IMPLEMENTING META DATA SERVER AND LOAD BALANCING IN DISTRIBUTED CLOUD ENVIRONMENT

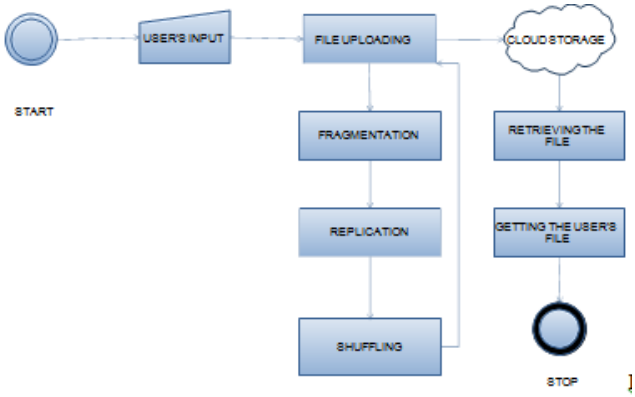
M.Akhila[1] | R.Sujitha[2]
[1](UG Student, Christ the King Engineering College, akimicheal@gmail.com)
[2](Assistant Professor, Christ the King Engineering College, srisuji14@gmail.com)

Abstract— *The Large class of modern distributed file systems treats metadata services as an independent system component, separately from data servers. The availability of the metadata service is key to the availability of the overall system. Given the high rates of failures observed in large-scale data centres, distributed file systems usually incorporate high- availability features. A typical approach in the development of distributed file systems is to design and develop metadata services from the ground up, at significant cost in terms of complexity and time, often leading to functional shortcomings. Our motivation in this paper was to improve on this state of things by defining a general-purpose architecture for HA metadata services that can be easily incorporated and reused in new or existing file systems, reducing development time. This project is developed using data replication with Meta server, it acquires more memory and therefore the data's won't be interact during a secure approach & not in ordered order .To overcome these drawback we elect the formula of Fragment and Snuffle algorithm. To overcome this problem we choose the Algorithm of FS The scope of this project is the Detach & reproduces methodology; we divide a file into fragments, and replicate the fragmented data over the cloud nodes. Each of the nodes stores only a single fragment of a particular data file that ensures that even in case of a successful attack, no meaningful information is revealed to the attacker.*

Keywords— Distributed File Systems; High Availability; System Recovery; Metadata Services; Fragment and Snuff

1. INTRODUCTION

The availability of the metadata service is key to the availability of the overall system: If clients cannot contact the metadata service, they only have limited (e.g., over the duration of a lease) or no access to the entire data set. To ensure high availability of the metadata service, systems must be able to handle failure. In large-scale systems, failures tend to be the norm rather than the exception and can be caused by either planned or unplanned events including hardware failures, software bugs,reboots, software updates and maintenance. Our implementation of the RMS architecture relies on three interoperating layered components: a highly1045-9219 .To make our approach practical, we base it on an existing replicated database exposing a keyvalue API, Oracle Berkeley DB (or BDB).Our experience with HDFS, a system that was not originally implemented over BDB, shows that it is possible to retrofit our solution into the metadata server with a reasonable level of complexity, achieving high availability as well as larger file system sizes than main- memory permits. A typical approach in the development of distributed file systems is to design and develop metadata services from the ground up, at significant cost in terms of complexity and time, often leading to functional shortcomings. Our motivation in this paper was to improve on this state of things by defining a general-purpose architecture for HA metadata services that can be easily incorporated and reused in new or existing file systems, reducing development time. This project is developed using data replication with Meta server, it acquires more memory and therefore the data's won't be interact during a secure approach & not in ordered order. To overcome these drawback we elect the formula of Fragment and Snuffle algorithm. To overcome this problem we choose the Algorithm of FS The scope of this project is the Detach & reproduces methodology; we divide a file into fragments, and replicate the fragmented data over the cloud nodes.



Each of the nodes stores only a single fragment of a particular data file that ensures that even in case of a successful attack, no meaningful information is revealed to the attacker. Modern large- scale distributed and parallel file systems such as PVFS, HDFS, GoogleFS, pNFS, and Ceph treat metadata services as an independent system component, separately from data servers (Figure 1). Two reasons behind this separation are design simplicity and the ability to scale the two parts of the system independently. The availability of the metadata service is key to the availability of the overall system: If clients cannot contact the metadata service, they only have limited (e.g., over the duration of a lease) or no access to the entire data set. To ensure high availability of the metadata service, systems must be able to handle failure. In large-scale systems, failures tend to be the norm rather than the exception and

can be caused by either planned or unplanned events including hardware failures, software bugs, reboots, software updates and maintenance. The high rate of failures in conjunction with the constant changes in modern data centers, typically call for replication as a standard method to implement highly available metadata services. Prominent distributed file systems, such as the Parallel Virtual File System1 (PVFS) and the Hadoop File System (HDFS), already offer highly available (HA) metadata services. PVFS uses stateless replication with multiple metadata servers over a shared network-accessible storage service, such as NFS, for storing file system metadata. A drawback of such a solution is the single point of failure posed by the shared storage server. HDFS version 2.x avoids this problem by using stateful replication over quorum based replicated storage.

2. PROBLEM DESCRIPTION

The prime disadvantage is security. To ensure security, cryptographic techniques cannot be directly adopted. Sometimes the cloud service provider may hide the data corruptions to maintain the reputation. To avoid this problem, we introduce an effective third party auditor to audit the user's outsourced data when needed.We are not the only researchers to have investigated software diversity for ROP attack mitigation. First, the software diversification is not done frequently enough. Second, some of the existing defenses require the source code or other additional information that is not usually available. Third, the randomization is not fine grained enough leaving large code chunks unrandomized. Fourth, significant runtime overhead is incurred throughout the runtime of the application by introducing additional data structures. Marlin addresses these limitations and provides a strong and efficient defense technique against ROP attacks.With any solution, there are always costs that must also be considered. In our proposed scheme, there is a performance impact when the process begins. We have evaluated the time to randomize compiled binaries on a selection of commonly used applications and Linux coreutils, showing that the performance penalty for Marlin is reasonable in the average case. Thus, our work demonstrates that, although Marlin imposes certain performance costs, its success in thwarting ROP attacks makes this a feasible approach for systems that prioritize execution integrity over optimal performance. In Section 3 we describe techniques for minimizing this performance impact. For instance, performing the randomization during offline pre-processing significantly reduces the startup costs.

3. IMPLEMENTATION

A Systems Development Life Cycle (SDLC) adheres to important phases that are essential for developers, such as planning, analysis, design, and implementation, and are explained in the section below. A number of system development life cycle (SDLC) models have been created: waterfall, fountain, and spiral, build and fix, rapid prototyping, incremental, and synchronize and stabilize. The oldest of these, and the best known, is the waterfall model: a sequence of stages in which the output of each stage becomes the input for the next.

3.1 User Authentication

User Authentication is the process of identity verification you are trying to prove a user is who they say they are. For a user to prove their identity, a user needs to provide some sort of proof of identity that your system understands and trust. The authentication process starts with creating an instance of the Login Context. Various constructors are available; the example uses the Login Context variety. The first parameter is the name (which acts as the index to the login module stack configured in the configuration file), and the second parameter is a callback handler used for passing login information to the Log server. Callback Handler has a handle method which transfers the required information to the Meta server/Log server.

3.2 Fragmentation

We are splitting the file in to small fragments. Once the file is split into fragments, this concept selects the cloud nodes for fragment placement. The selection is made by keeping an equal focus on both security and performance in terms of the access time. The process is repeated until all of the fragments are placed at the nodes. Partial Replication represents the fragment placement methodology. Mainly we focus on the storage system security in this work. As stated above, the probability of a successful coordinated attack is extremely minute

3.3 Data Replication and Data Encryption

This component supports the replication mechanisms by invoking replicas and managing their execution based on the client's requirements. We denote the set of VM instances that are controlled by a single implementation of a replication mechanism as a replica group. Each replica within a group can be uniquely identified, and a set of rules R that must be satisfied by a replica group are specified. The task of the replication manager is to make the client perceive a replica group as a single service, and to ensure that the fault free replicas exhibit correct behavior during execution time. To support a replication mechanism, the replica invoker first contemplates the desired replication parameters such as the style of replication (active, passive, cold passive, hot passive), number of replicas, and constraints on relative placement of individual replicas, and forms the replica group.

3.4 Server Analysis

The task of offering fault analysis as a service requires the service provider to realize generic fault analysis mechanisms such that the client's applications deployed in virtual machine instances can transparently obtain server status properties. To this aim, we define ft−unit as the fundamental module that applies a coherent server analysis mechanism to a recurrent system failure at the granularity of a VM instance. The notion of ft−unit is based on the observation that the impact of hardware failures on client's applications can be handled by applying server analysis mechanisms directly at the virtualization layer than the application itself For instance, server analysis of the banking service can be increased by replicating the entire VM instance in which its application tier is deployed on multiple physical nodes, and server crashes can be detected using well-known failure detection algorithms such as the

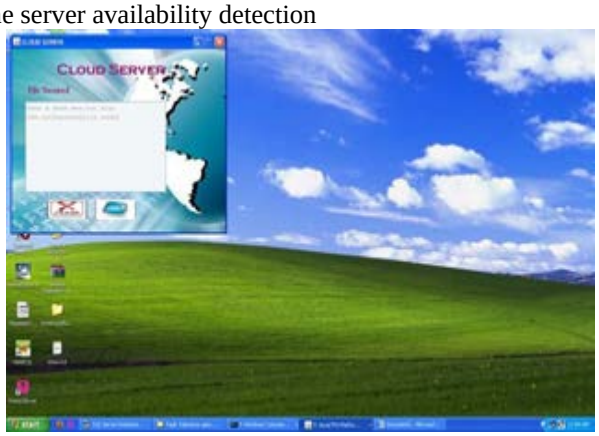
heartbeat protocol. The design stage starts when a client requests the service provider to offer server analysis support to its applications

3.5 Data Retrival & Decryption

This component supports the replication mechanisms by invoking replicas and managing their execution based on the client's requirements. We denote the set of VM instances that are controlled by a single implementation of a replication mechanism as a replica group. Each replica within a group can be uniquely identified, and a set of rules R that must be satisfied by a replica group are specified.

4. RESULTS

Output design generally refers to the results and information that are generated by the system for many end-users; output is the main reason for developing the system and the basis on which they evaluate the usefulness of the application. Computer output is the most important and direct source of information to the user. Output design is very important phase because the output will be in an interactive manner. The output are the fragmentations and the server availability detection



5. CONCLUSION AND FUTURE SCOPE

In this paper we proposed a general-purpose architecture of replicated metadata services in distributed file systems, named RMS. Our evaluation shows that an RMS variant of HDFS performs comparably to native implementations when contrasted in micro benchmarks. However, this has to come at the expense of durability: Write transactions with synchronous commits to disk are more expensive in HDFS-RMS (SYNC mode)HDFS-HA with similar behavior, pointing to the efficiencies possible in special-purpose software (HDFS-HA) compared to general-purpose designs (HDFS-RMS); one can improve HDFS-RMS performance to the levels of HDFS-HA by offering somewhat weaker durability semantics using the NOSYNC mode, which we consider as an acceptable tradeoff in replicated setups. Our experience with tuning our HDFS-RMS prototype implementation provides important lessons for RMS implementers: Avoid a schema that results into spreading of metadata onto a large number of small tables, and use underlying database implementations that allow for sufficient concurrency. In most cases, including application level benchmarks, RMS variants do not incur an end to-end performance penalty. In terms of availability, the RMS variant of HDFS matches the recovery characteristics of HDFS-HA v2.x, a state of the art implementation. HDFS-RMS is resilient to the loss of any number of Name Nodes (assuming sufficient number of replicas), whereas HDFSHA allows only up to two Name Nodes. Our results show that there would be a performance benefit if HDFS-RMS was able to maintain hot-spare Name Nodes at backup BDB replica nodes. To achieve this, it would have to capture updates communicated from master to backup BDB replicas (for example, by inspecting the BDB write-ahead log) and apply them to HDFS in-memory structures. When the master fails, recovery code at a backup would have to ensure that log updates are fully reflected at Name Node state prior to appointing it a master. Implementation of this functionality within HDFS is beyond the scope of this paper and subject of future work.

REFERENCES

[1] Stamatakis, D., Tsikoudis, N., Smyrnaki, O. and Magoutis, K.,"Scalability of Replicated Metadata Services in Distributed File Systems," in Proc. of 12th IFIP Int. Conference on Distributed Applications and Interoperable Systems (DAIS) 2012, Stockholm,Sweden, 2012.

[2] Ligon, M., Ross, R., "Overview of the Parallel Virtual FleSystem," in Proceedings of USENIX Extreme Linux Workshop, Monterey, CA, USA, 1999.

[3] Shvachko, K., Kuang, H., Radia, S. and Chansler, R., "The Hadoop Distributed File System," in Proc. of IEEE Conference on Mass Storage Systems and Technologies (MSST), Lake Tahoe,NV, 2010.

[4] Ghemawat, S., Gobioff, H. and Leung, S.-T., "The Google File System," in Proc. of 19th ACM Symposium on Operating Systems Principles (SOSP-19), Bolton Landing, New York, 2003.

[5] Shepler, S. et al., "Parallel NFS, RFC 5661-5664," http://tools.ietf.org/html/rfc5661, IETF.