

Middleware API's should take a minimal amount of time for providing the fault tolerance without affecting the application performance. Poor design of middleware will result in lower application performance and will seriously impact user experience. Middle ware records the session activity in a XML format in the Memory Card.

Since the Memory Card is also used by user for his contents, we should provide secure and incorruptible way for our XML. This can be provided by creating a folder in the Memory card for the purpose of middleware alone and using a virtual file system. These way middleware operations are assured a separate workspace in the Card. This virtual file system is referred as MidStore from here on.

Applications uses the middleware API's to record the current application session activity into the MidStore. At any point of time, the information in the middleware provides the snapshot of mobile user's current activity. The information is MidStore has this snapshot. Our job of Application mobility in case of fault tolerance is very easy, if we can reconstruct this snap on any other device or platform. To make this easy, first step is standardizing the format of recording the snapshot. We use XML standard for this purpose. XML is well standardized way to share data across cross platforms.

Middleware must also provide a reconstruct job as service which can invoked on the other platform to reconstruct the snapshot. Based on the discussion above, so far we formulate the architecture of middleware as



The core of our proposed solution is in the Middleware App Layer, Mid Store Manager, MidStore and Snap Re constructor.

B. Middleware App Layer

This layer provides the API's for the application to store the session activity in the MidStore. Say the Application is browser on the mobile handset. User has entered a URL and got the webpage. This activity must be recorded into the midstore.

Say the Middleware provides the API like below.

RecordActivity (Application, DescritorParam);

The Browser Application will call RecordActivity with the parameters filled as

Application: IE

DescriptorParam: www.yahoo.com

Once this API call is made, middle API Layer delegates the API to the MidStore manager.

We care not addressing any specific implementation of API's in this paper. But as a general guideline, the API should give the following information about user activity to the Middleware platform. It should provide the activity is related to which application and the additional descriptor of the activity.

If the user is browsing a webpage with internet explorer, the application is IE and the descriptor is URL for browsing. If the user is playing a song with media player, the application is WMP and the descriptor is the song name.

C. MidStore Manager

As discussed previously the memory card have a virtual file system called as MidStore. In order to manage the read and write to the MidStore we need the MidStore manager.

MidStore Manger handles the api call from the APP Layer and records this activity in the MidStore. It is upto the specific implementation of Mid Store Manager , to record all the activity in a single XML file or make separate XML file for each application. The case of making separate XML for each application has a particular advantage during reconstruct process. This will be addressed in the Snap re constructor.

We only suggest a way for the XML format, but it is upto the specific implementers to have their own way

```
<midstore>
  <application>
   < instance = "IE"/>
  < descriptor url =www.yahoo.com />
  </application>
</midstore>
```

Suppose the User have closed the browsing application , the browser should call an API

RemoveInstance(Application , Descriptor)

With Application value as IE.

When this api is called, the midstore manager must remove the application entries for IE.

D. SnapReConstructor

SnapReConstructor is an important module in our design. It reads the midstore and replicates the user activity on any target platform.
Suppose the mid store has the following

```
<midstore>
  <application>
   < instance = "IE"/>
  < descriptor url =www.yahoo.com />
  </application>
</midstore>
```

And reconstuctor is running on desktop system with windows. Then by reading this snap the reconstuctor should be able to open the IE browser with url as www.yahoo.com. SnapReConstructor can also have additional logic called as ApplicationAliter.

ApplicationAliter can help partial replication in target platform. Say IE is not in the target platform, but it's found out that Mozilla is installed in the target. Since both IE and Mozilla are web browsers, SnapReConstructor can start the Mozilla with url as www.yahoo.com. So this guarantees partial restoration.

SnapReConstructor must always try to achieve maximum restoration without user intervention. Instead of SnapReConstructor to be provided as an application and user always click to start , it can be made auto start whenever the device is introduced into target platform. But we leave this to implementers choice.

For any new application to be supported by the middleware, we need to extend the middleware API layer for providing parameters and SnapReConstructor must be extended to start the application or must ApplicationAliter must be extended for doing partial reconstruction.

Instead of application reading each application tag in the MidStore , if it can get all instances for that specific application, it can start the application at one start with all instances. We find this very useful in certain targets for some applications. In windows, stating a IE can be done with multiple url , so one ie instance comes up with different tabs for each url.

Not all application available in mobile platform is available for desktop platform. Also not all applications are available across different mobile platform. But popular applications for consumer user and the enterprise application are moving towards availability in all platforms. Also with the use of ApplicationAliter we can always find alternative for the applications.

E. Application of proposed Platform

In this section we detail the possible usage scenarios for our project.

A user adds contacts to his phone. He adds as many contacts to his phone. Since mobile phones are cheaper and new model comes to market every day, and in rapid mobile use in countries like India , people often change phone , so every time they change they have sync with contacts , important messages , calendar events etc , but with our platform in place , the user can just change the handset and connect to his computer, the platform will immediately get the application information like messages , calendar events etc applications continues in the new handset.

F. Platform against Battery Backup

Modern smart phones even claim a battery backup of 10 days or so, but for busy business users, they see that battery does not last for even 2 days. Many times business user has to carry another phone. He has to continue his operation using the new handset, but from the same point of continuity.

Currently there is no solution for this problem and the transitory phase for business user from handset to handset is a very bad experience, but with our platform the transitory is smooth for the user, with the application snapshot same as he worked previously.

Since our platform is not so complex and uses less resources it can even run with low end platforms.

IV. CONCLUSION

In this paper work, we have provided a solution for application fault tolerance and mobility across different platforms. Any Implementation can use the solution to realize it on different platforms. The Application mobility will be of great advantage to the enterprise and consumer applications. The application has to change a bit to save the user activity into the MidStore. This will be a disadvantage for the existing applications. In Android platform , with the concept of Broadcast receiver, any application can watch for certain events , so browsing a url , playing a song etc are all events , so a watcher application cab be written to watch for the events and record user activity to the MidStore. We are looking for the same kind of solutions on other platforms too. Once we are able to find the generic watcher solution, the Watcher

component can also be added to the middleware. This will be a big advantage for providing mobility for existing applications.

REFERENCES

[1] Bhargava B. and Lian S. R., "Independent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems-An Optimistic Approach," Proceedings of 17th IEEE Symposium on Reliable Distributed Systems, pp. 3-12, 1988.

[2] Cao G. and Singhal M., "On coordinated checkpointing in Distributed Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 9, no.12, pp. 1213-1225, Dec 1998.

[3] V. Agarwal, Fault Tolerance in Distributed Systems, I. Institute of Technology Kanpur, www.cse.iitk.ac.in/report-repository, 2004. ,

[4] "XML Media Types, RFC 3023". IETF. 2001-01. pp. 9–11. http://tools.ietf.org/html/rfc3023#section-3.2. Retrieved 2010-01-04.^ "XML Media Types, RFC 3023".

[5] Adnan Agbaria, William H. Sanders, " Distributed Snapshots for Mobile Computing Systems", Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications](Percom'04), pp. 1-10, 2004.

[6] Parveen Kumar, Lalit Kumar, R K Chauhan, "A low overhead Non-intrusive Hybrid Synchronous checkpointing protocol for mobile systems", Journal of Multidisciplinary Engineering Technologies,Vol.1, No. 1, pp 40-50, 2005.

[7] Parveen Kumar, Lalit Kumar, R K Chauhan, "Synchronous Checkpointing Protocols for Mobile Distributed Systems: A Comparative Study", International Journal of information and computing science, Volume 8, No.2, 2005, pp 14-21

[8] "XML Serialization in the .NET Framework". Msdn.microsoft.com. http://msdn.microsoft.com/en-us/library/ms950721.aspx. Retrieved 2009-07-31

[9] V.K Garg,. "Implementing fault-tolerant services using fused state machines," Tech-nical Report ECE-PDS-2010-001, Parallel and Distributed Systems Laboratory,ECE Dept. University of Texas at Austin (2010).

[10] Extensible Markup Language (XML) 1.1 (Second Edition)".W3.org. http://www.w3.org/TR/xml11/#charsets. Retrieved 2010-08-22.IETF. 2001-01. pp. 7–9. http://tools.ietf.org/html/rfc3023#section-3.1. Retrieved 2010-01-04.

[11] M. Murata, D. Kohn, and C. Lilley (2009-09-24). "Internet Drafts: XML Media Types". IETF. http://tools.ietf.org/html/draft-murata-kohn-lilley-xml-03. Retrieved 2010-06-10.

[12] "XML 1.0 Specification". W3.org. http://www.w3.org/TR/REC-xml. Retrieved 2010-08-22.

[13] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso," Understanding Replication in Databases and Distributed Systems," Research supported by EPFLETHZ DRAGON project and OFES).

[14] Checkpoint-based Fault-tolerant Infrastructure for Virtualized Service Providers.

[15] A Review of Checkpointing Fault Tolerance Techniques in Distributed Mobile Systems.