Design of a FUSE based file system interface coupled with Stork Cloud

Shreyas Garg
CSE Dept. SUNY Buffalo

Archana Suregaonkar
CSE Dept. SUNY Buffalo

Subhadeep Karan
CSE Dept. SUNY Buffalo

I. Problem Description:

There has been a very long battle towards the optimal file systems on Linux. Many have been developed that give us different options and are being built into the kernel itself. But integrating a file system into the kernel space is a long and tedious task to undertake. Coding a file system into the kernel of a Linux environment generally means getting a more generic look out of the file system properties. One of such being mounting available to only root privileges. For the same purpose, we have a file system in user space called FUSE.

The main purpose of the current project is to link or rather build a bridge between the fuse file system and a cloud storage system that has access to the remote file servers and stores the file metadata. Through a virtual file system interface, the user would be able to access the cloud hosted services and parse the remote servers as conveniently as a local file system. Moreover, the file system's aim is to implement local caching upto some level and work on a prefetching mechanism. Prefetching here essentially means that when a request is sent out by the thin client, the file system should be capable of prefetching the directory structure from the response received through stork cloud and maintain a local copy of the same upto two levels.

II. Kernel Space and User Space:

Kernel space is indicated by the space something is happening within the kernel code. User space is the space where any user with privileges or not can make and change any application using system resources. Adding code in the user space is advantageous in many different ways. One of them is prevention of bringing down the whole kernel due to a bug presenting itself inside the

newly added code. Moreover, adding code to kernel as a programmer would agree a great pain.

If we are to use the user space to add some new applications, we are sure to get basically three advantages:

- Easy distribution of code, easy maintainability of code, limited testing and security concerns as compared to the code being in the kernel space.
- Easy fixing of bugs, updation of the application required as per need. If the application crashes, we just have to remount it.
- Unlike the kernel level file system. If a kernel file system crashes, it is likely that the entire OS crashed with the kernel panicking.

III. FUSE:

Since long, there has been difficulty trying to create a file system in user space where it can easily interact with the virtual file system in the kernel space. [2] Fuse is an operating system mechanism that allows users with a Linux like OS to feel like home using a file system. This is achieved by running a file system code in the user space with fuse providing a bridge to the kernel space.

Essentially, fuse is useful for implementing a virtual file system. Using this module. We can implement any resource available to export into a file system using fuse platform. The figure below demonstrates how this essentially works:

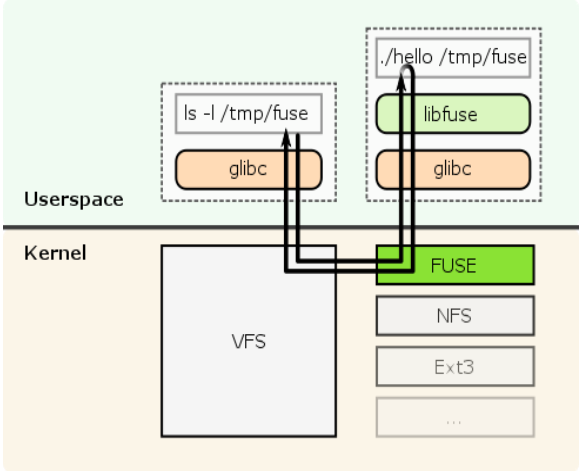

Figure 1

[1] The illustration is the hello work file system that comprises of about 100 lines of code. The sample snippet from the fuse website is:

```
~/fuse/example$ mkdir /tmp/fuse
~/fuse/example$./hello /tmp/fuse
~/fuse/example$ ls -l /tmp/fuse
total 0
-r--r--r--  1 root root 13 Jan  1
1970 hello
~/fuse/example$ cat /tmp/fuse/hello
Hello World!
~/fuse/example$ fusermount -u
/tmp/fuse
~/fuse/example$
```

Fuse helps provide a platform link between user space and the kernel. Features include:

- Simple library API
- Simple installation (no need to patch or recompile the kernel)
- Secure implementation
- User space - kernel interface is very efficient
- Usable by non-privileged users
- Runs on Linux kernels 2.4.X and 2.6.X
- Has proven very stable over time.

The aim of FUSE is to provide an interface or rather a way for limited privileges to interact with the kernel and has an API for easy access. [3]The best part of Fuse is the provided API. The API allows us to write file systems for the user space in any way we want. Moreover, FUSE has a vast pool of language binding. Some include:

- C
- C++
- JAVA
- Ruby
- Perl
- Python
- Many more..

Installation of FUSE is very simple and once that is done, we can see the example directory as described by the FUSE website, this is what we get from cat */tmp/fuse/hello*:

```
~/fuse/example>./hello /tmp/fuse -d unique:
2, opcode: LOOKUP (1), ino: 1, insize: 26
LOOKUP /hello
        INO: 2
        unique: 2, error: 0 (Success),
outsize: 72
```

unique: 3, opcode: OPEN (14), ino: 2,
insize: 24
unique: 3, error: 0 (Success),
outsize: 8
unique: 4, opcode: READ (15), ino: 2,
insize: 32
READ 4096 bytes from 0
READ 4096 bytes
unique: 4, error: 0 (Success),
outsize: 4104
unique: 0, opcode: RELEASE (18), ino: 2,
insize: 24

By default, the FUSE file systems are multithreaded and this can be tested by recursively entering mount point in fusexmp.

It is important to know how the FUSE bridges the user space and the kernel so: Whenever system I/O calls are made towards mounted resource, FUSE will capture these I/O calls in kernel and forward them to user space library called libFuse. This library will map local system I/O calls into remote storage I/O calls. The FUSE library is available in most Linux distributions today. It is a very practical way of implementing a user-level file system.

IV. Design Specifications:

This is a Web App which will work as a client for Stork Cloud. It will call Stork's Web Services using Rest APIs and then communicate the responses to the thin clients. This client will consume the RESTful APIs via the standard HTTP protocol.
Some of the features its first version will have would be Exporting virtual file system on client using FUSE library.

- Maintaining local cache.
- REST API connection.
- Querying server with user Authentication parameters (email = “EMAIL” and password = “PASS”)
- Support for CRUD operations (examples):
  - ✓ POST - Creating new directory (cp, mkdir)
  - ✓ GET - Returns list of directories (ls)
  - ✓ PUT - Create/Update Directory (cp , mv)
  - ✓ DELETE – Deleting file or directory (rm)
- Error Handling
- Parsing the Jason response and populating file meta-data on the client device.

A. Implementation details:

For the purpose of the implementation, we used Python and python-fuse along with all the different APIs that are available. Some of the basic ones includes FUSE and requests.

The basic idea is to cache the requested file or directory in the local client and check it every time a request is made. When a request is made to the stork cloud, it contacts the remote server and provides us with a JSON object to work with. Our implementation takes that JSON object, parses it and stores the file and directory metadata in the form of key value pairs. Key being the name of the file or the directory and the value storing the complete object that comes from the stork cloud. Whenever we have a request from the client, first the cache is checked for that key and JSON is parsed to give the result. If the key is not present in the cache, a request is sent over to the stork cloud. The implementation works on a configurable level of prefetching. At present, we prefetch a level below the desired directory and store the JSON object as is until requested upon.

Working with fuse API with python is essentially very simple with all the laid out structures of a file system and its operations. Currently, the implementation supports read only as supported by the stork cloud.

B. Design Overview:

The Fuse Interface between stork cloud and thin clients is a virtual file system interface that allows users to access the Cloud and parse remote storage server contents as convenient as accessing the local file system. It will enable mounting remote storage servers into the users' local host. This interface will allow non-root users to be able to mount remote file systems locally and it is based on FUSE. We are using FUSE library to develop the client side of a wide area file system. Our REST Client will query the Server APIs and obtain server's response. It will then appropriately parse the JSON response and populate the file metadata in the thin client devices. Client will do some level of caching and also make sure the cache is up to date with server by periodically querying the server. Client will handle all the exposed server's endpoints and also convey appropriate message to the thin clients in case of failure.

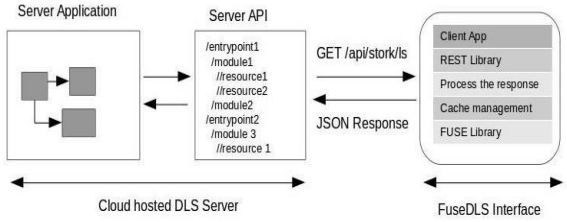

Figure 2

C. About Stork:

Stork is a batch scheduler specialized in data placement and data movement, which is based on the concept and ideal of making data placement a first class entity in a distributed computing environment. Stork understands the semantics and characteristics of data placement tasks and implements techniques specific to queuing, scheduling, and optimization of these type of tasks.

Stork acts like an I/O control system (IOCS) between the user applications and the underlying protocols and data storage servers. It provides complete modularity and extendibility. The users can add support for their favorite storage system, data transport protocol, or middleware very easily. If the transfer protocol specified in the job description file fails for some reason, Stork can automatically switch to any alternative protocols available between the same source and destination hosts and complete the transfer.

Stork can interact with higher level planners and workflow managers. This allows the users to schedule both CPU resources and storage resources together. Currently, some implementations of Condor DAGMan and Pegasus come with Stork support.

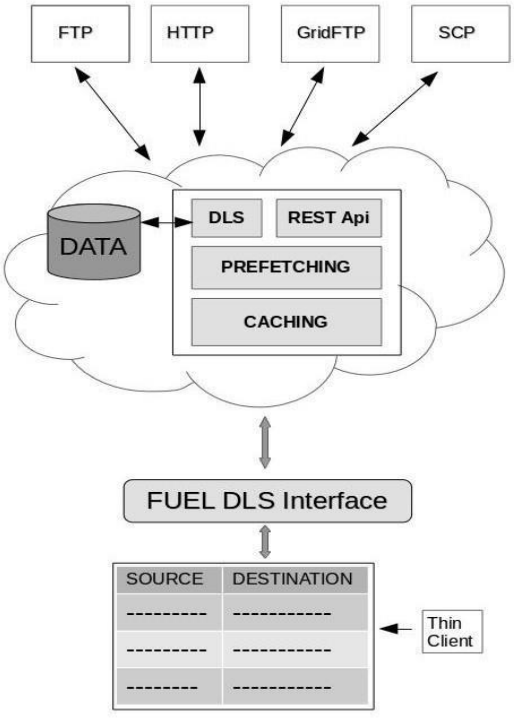

Figure 3

V. Challenges Encountered:

- Configuring and integrating FUSE with local file system in python.
- Analyzing directory/file permissions and storing in an understandable format by fuse.
- Pre-fetching multiple directory levels from the remote server.
- Downloading a file from the remote server
- Maintaining cache freshness
- Handling directory/file not found exceptions
- Executing copy posix command.

VI. Implementation Analysis:

1. **FUSE Integration**: This system uses latest version of FUSE- File System in User Interface in a Linux environment. The FUSE API comes in different flavors and we have used the Python version for our implementation.

- File system methods:
    1. def getattr(self, path, fh=None): This function is called to get the attributes of a specific file. Attributes are meta-information about the file that is typically stored in the i-node of the file or a similar structure. Similar to stat(). It is invoked when "ls" or "cd" command is executed.
    2. def readdir(self, path, fh): This function returns what files and folders are in the file system. It is invoked when a "ls" command is executed.

- Our Helper methods:
    1. def __mount__(self): This function is called when first time the remote server is mounted locally. The Cache is updated and the mounting directory is created.

    2. def_convert_to_stat(self, response): This function is called on getting a response from the Stork Cloud. The local file data structure is populated and returned after parsing the JSON received from the response for relevant information about the files and directories.

2. **REST Client:** REST Client directly communicates with remote servers bypassing the DLS server. We provide only GET and PUT services. The client contacts the stork cloud with the including email Id and password as URL parameters: (email="<email_id>", password="<password>"). Our final software will support to connection to any remote server through Stork Cloud APIs, however currently in our prototype we support connection only to the ftp://mozilla.org server.

3. **POSIX commands supported:** "ls" and "cd" : Our software first looks up into the local cache, If the current directories sub-directories were previously fetched, then the directories and files are listed down in no time. Else, a GET command is issued to the server and after processing the response, the directories and files are listed down.

4. **Caching and Prefetching:** Caching: Local copy of directory/file metadata is stored as a <key, value> pair where key is the name of the directory/file and value is the JSON response from the server.

- Pre-fetching: To decrease the latency and increase the performance, system implements pre-fetching. The number of

levels of directories to pre-fetch is configurable.

`

1. Cache Freshness: To ensure cached data is not stale, our software periodically queries and fetches the updated directory metadata from the remote server.

5. **Limitation and Future work:** This current prototype of the software supports only Mozilla, we will support other remote servers by considering user entered arguments in the future.

   - Our software currently supports read-only file-system operations. In future we will also support write-operations.

6. **How to Test:**
   - Download and install the latest version of FUSE and linux environment
   - Create a mount folder in the current folder as fuse_mount/dls
   - Execute python FuesFS.py

7. **Time Analysis:** Below is the table depicting the time spent when an initial request is sent to the stork cloud and later requests on the prefetched cache. Here Cache miss represents the time when a certain file/directory is not yet mounted and the cache hit represents the availability of the metadata in the cache as key value pair.

Getattr	**Server Access Time**	**Cache Miss**	**Cache Hit**
Initial Request	6 sec	17 sec	52 ms
After Mounting	----	10 sec	Prefetched

VII. Related Work:

With the recent advancement in the computing paradigm, the heavy lifting for the majority of the computational jobs are being shifted to the network-oriented ecosystem. The prime characteristics of such systems are high-performance, network-aware file systems which are able to provide the storage and the data sharing requisites of the workgroups and the clusters in the given network. Cloud has been emerging as the prime choice to do the very expensive operation in terms of resources and time of the data transfer. Degoo, Box, DropBox, MegaUpload etc. are the well-known cloud storage services. However, they lack in the performance for heavy data transfer, which inhibits them from being widely in the network-oriented computing ecosystem. So, in order to overcome such issues we aim at constructing an optimized data transfer, cloud based computation mechanism among the nodes in the given ecosystem.

Globus Online: Is one of the technology which is by far the most similar work to ours. So, we concentrate on reviewing their work. It's being touted one of the services that provides a very simplified process of secure data transfer for the research community and at the same time being very fast and robust. It has been primarily used to manage the file activities on the environment comprising of supercomputing facilities, lab servers /computers and the clusters. Globus Online leverages the Amazon web services infrastructure-as-service for providing the required reliability and scalability. It uses GripFTP for providing the file transfer services. Software as a service is also one the features that's being supported whose advantage is that it provides the users and experts the ability to troubleshoot the bugs on their behalf. REST API is being used for supporting the file related activities to the users. However, one of the major

shortcomings of the Globus Online would be its compatibility with limited number of the protocols and the sub-optimal management of the firewalls which eventually affects the traffic over the network.

VIII. Conclusion:

A console application was successfully implemented with a one level pre-fetching. It also had the ability to copy the file from the remote server to its temporary directory. The application is also suitable for multi-user environments. The application is highly scalable and the directory listing capability is only limited by the memory and the disk space available.

IX. References:

[1] FUSE website http://fuse.sourceforge.net/

[2]Wikipedia page: File system in User space

[3] Linux Magazines: User space File Systems

[4] Karlsson, Magnus, Fredrik Dahlgren, and Per Stenstrom. "A prefetching technique for irregular accesses to linked data structures." HighPerformance Computer Architecture, 2000. HPCA6. Proceedings. Sixth International Symposium on. IEEE, 2000.

S[5] Patterson, Russel H. Informed prefetching and caching. No. CMU-CS-97-204. Carnegie-mellon univ pittsburgh pa school of computer Science, 1997.

[6] Griffioen, James, and Randy Appleton. "Reducing file system latency using a predictive approach." Proc. of USENIX Summer Technical Conf. 1994.