Paper title: The Sun Network Filesystem: Design, Implementation and Experience
Paper authors: Russel Sandberg
Your name: Parth Kansara (115135130)

What problem does the paper address? How does it relate to and improve upon previous work in its domain?
Several existing remote filesystem implementations were built under UNIX, and were unable to serve other operating systems and machine architectures. Thus, it was difficult to provision sharing of filesystem resources in a network of non-homogenous machines. Further, it was difficult to achieve remote file access at speeds similar to that of local accesses.

Previous works including **Locus**, **NewCastle Connection**, **RFS**, **IBIS** & **EFS** were either unable to handle diverse operating systems or were not yet released. **RFS** was similar to NFS in functionality but it was limited to UNIX filesystems. RFS uses a special purpose transport protocol that cannot be replaced. It also uses UNIX system calls instead of RPC mechanism, which introduces the issues related to interruption of system calls and modifications to support partial lookups. While the RFS protocol can support different machine architectures, it only supports System V.3 based UNIX operating systems. And since RFS uses the UNIX semantics, it does not provide the much required flexibility. Another drawback for RFS is its utilization of the stateful protocol, which requires the servers to maintain information about its clients. Consequently, it becomes very difficult to rebuild server state in case of a crash, making server recovery expensive. This becomes a pressing issue in cases of expanding networked filesystems, where the probability of a crash increases with every new node added. In terms of administration, RFS uses a centralized name service, which again limits flexibility.

What are the key contributions of the paper?
The paper addresses the issues of the existing remote filesystems and strives to build a system that is efficient in a network of diverse machines. To meet its goals, the NFS design comprises 3 main components - the protocol, the server end and the client end. The **NFS protocol** uses the SUN RPC mechanism, which simplifies the operations as RPC calls are synchronous. Further it is a stateless protocol, which facilitates crash recovery. In case of client crash, no recovery is needed whereas in case of server crash, client resends the NFS request until some response is received making the event seem almost equivalent to a much less concerning server slowdown. The RPC mechanism also allows the NFS protocol to be independent of the transport protocol used. There is also a separate MOUNT protocol used for fetching the *fhandle* of the root of the filesystem. This makes it easier for new filesystem access checking methods to be inserted. The NFS protocol uses External Data Representation (XDR) specification, which makes the protocol machine and language independent, along with easier definition.

The **Server Side** in an NFS server is stateless and it is required to commit any changes to the storage before returning the result. It also requires an inode generation number and filesystem id to be appended, which are a part of the generated *fhandle.* The inode generation number is incremented consistently and is used to address unique files.

The **Client Side** strives to provide transparency to applications via an interface. For this, it aims to resolve differences in remote and local path names by using the mount command to attach a remote filesystem to a directory and which reduces file address binding to only once per file system. It also provides the server with the access checking ability. The new filesystem interface in the kernel has two parts - Virtual File System interface and Virtual Node interface. For every mounted filesystem, there is one VFS structure and for every active node, there is one vnode structure. Since every vnode contains a pointer to its parent VFS and mounted-on VFS, any node can be used as a mount-point for another file system. For traversing pathnames, the path is split into components and a lookup call is made through the vnode for each component. This can further be improved by using a cache of directory vnodes. NFS also uses a memory allocator which allows expansion of the number of active vnodes and VFS structures.

Briefly describe how the paper's experimental methodology supports the paper's conclusions.
NFS performs certain design choices to navigate implementation issues. These include filesystem naming which provides a set of basic mounted filesystems on each machine on top of which other filesystems may be mounted, user authentication and security using a UNIX-type permission checking along with a new Yellow Pages database lookup service, RPC-based file locking, removal of open files by renaming to uphold UNIX semantics and immunity to changes in access permission to open files by saving client credentials to the file table. NFS was able to achieve comparable performance for remote and local files by implementing read-ahead and write-behind buffer caches on both client and server, caches for file attributes and directory names on client side, increasing max UDP packet size to 9000 bytes and reducing the number of times data is copied.

Write down one question you may want to ask the authors.
What are some possible ways to integrate parallelism in the NFS system?