# Snapshot Service Interface (SSI)

A generic snapshot assisted backup framework for Linux

Faraz Shaikh
Calsoft Private Limited.
Baner Road, Pune- 411045, India
91-20-39852900
faraz.shaikh@gmail.com

Zoheb Shivani
Pune Institute of Computer Technology,
Dhankavadi, Pune 411043, India
020-2437-1101
zohebshivani@yahoo.co.in

Abstract— This paper presents the design and implementation of *"Snapshot Service Interface-SSI*", a standardized backup framework for the Linux platform. Linux is a prominent candidate for using such a backup framework because of its ubiquitous nature on the high-end server market. Such a backup framework is introduced in Windows 2003 under the name of volume shadow copy service VSS [1].

SSI being a first of its kind backup solution for the Linux platform, the main contribution of this paper is to discuss the pros and cons of different design alternatives available on Linux. The paper also provides a quantitative measure on performance hits incurred due to using such framework in lieu of using traditional backup methods.

## 1. Introduction

Currently backups on Linux are taken using application agents, open file managers and other propriety backup protocols. At present there exists no backup solution on the Linux Platform that can guarantee snapshot assisted consistent online backups. A consistent backup set for us is both application as well as file system level consistent.

The current backup solutions face problems like the infamous Backup Window phenomenon, Open files, Multiple API problem and File-level and Application-level inconsistencies [1]. The primary reason for all these problems is the lack of coordination between the different actors related to backup. Solutions like Open file Managers and Application agents tend to eliminate the above-mentioned problems. However, each one has some shortcomings associated with it. In short, none of these is a complete backup solution.

As a solution, we propose SSI, a common unified backup framework that can accommodate a myriad of backup *actors* viz. business applications, snapshot providers and backup applications. We have generalized the operations of *actors* using common interfaces and the interaction between them during a snapshot assisted backup operation. The actors get registered with SSI by providing implementations for the generalized interfaces. Once the actors have registered, SSI can take automated backups by coordinating the operations of all the actors using their registered callbacks

From empirical evidences and insights from user acceptability of VSS, we have concluded even backup frameworks like SSI and VSS are not free from shortcomings. These shortcomings are also discussed at length in [2]. The principal disadvantage of backup frameworks is that the actors should be aware of the framework. Making the actors aware of the backup framework requires making changes to their existing code base. To alleviate the problem, SSI's design provides for ways in which the applications can be made aware of the framework without modifications or at most minimal alterations. Also, the alterations are such that they can be easily incorporated in the applications. The gravity of the disadvantage is also alleviated due to the open-source nature of Linux and its applications. Availability of source code of all the actors makes it easy to adapt them to the SSI framework.

SSI is targeted at high-end enterprise servers hosting critical services like mail servers & data base servers which demand a 24x7 availability. In such an environment downtime for backups or otherwise is not acceptable and this mandates an online consistent backup.

## 2. Architecture

The architecture of SSI is as shown in figure 1. The figure is marked with the steps explaining the snapshot assisted backup process. The flow of operation during a backup is as follows:

1 The operation starts with backup application requesting SSI for a snapshot-assisted backup.

2 The main objective of SSI is to ensure application's consistency at the time the snapshot operation is performed. As a result, SSI asks all the registered business applications to get consistent and waits until all of them respond. Here, it may happen that a business application does not reply, resulting in SSI waiting indefinitely for the application to respond. To

avoid the condition of SSI waiting indefinitely, a timeout mechanism is implemented in SSI.

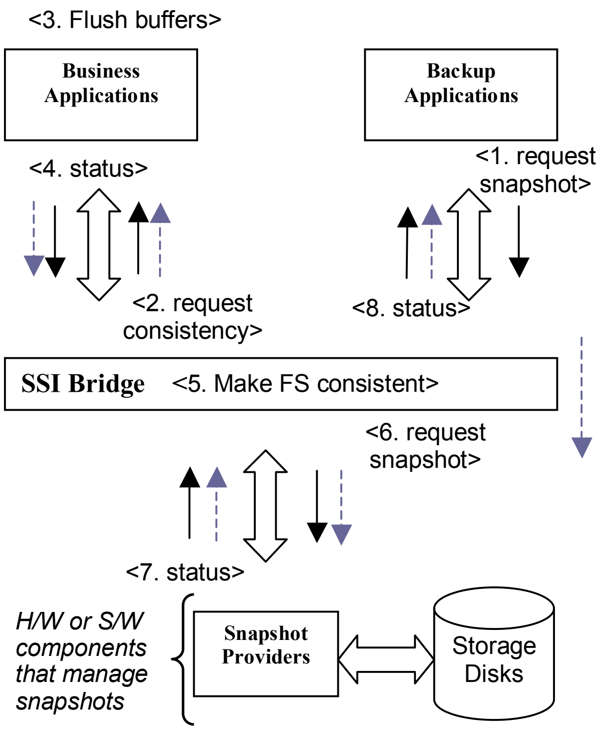

Figure 1: Architecture of SSI

3 The business applications make themselves consistent by flushing their buffers.

4 After the application is made consistent, it returns the status to SSI. Now, the application is paused until the snapshot operation is performed. A timeout mechanism is also maintained within the business applications to guarantee that the applications need not to wait forever in case SSI fails to respond.

5 The file system is now made consistent and writes to the disk are stopped. (Currently handled by the snapshot providers)

6 The snapshot provider is then requested to perform the snapshot. Since everything on the storage stack viz. applications and the file system are consistent, a consistent volume level snapshot takes place and a consistent backup can be taken out of it.

7 The snapshot provider returns the status of the snapshot operation to SSI. If the snapshot is successful, a positive status is returned. In case the snapshot does not succeed, a negative status is returned.

8 SSI in turn returns the status to the backup application.

9 At this point the file system writes are resumed and all the business applications are signaled to continue.

## 3. Implementation

### *3.1 Interfaces provided by the framework*

The key issue in implementing such a system is handling the communication between the Backup Framework and the multitude of applications. Carefully chosen communication protocols would ensure minimum interference to existing applications.

A common and simple approach could be to alter each actor explicitly to support SSI. But that would hamper the flexibility and efficiency of the backup solution. Also this will require a considerable amount of code alteration to be done to the existing applications, which is not feasible.

An alternative solution could be, the backup framework exporting an interface for each of the actor viz. business applications, backup applications and snapshot providers. An interface would facilitate communication between the Backup Framework and the corresponding actor. In SSI, the interfaces are implemented in the form of static libraries. There are three libraries corresponding to the three actors. An actor simply needs to compile itself with the library and use the functions to communicate with SSI.

Each of these interfaces comprise of functions that register and un-register an actor. Besides this, the interface corresponding to the business application also consists of an operation named *i_am_consistent()*. This function is called by the business application when it gets consistent. The function implements the code that makes the application to wait until the framework responds.

Similarly, the interface related to the backup application encompasses functions related to snapshot like *take snapshot*, *delete snapshot* and *reset snapshot*. It also provides methods to *list snapshot providers* and *list SSI aware business applications* to the backup applications.

### *3.2 Making business application consistent:*

Before the snapshot is taken the business applications have to get into a consistent state and then hold their writes until the snapshot is taken. These applications need to be notified when to get consistent and when to thaw writes. Therefore the business applications need to export an interface through which the Backup Framework can control these activities.

The business applications can make use of sockets for this reverse communication. Each business application provides with a socket at the time of registration. A thread is implemented by each business application that waits for a message to come on that socket. This thread implements the code related to consistency. Thus, the framework can make a business application consistent by passing appropriate message at the corresponding socket. Since sockets can be implemented in any language, the mechanism becomes language independent. But, the overhead with this approach is implementation of an extra thread associated with each business application.

To conquer this deficiency signals could be used. With this approach, each business application implements a signal handler to make it consistent. The signal number is notified to the framework during registration. The framework sends a signal to the business application to make it consistent. The only problem with using signals is that barely few programming languages like C/C++ support signals. Nevertheless, majority of servers are implemented in C/C++. Thus, using signals the mechanism can be implemented easily and efficiently. This made us choose signals in our solution.

### *3.3 Communication with Snapshot Provider:*

The main objective of communicating with the snapshot providers is that the Backup Framework needs the services provided by them. Different snapshot providers expose different interfaces. The Backup Framework should be made aware of these interfaces.

This goal can be achieved using callbacks. Thus, the snapshot provider at the time of registration supplies a set of functions to the framework. The framework maintains the structure in the database and uses the functions to perform snapshot operations. The major drawback with using callbacks is that, the framework needs to maintain the database.

Ideally all the snapshot providers are supposed to export a common interface. But, this needs to be achieved without making any changes to the snapshot providers. Changes to the snapshot providers need to be specifically avoided in the light of most of them being critical pieces of code (most of them kernel mode).

A way out of this dilemma is that the Backup Framework would request a separate entity to export an interface that it could use for communicating with snapshot provider. This entity would forward the snapshot requests to the appropriate snapshot provider. SSI requests all snapshot providers to create this entity, which is nothing but a dynamic linked library. This DLL written purely in user mode then becomes the bridge between snapshot providers and SSI.

Thus, existing snapshot providers can easily be integrated with the Backup Framework by simply creating a DLL that uses the services provided by snapshot providers and registers the snapshot provider with the Backup Framework.

### *3.4 Timeout Mechanism:*

As mentioned in the previous section, the framework supports a timeout mechanism. Whenever the framework signals the business applications to get consistent, a timer is started. If all the business applications do not respond within the defined time, the snapshot operation is abandoned. The timeout associated with the framework is configurable. Thus, it is ensured that the framework will never hang. The mechanism is implemented using *alarm* function available on Linux.

The business application also has a timeout associated with it. When a business application receives a signal from the framework for consistency, it makes itself consistent and starts the timer using *alarm*. The business application now pauses itself until it receives the status from the framework. If within the predefined time, the framework does not respond, the application thaws and backup is aborted.

## 4. Performance Results

The business applications tend to continuously write data. When a snapshot operation is requested by a backup application to the SSI framework, the SSI framework requests all the registered business applications to flush their buffers and temporally suspend their writes until the snapshot is taken. This process involves some delay because of the inherent delay in communication between the SSI framework and the business applications and also the time taken by the business applications to flush their buffers.

The tests were conducted on a standard PC with a Pentium 4 2.4 GHz processor and 256 MB of physical memory. A typical configuration of Linux, Fedora Core 3 provided the operating environment. The sample backup application that comes along with the SSI framework was used. For the business applications, one instance of MySql server and other instances of the sample business application provided with the SSI framework were used. MySql v5.0.17 was upgraded to support SSI framework. Device Mapper, that comes integrated with the 2.6.x Linux kernels, was used as the snapshot provider for the system.

| No. Of Business Applications | Time to write 1000 records without SSI intervention (Rdtsc values) | Time to write 1000 records with SSI intervention (Rdtsc values) |
|---|---|---|
| 1 | 5726900549 | 6364583915 |
| 3 | 5725800942 | 6594063443 |
| 7 | 5727397361 | 6990746543 |
| 15 | 5724200942 | 8488965541 |
| 25 | 5724984906 | 10955689559 |
| 50 | 5727548963 | 19435875936 |
| 100 | 5724156987 | 35574896562 |

Table 1: Delay introduced due to SSI as number of business applications are added

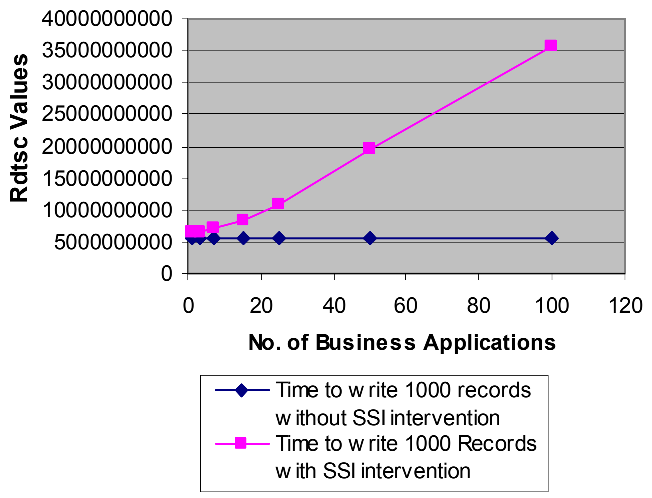

Figure 2: Delay introduced in the business applications due to SSI Intervention

Now, while writing records to a MySql table if a snapshot is requested by a backup application there would be some delay and the time taken to write those records would increase. This delay as mentioned above is due to the communication between the business applications and the SSI framework. As the number of applications increase this delay will increase and thus the time to write the records. As expected this can be observed in the following table and graph (Table 1 and Fig. 2).

The number of records chosen for the test was 1000. Since the time taken to write 1000 arbitrary records in a sample MySql table is very small, the timing measurements were made using Rdtsc (read time-stamp counter)[6], a facility provided by the Intel Pentium Processors. This time-stamp counter is simply a register which is incremented every clock cycle. Thus by placing the code to write the records between two reads of this time-stamp counter it is possible to count the number of processor cycles taken to execute that code. But, one problem faced while using rdtsc was that it measures the time delay inclusive of the time taken by other processes running in the system. As, we do not have a control over the operating system's scheduling; the timing measurements with dummy business applications were heavily fluctuating. The dummy business applications did not actually do IO to become consistent instead; they just signal that they are consistent. Therefore, to stabilize these fluctuations a sleep of 2 seconds was inserted as a token of time taken for real business applications to become consistent. This caused the rdtsc values that measured the time taken to write records, to increase tremendously which in turn minimized the fluctuations. The 2 second constant added to all readings heavily overshadowed the fluctuations in the rdtsc values due to number of processes being scheduled between readings. The final effect of the 2-second delay in the dummy business applications as a token of time required to become consistent is that of making the timescale coarser. The coarse timeline hides the effects of the scheduling fluctuations. This can be seen clearly in figure 2.

As observed above we can see that as the number of business applications increase, the delay due to SSI intervention increases linearly. This is logical since as more and more business applications pour into the system, the SSI framework would have to make sure that all those applications are consistent before the snapshot is taken. Thus, it would have to send every registered business application a signal to get consistent and then wait for the responses from all of them. Thus we can conclude that performance degradation of writes due to intervention by the SSI framework while taking snapshots is linearly dependent on the number of business applications registered with the framework.

## 5. Comparison with VSS.

VSS is a dominant backup framework for the windows systems and is fast gaining popularity. SSI does not provide all of the advanced features provided by VSS. Some of these advanced features are extremely useful and must be included in SSI in the near future. Top of this list of useful features are the concept of "shim writers"[7] and "snapshots for shared folders". Shim writers are fake business applications that run as services and assist in consistently backing up the operating system state in accordance with VSS framework. Shim writers would enable SSI to ensure consistency of operating system critical data like the system log files, service configuration files, boot configuration information, driver configuration information etc during backups. A shim writers' Contribution will allow backups of operating system state using the standard VSS framework.

"Snapshots for shared folders"[8] is a feature using which clients of file server can ask for previous version of a file on the file server with no system administrator intervention. Windows Server 2003 (win2k3) acting as a file server is configured to take periodic snapshots for volumes hosting filer shares using the windows job scheduling framework. The server maintains a predefined number of snapshots per volume. The client can then ask for any of the previous version of files on the file server, for which the snapshots have been taken. The windows clients can then access the pervious versions of a shared file using a modified CIFS client which issues special SMB commands understood by the CIFS server on the filer (win2k3). This is a neat feature which simplifies the shared file restoration and recovery process very simple, which would otherwise have required system administrators intervention at the file server side.

## Conclusions

In this paper, we have presented design alternatives for implementing a Backup Framework for the Linux platform. We also discussed the most appropriate design options as of current picture. The performance results show that degradation of writes due to intervention by the SSI framework while taking snapshots is linearly dependent on the number of business applications registered with the framework. This is not a major concern since the hardware configuration of servers running higher number of business applications is usually better. The major gain of using such a framework is that the business applications can continuously write data while their snapshot assisted backup is taken and the time taken for creating such a snapshot does not depend of the amount of data to be backed up. The presence of the web is making it mandatory for the business applications to run continuously and such a framework is definitely a must have for ensuring consistent and uninterrupted backups. The code and other resources are available for download at www.geocities.com/zohebshivani.

## REFERENCES

[1] Dilip Naik, “On Backup and restore technologies”, in Inside windows storage.

[2] Gary Stowell, “On the pros and cons of Microsoft’s Volume Shadow-Copy Service”.

[3] Device mapper manual pages at http://sources.redhat.com/cgi-bin/cvsweb.cgi/~checkout~/devicemapper/man/

[4] LVM2 manual pages at http://sourceware.org/lvm2/

[5] Kurt Seifried, “On various backup schemes” at http://www.seifried.org/lasg/backups/

[6] Intel. Using the rdtsc instruction for performance monitoring. Intel, 1997.
http://www.cs.usfca.edu/~cruse/cs210/rdtscpm1-1.pdf

[7] VSS SDK *7.1* Documentation. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vss/base/vss_portal.asp

[8] “Snapshots for shared folders” – HP solutions with VSS.
http://h71028.www7.hp.com/ERC/downloads/5982-6827EN.pdf