Global-scale peer-to-peer file services with DFS

Antony Chazapis, Georgios Tsoukalas, Georgios Verigakis, Kornilios Kourtis,
Aristidis Sotiropoulos and Nectarios Koziris

National Technical University of Athens
School of Electrical and Computer Engineering
Computing Systems Laboratory
{chazapis, gtsouk, verigak, kkourt, sotirop, nkoziris}@cslab.ece.ntua.gr

***Abstract*— The global inter-networking infrastructure that has become essential for contemporary day-to-day computing and communication tasks, has also enabled the deployment of several large-scale data sharing overlays. Communities collaboratively aggregate and distribute file and storage resources either in the controlled environment of the Grid, or hidden under the anonymity cloak created by peer-to-peer protocols. Both designs exhibit unique properties and characteristics: Peer-to-peer algorithms address the formation of vast, heterogeneous and dynamic sharing networks, while Grids focus on policy enforcement and accounting features. A distributed data management facility that will assimilate respective practices has been envisioned by numerous related research initiatives, especially when there is a need to incorporate disperse resources in large pools, without relinquishing participants of their respective rights. In this paper, we describe the Distributed File Services (DFS) architecture — a peer-to-peer service overlay, which allows distinct administrative entities to form arbitrary file distribution relationships. Each DFS peer can be uniquely authenticated and maintains direct control of its own namespace and storage assets by defining corresponding authorization directives and policies. The peer-to-peer nature of the system allows for scalable deployment and resource allocation, either in a stand-alone scenario or in the Grid context. Moreover, we introduce the notion of a "web of files", as a non-hierarchical, global-scale namespace of distributed data collections and elaborate on a prototype implementation that features novel semantics for integrating our architectural principles and concepts into the operating system level.**

I. Introduction

In the past decades, research and practice in the world of computing systems has greatly transformed, moving deeper and deeper into the new academic and industrial arena that was born of the Internet phenomenon. One of the greatest challenges of this new era is to eventuate in an agreement of the protocols and methods that will allow global-scale sharing and pooling of resources over the common network infrastructure. For instance, consider distributed data management and in particular file handling. Files, representing one of the most intuitive and powerful of the traditional computing concepts, live on in a planetary-wide scale. Once stored inside a single administrative domain, they now have to be accessed, managed, shared, located, replicated and transferred across domains using distributed operations that must address issues like authentication, authorization, scalability and fault-tolerance. Operating systems may have been designed for operation in a networked environment since the early days, but their design does not anticipate any of the modern challenges of an active presence in the Internet. In the global repository of resources, data is volatile and communication is insecure and unpredictable. Cooperative data sharing environments impose unique features to distributed files that must be handled by emerging filesystem designs.

The recognition of this issue is not new, although relevant proposed architectures that tackle the problem are still largely classified into the research domain. The same vision is also reflected by the joint effort of researchers around the world that have designed and deployed the Grid — a service-oriented facility that allows the integration of remotely located, disjoint and diverse processing and storage resources [1], [2]. The Grid may soon provide the generic mechanisms required for ubiquitous computational and storage consolidation, analogous to the paradigm the World Wide Web has become for content viewing. Furthermore, a major attempt in the same direction is represented by the work done in peer-to-peer overlay networks, although using a completely different architecture and approach regarding sharing policies. In the Grid nomenclature, file services are provided by the Data Grid layer, a set of basic services interacting with one another in order to allow the distributed storage and management of a large set of shared data resources and to expose well known APIs to end user applications and other higher-level Grid software layers [3], [4]. The Grid design requires that such services are highly-available and statically located, while users of resources must be authenticated and authorized in order to enforce relevant policies and accounting. On the other hand, peer-to-peer file sharing deployments focus on scalable and fault-tolerant file transfer services in an environment where all participants share the same capabilities and responsibilities, while there is a strong requirement for anonymity. In a peer-to-peer file sharing system there can be no authority and no well-known, static resource. While these two approaches seem contradictory, there is a growing trend of merging corresponding algorithms into new designs that retain the best characteristics of both worlds. Peer-to-peer techniques pose unique properties that have been recognized by many to be necessary in order to shape next-generation Grid deployments [5], [6], [7].

Our proposed Distributed File Services architecture (DFS) enables distributed access and management of files and storage resources in a peer-to-peer environment while retaining full local control and allowing accountability of corresponding actions. In essence, we introduce a peer-to-peer service-oriented architecture for distributed file operations that allows us to synthesize the diversity, scale and dynamicity of a typical peer-to-peer network with the control and accounting features required by the Grid. DFS allows users of the system to

administer file metadata and actual data as two distinct entities, independently of each other, although combined into a single overlay. Moreover, we present a novel distributed filesystem design where there is no global hierarchical namespace, but a graph of names that can have unlimited traditional "root" points and views — what we call a "web of files". One of the greatest challenges has been to integrate all concepts of the DFS design in an implementation that would integrate seamlessly with already deployed Grid services and present a straight-forward interface to the user.

In the next section we discuss the specific requirements and scenarios that have formed this project, before proceeding to describe the principles and architecture of our design. In subsequent sections we present details of the prototype implementation and its specific features, as well as representative related work in the field.

II. Design

A. Requirements and challenges

The design and implementation of our distributed file services framework has in large part been driven by the requirements set by the European Union's Grid4All research project [8]. Grid4All's main objective is to produce the necessary infrastructure that will enable domestic users, non-profit organisations such as schools, and small enterprises to participate in a massive resource sharing network over the Internet. The vision of the project is a "democratic" Grid where inexpensive resources are cooperatively pooled in a dynamic and scalable fashion. Even small-scale users that do not have the necessary computing and storage assets to participate in current Grid deployments should be able to contribute their resources and in turn be able to utilize the unified substrate. All services implemented in such an environment face great challenges regarding security, support for multiple administrative and management authorities, on-demand resource allocation, heterogeneity and fault-tolerance.

Most relevant issues have already been addressed in the context of peer-to-peer data sharing and resource location networks — most, except administration and authority, which contradict the purpose the majority of such systems have been designed for. In classic peer-to-peer overlay designs, participants agree on the protocol to be used, which in turn enforces the policy of resource allocation and usage. Peers donate their storage resources to a common pool which is then managed by the network itself. Of course, one can refuse to implement the complete protocol semantics or process selective commands in an arbitrary way, although the corresponding results of such behavior are ambiguous. On the other hand, the design proposed in this paper takes into account a peer-to-peer environment where:

- Peers have direct control of their resources. Each peer may administer its own storage and file objects and perform operations on them independently of their location and usage in the network.
- Peers have control of how their resources are used. Each peer may authorize specific peers to certain actions. Also each peer may define its own sharing policy.
- Peers should be able to allocate and use resources they do not physically possess. This can be achieved either by pooling of resources or sharing, as long as the process complies with the previous requirements.
- All actions should be accountable. Every transaction in the network should be traceable to a named peer, resource or combination of two.
- The network's capacity should grow as more nodes join it, in typical peer-to-peer fashion. Moreover, well connected and well resourced nodes should be exploited when needed and if they allow so.

Moreover, the Grid environment we target has imposed special requirements, including:

- Shared namespaces: In addition to sharing file contents, participants should be able to agree on common collections or clusters of files. This is traditionally achieved through distributed filesystem designs where numerous peers agree on a common namespace of data. We should allow equal functionality, additionally supporting the ad-hoc creation and management of multiple such views.
- Support for multiple storage types: As we presume co-operation among new and already deployed file services, we should provide mechanisms for merging existing data exported via GridFTP, FTP, HTTP, etc. into the same distributed namespace and allow seamless access to objects disregarding the transfer protocol or location.
- Support for special file types: Data contained in files may have special semantics, and as so require or support special operations beyond access, move, copy, delete, etc. For example log files may provide special mechanisms to append entries or files storing experimental results from scientific measurements may contain special metadata.

B. Architecture

Before delving deeper into the details of the proposed architecture, we discuss one of its most prominent characteristics: The separation of storage resources into two distinct entities:

- The *filesystem resource* refers to a namespace and file metadata, as in typical computer filesystems. It should be noted that only the file description and information content belongs to the resource — storage space for data blocks belongs to the storage resource, defined next.
- The *storage resource* refers to the actual storage of content. Filesystem resources are used to associate data with filesystem peers and policies, whereas storage resources are consumed to store file data and make it available for retrieval over the network. Storage is allocated in *data block* units.

Thus, we define two distinct but overlapping peer-to-peer networks: One for namespace aggregation and unification and one for collaboratively consolidating storage assets. These overlays collectively form the *DFS Network* — a system that complies with the principles and design of the *DFS Architecture*. Metadata and data separation allows the same

data to be available under many names and moved in the namespace without actually being transferred. Moreover, it offers unlimited flexibility in managing and defining namespaces, file types and special file attributes. Current Grid middleware implementations follow a similar approach, although using static, centralized services for metadata indexing [9]. Such services are responsible for maintaining files' "logical names", which may in turn correspond to replicas located in numerous physical sites (actual data) [10]. Our approach offers augmented flexibility when managing namespaces and storage, though it can be used in conjunction with current deployments.

The DFS Architecture is based on the following principles:

- *DFS Users* are uniquely identified by a cryptographic certificate. With this certificate, they can be named, authenticated, authorized, accounted, attributed an action, associated with a resource entity or with each other. The term user throughout this text refers to a DFS User.
- Any resource and policy concerning resources is identified by and submitted to the unconditional *authority* of a unique DFS User.
- Any *action* upon a resource can be ultimately attributed to a single DFS User. This facilitates accountability of actions and simplifies resource access control.
- Any resource manipulating action is equivalent to a *transaction* between the DFS User performing the action and the authority of the resource. The former must acquire the permission of the authority in order to successfully complete the action.

In contrast to current peer-to-peer designs, where peers correspond to physical users or machines, we define a peer in the DFS Architecture as an autonomous, cryptographically certified authority, that has complete control over a specific set of namespace and/or storage resources. We then provide the mechanisms for resource management and sharing between authorities, always in the context of conformity to relevant policies and authentication schemes that are enforced by the authority providing the resource. In practice, assuming each DFS User corresponds to a single public/private key pair, resource control may be achieved through directives in an access list. A physical user may choose to create and administer multiple certificates — one DFS User for each group of resources exposed using different sharing attributes. Respectively, the physical user accessing a resource may pose as any DFS User under his control in order to request a transaction. The result will depend on the permissions and policies defined by the resource's authority in respect to the identifier of the consumer DFS User.

According to the principles, a DFS User (a cryptographic key) may *own*, *provide* and *consume* resources in any combination. We use the term *provide* to emphasize the delegation of management from one user to another, without sacrificing ownership. Users owning resources have direct control over respective access rights and may choose which namespace and storage assets can be accessed. Resources may also be allocated and then managed by another user. The allocation action corresponds to a generic mechanism for pooling raw storage from multiple providers or using some other peer's namespace in order to remotely register file names and metadata. In the DFS network some users may not posses resources at all and some may relinquish their rights and only choose to offer resources to the infrastructure.

C. Resource naming and structure

Both namespace and storage resources are identified in the DFS Architecture using the URI model: `dfs://user:service/path`. The `dfs://` prefix stands for the entire protocol family within the DFS network, while the `service` directive denotes that the URI points to either filesystem (namespace) or raw storage resources and encodes the specific protocol that should be used to access the corresponding `path`. The latter may equally represent a file in the respective user's namespace or a storage handle that uniquely identifies a data block of arbitrary size. By utilizing URIs, DFS software components can transparently incorporate established protocols, as well as name and manage external resources along internal, architecture-specific objects.

Namespace entities identified by URIs may refer to regular files, directories or any special file type. Each such entry is managed by the associated user's filesystem service and stored along corresponding metadata. In the case of regular files, data appears to the filesystem resource as URIs *pointing* to the block contents of the file. This is dictated by the principle of explicit filesystem and storage resource separation. Other metadata encode the policy according to which both the object and file data can be manipulated. Note, however, that permission from the filesystem is not in principle capable to control a user's access to the actual data, as storage belongs to a different resource. Users define separate access rights when owning namespaces and providing storage. Also, while filesystem rights refer to the usual set of operations like read, write, copy, move, etc., storage resource privileges can be defined on a different set of actions, like allocation, which may take into account size quotas and storage types.

Another prominent feature of the DFS Architecture is that filesystems can link to each other, much like web pages do. While each user will normally maintain an hierarchical namespace, resembling a traditional filesystem structure, the overall pattern of the file namespace throughout the DFS network may be arbitrary — a "web of files". Namespace manipulation is achieved through a special type of file, the *metalink*. Metalinks may point to any DFS filesystem URI. In this way, users may form simple or more complex filesystems in a peer-to-peer fashion, symbolically including foreign hierarchies into their own. Resources may be combined in shared namespaces, or a single file may appear under numerous paths. Note that in the latter case, only one of those paths remains the *authoritative path* for the resource. This is by definition the path whose user also owns the file. When using metalinks, it is evident that each user can define access rights only to the namespace under his control. The user controlling the authoritative path may define the privileges in respect to the file's metadata,

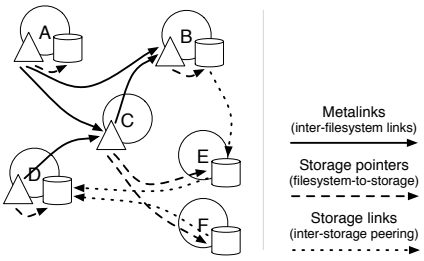

Fig. 1. Simple example of DFS User interconnections

while users linking to the file may only define access rights to corresponding links.

Metalinks in the DFS network can be considered as an extension of the traditional filesystems' symbolic links. Nevertheless, the resulting effect is unique. Traditional *distributed file systems* focus on the management of a single, global namespace over a distributed group of participants. This also implies and imposes a single administrative domain — that of the shared namespace, that can be managed either by a single entity, or collectively by all group members using complex consistency mechanisms. In the DFS Architecture, the ad-hoc, peer-to-peer nature of participants is also extended into the filesystem structure. Each DFS User manages and controls namespace hierarchies in owned or delegated administrative domains, while inter-filesystem connections allow for a global structure. This represents a novel conceptual approach to distributed filesystem designs. Intuitively, it is a *peer-to-peer filesystem* and not a distributed filesystem formed by peers.

Storage services are structured in a similar way. While a DFS User has complete control over the storage owned, there can be users that do not contribute, but rather pool storage from selective providers. This is achieved through links in the storage services layer. Each user maintains a storage service that may manage owned storage, linked storage or a combination of both. Filesystem metalinks and links at the storage level may encode peering agreements on resource provision and allocation.

An example scenario is depicted in Figure 1. Circles A through F represent DFS Users that manage namespace resources (triangles), storage (cylinders) or a combination of both. Note that the filesystem structure of user A is extended to include part of B's files, either with the metalink from A to B, or via C. File data for each user may be stored locally or remotely. User C does not own storage and is the only one that does not run a storage service. However, what appears to be local storage space for other users may be an aggregation of both local resources and remote offerings. User B provides part of his storage to E, which in turn offers part of it to D. D also uses F's local storage.

D. Resource discovery

The use of DFS-specific URIs for peer and resource identification necessitates employment of a Global DFS User Index for aquiring network access to resources. Physical peers participating in the DFS system cooperatively store user-to-service mappings via a Distributed Hash Table or similar structure. Thus, the actual network location of each user's service can be found by querying the index for the respective public key. Similarly, DFS URIs translate to network paths and resources can be accessed by contacting the right network endpoint using the URI-encoded protocol. The DFS-wide user index also allows peer mobility.

The aforementioned requirements dictate a mechanism for discovering globally available resources and exploiting the fact that some nodes may offer more resources than others. Both challenges can be addressed through the thoughtful deployment of special DFS Users — special administrative entities that only pool and offer resource pointers and may incorporate specific algorithms for load-balancing or load-skewing through provisioned resources. Similarly, special peers may be deployed to implement a variety of higher level services, such as indexing and search engines according to the web paradigm.

E. Grid integration

In the Grid context, the authenticity of each physical user is certified by a local certification authority. Moreover, multiple physical users may form arbitrary virtual groups, referred to as "Virtual Organizations". Resources are safeguarded by authorization directives and respective policies [11]. The DFS Architecture relies on similar principles, though there is no distinction between physical users and virtual groups. Considering the deployment of a DFS network in the context of the Grid, there is a need to map corresponding "Grid authoritative entities" into DFS Users. This is to be handled by a distinct architectural component — an *authentication plug-in*, that in the case of the Grid may be deployed side-by-side to organizational certification authorities. A variety of current Grid VO management utilities may need to become DFS-aware in order to transparently administer dynamic changes in relations between Grid users and groups and DFS Users.

Furthermore, if automatically created DFS Users own or provide resources, there is a need to initialize, deploy and publish corresponding physical network services over the network. This must also be handled by a new breed of VO management tools. In summary, when a group of users need to form a new VO, they must:

- Create a new DFS User (a new set of cryptogaphic keys).
- Record the relationship of the VO with the DFS User so members of the VO may also be authenticated properly in the DFS network.
- Run and register corresponding DFS services, if the VO is to own or provide resources.

Additionally, special VO-representative DFS Users may be deployed in order to pool their members' resources by hosting metalinks and storage links to respective filesystem and storage objects. Individual users may choose to delegate selective permissions to group members, as they retain full local administrative control over their resources. Metalinks from the VO User's filesystem may not be authoritative in regard to content, but may pose as "official" pointers to the group's set of collective assets.

III. Implementation

A. Overall structure

The implementation of the DFS Architecture is currently at a prototype status. Most of the design has been realized as software components that closely resemble distinct mechanisms and functionalities described in the architectural overview. The core of our implementation framework is the *DFS Peer* which provides the generic protocol handling functions in order to implement network communication endpoints. The DFS Peer manages and deploys software modules that implement services, depending on the role of the respective user in the DFS network. Such roles include:

- Filesystem resource ownership and provision, handled by the *MetaData DataBase (MDDB)* component. DFS Users use their MDDB to host and export namespace structures to the overlay. Corresponding file name and metadata entries - including ACLs, pointers to storage facilities, etc. - may be physically stored at the host filesystem or a local SQL database server.
- Storage resource ownership and provision, handled by the *Virtual Block Store (VBS)* component. A VBS peer manages and exports locally attached raw block storage on behalf of a DFS User. It also handles remote allocation requests and logically aggregates all the local and/or distributed storage resources under the respective user's disposal.
- Resource consumption, through a *DFS Client Library* that provides external software the ability to access the DFS network of resources.

Peer intra-communication is achieved via generic *DFS Messages* that comply to either the *MDDB* or *VBS protocol*. The DFS Message structure suggests a generic layout for protocols, based on request/reply verbs and corresponding arguments. MDDB and VBS protocols differ in the semantics of associated verbs and the handling mechanisms implemented in each component. The common message format permits peers to ignore protocols or protocol extensions they do not support or even to partially or abstractly parse messages.

DFS Users materialize in the network as their corresponding roles are handled by service components deployed in DFS Peers. Services, named by URIs which include user identification and protocol description directives, can be located via the global DHT index. Some users may only provide storage, thus appear as a VBS-enabled peer in the network, while others may export files, storage and access the network — all through a single DFS Peer hosting all necessary components. Note that a user not owning resources may only use the Client Library, although deploying an MDDB and VBS will result in imposing authority to corresponding namespace entries and flexibility in the management of remote storage offerings.

B. Implementation features

Specific features of the prototype DFS implementation, sketched out in Figure 2, are discussed in the following sections.

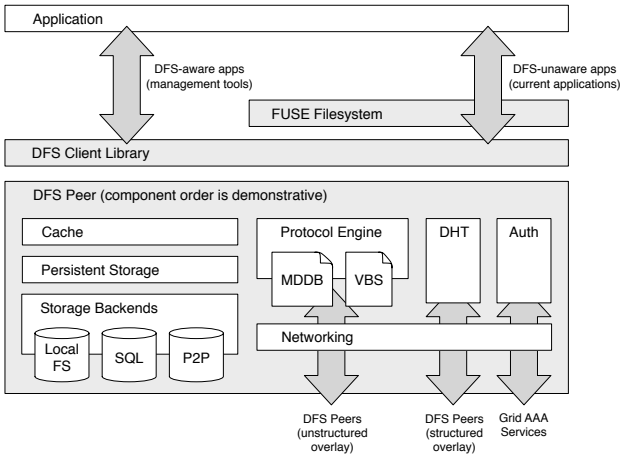

Fig. 2. Anatomy of the component-based DFS prototype

1) Operating system integration: The DFS Peer's Client Library exposes a high-level API that presents the peer-to-peer namespace and storage through traditional POSIX-like calls. Additionally, we have implemented a FUSE filesystem [12] that uses this API and allows "mounting" an MDDB hierarchy alongside local files in a Linux system. Normal applications can then transparently access the DFS Network via a file interface regardless of whether those files and their corresponding data is local or remote.

When viewing DFS through FUSE, metalinks appear as regular symbolic links, only with special naming semantics. The mounted DFS namespace, rooted at the local filesystem path specified at mounting time, includes a specific MDDB's hierarchy. Nevertheless, it also provides access to any other URI in the DFS Network through special paths. A virtual directory, named '`@`', is present in every FUSE directory, enabling the creation of metalinks via symlinks with the idiom `@/URI` as their target. Thus clients can navigate freely throughout the DFS network. Request for `@/URI`-like paths anywhere in the FUSE filesystem will dereference the URI and access it directly over the network. Likewise, local filesystem symbolic links can refer to any DFS file by prefixing the URI with the FUSE mountpoint and the `@` directory.

The `@` directory is actually a special case of a *virtual file*. Virtual files are special files that are not physically stored. They are used to access file metadata and unique operations of the DFS implementation through the traditional filesystem API (provided by the FUSE layer). Virtual files are formatted as: `<base path>@<virtual path>`. The `<base path>` signifies the actual (non-virtual) path that the virtual file is associated with. The `<virtual path>` can either be a name (if the virtual file is a plain file), or a whole hierarchy (in the case the virtual file is a directory). For example, the DFS User owning a file is accessible via `file@owner` and the visibility of virtual files when listing directories can be controlled by writing a boolean value in the `@visible` file in any directory.

2) Caching and disconnected operation: DFS Peers include a cache component to temporarily store frequently-

accessed remote objects (file metadata and contents). Users may also explicitly request that certain files remain cached through the client API or a virtual file (when using the FUSE filesystem). By exploiting the cache module we also have an initial implementation of a disconnected operations component. To support disconnected operations, DFS protocol mechanisms do not make any assumptions regarding remote peer availability. Locally cached and modified objects, as well as communication messages remain in pending state until their synchronization or delivery is acknowledged. Also, a persistent storage module allows such objects to survive peer downtime.

Offline operations can result in complex file consistency issues. In the prototype, we follow a variant of the *open-close consistency* scheme. When a remote file is opened, its contents can be modified in the cache and are only written back to the remote location and become globally visible when the file is closed (or flushed). We also allow users to define a policy of read-only offline files in order to completely avoid synchronization problems. Future work will address relevant mechanisms in more detail.

3) Publish-subscribe services: DFS Peers can subscribe to any object addressable by a URI and be notified asynchronously when events associated with the object occur. Event notifications can be reliably propagated and their delivery is deferred by the online status of a peer. There are two kinds of subscriptions to an object. One delivers a notice about an event that occurred, creating a local event, or *activity log* of the remote object. The other forwards the very actions received by the remote object, enabling a kind of selective state replication by locally executing the same actions on an associated object. File replicas, whose consistency is maintained through the action notification mechanism, are called *online files*.

The FUSE filesystem implementation provides a full virtual file-based API to facilitate DFS publish-subscribe services, as depicted in Figure 3 (virtual files and directories except `@` are shown in grey). The file `<base path>@subscribers` lists all current subscribers of the namespace object at `<base path>` and can also be used to subscribe for notifications, which in turn appear in corresponding virtual files of the `<base path>@notify/` directory. Writing to `<base path>@publish` produces message notifications at all registered subscribers and can be used by the owner of the object as a communication mechanism. Of course all these files are under the full administrative control of their respective authority or delegates.

4) Exporting local filesystems: To ease the deployment of DFS Peers, the prototype includes a mechanism to expose local Linux filesystems to the overlay in a single step. This is achieved through special MDDB and VBS components that exploit the local filesystem for metadata and data storage. The user specifies authority credentials to a script, which in turn starts all necessary services, including a FUSE filesystem that is mounted over the original Linux filesystem. The local directory must be hidden from the user in order to avoid consistency issues. Namespace operations, such as renaming, moving in the hierarchy and so on, pass through the MDDB

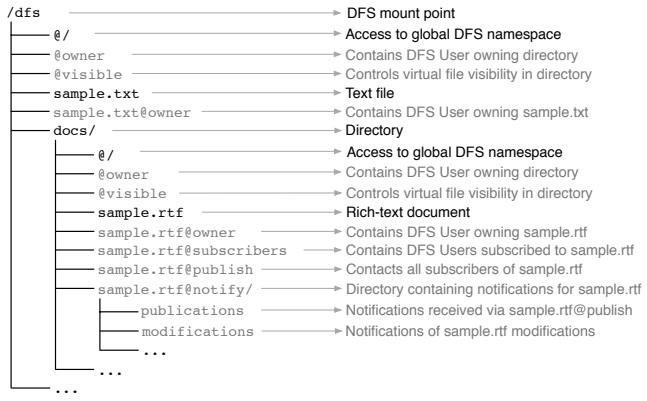

Fig. 3. Portion of the resulting filesystem structure when mounting a DFS namespace locally through FUSE

component before being applied to the original files. Metadata values that cannot be directly mapped to operating system semantics are stored in a hidden directory called `.mddb`. Similarly, the VBS instance exports the original files' data as storage containers named after their corresponding filesystem path. Again, storage allocation operations that cannot be mirrored directly in the filesystem structure, result in the creation of arbitrary files in a hidden `.vbs` directory under the mountpoint root.

5) Using a DHT for storing data blocks: We have also experimented with a P2P storage backend, which enables the deployment of special VBS peers that act as proxies to a Kademlia-based DHT. Each such VBS peer spawns a Kademlia instance into the specified overlay and translates operations from the VBS protocol, into accesses to content-hash blocks in the DHT. Peer-to-peer VBS peers allow a group of physical users to form configuration-free, self-managed, scalable and fault-tolerant collective storage pools, although by design they do not support all VBS operations (for instance, allocation requests have no actual effect).

C. Future directions

We are investigating various solutions for managing file and data replication and addressing respective consistency mechanisms and policies. In the DFS Architecture replicas can have the form of online files (replication through action notifications), result from offline, disconnected operations or be explicitly defined as distributed copies of a file in other authoritative domains. Furthermore, we target the implementation of an advanced replica-aware transfer component that will be able to take advantage of the availability of multiple copies when transferring the block contents of a file. Analogous algorithms and protocols have been deployed in both the Grid and peer-to-peer file-sharing contexts [13], [14], [15].

Another design issue that relies on similar implementation mechanisms is the DFS User's service availability. By replicating services at a number of DFS Peers we envision the notion of a *floating* DFS User that may selectively or collaboratively always be active in the DFS Network. The current, proof-of-concept prototype does not address DFS User availability issues and requires that "important" DFS Users are deployed

on highly-available DFS Peers. In the Grid environment, where most services are statically hosted at highly-available, fault-tolerant sites, floating DFS Users may not be necessary. Nevertheless, DFS User availability is critical for deploying DFS in dynamic networks.

IV. Related Work

Peer-to-peer networks are generally classified in two basic categories, depending on the structure of the overlay maintained by participating peers. *Unstructured* designs, like the pioneering Gnutella system, leave the peers free to form arbitrary message routing paths. To reach a resource, each peer will forward requests to his virtual neighbors, practically flooding the physical infrastructure with messages (at least this was the initial approach, in time replaced by more "clever" routing algorithms). Such deployments usually converge to a "small-world" network, thus allow reaching any resource with high probability and in a very small number of steps [16]. On the other hand, *structured* overlays or Distributed Hash Tables (DHTs), such as Kademlia [17], impose a specific virtual structure which accommodates peers in particular slots as they join in. Such overlays are mostly used for key-value pair storage, as the lookup procedure is highly deterministic and operations complete in a predefined number of steps — proportional to the logarithm of the number of participating nodes [18]. According to the DFS Architecture, DFS Peers are actually organized in both structures. A similar peer layout is described in [19]. The DFS collaboration network resembles an unstructured peer-to-peer design, as resource dependencies and links can be arbitrary between participating nodes. However, a subset of requests follow the overlay of namespace and storage aggregation and distribution. DFS User and resource location is handled by a Kademlia-based DHT maintained by all peers in the DFS universe. Relevant work can also be found in the context of peer-to-peer *content distribution networks* (CDNs) [20], as well as the Freenet censorship-free data distribution overlay [21].

The *DFS* acronym usually refers to *Distributed File Systems* — the software that enables a distributed group of interconnected computers to view and manage a single namespace of files and their collective storage. Distributed filesystems can be considered as the predecessors of most modern peer-to-peer and distributed file sharing services and as such have provided from early on the necessary tools and mentality required to envision contemporary global-scale data collaboration architectures. Most notable examples of initial distributed filesystem designs include Frangipani [22] and xFS [23]. Frangipani follows a two-layer approach, distinguishing between file metadata and actual storage (a shared virtual disk). The shared storage is the core of Frangipani — even metadata servers use it for communication. On the other hand, the authors of xFS describe a server-less network filesystem in which nodes cooperate with each other to provide all filesystem services. Our architecture of distributed interconnected metadata and file data servers closely resembles Lustre, a modern cluster filesystem [24]. In general, distributed and cluster-wide filesystems have similar objectives to peer-to-peer and distributed file sharing services, but mainly focus on different issues: High-performance installations inside a single administrative domain. An exception is represented by the Farsite project [25], which resembles xFS's peer-to-peer design, but does not require central administration and global trust. Farsite pools distributed resources into a single, logical filesystem.

Related work includes version 4.1 of the NFS network filesystem, which significantly enhances the traditional NFS design towards Internet-wide usage [26]. It supports an improved security model, where users are identified with strings of the form "user@domain" and separates data and metadata allowing them to be stored in different servers. In addition, multiple data servers can host the same data, enabling the client to perform parallel transfers. Client sessions, ACLs and on-demand mounting of remote filesystems further extend the usefulness of NFSv4, however its scalability and flexibility is far behind a truly distributed or peer-to-peer file system. Other relevant features of NFSv4 are aggressive client caching and delegation of file serving to clients. Furthermore, we should note CODA — a filesystem designed for user mobility [27]. Disconnected operation is provided as an extension of a caching scheme to handle indefinite write-back time intervals. CODA also performs file "hoarding", which is the automatic caching of specific files that have been listed as critical for a user. These files are monitored so that fresh copies are always locally cached.

An evolutionary step is represented by DHT-based storage architectures. For instance, CFS [28] uses a DHT to store file blocks and namespace structure, exploiting the peer-to-peer overlay much like Frangipani's shared virtual storage. DHTs have been extensively used as storage substrates in numerous related designs, as they provide a scalable and fault-tolerant mechanism to store data in a distributed fashion. Nevertheless, DHTs can only abstract a read-only data storage facility, which has forced researchers to propose hybrid designs: Pastis is similar to CFS, but uses the notion of modifiable i-node blocks [29], Eliot uses centralized metadata servers to index files stored in a DHT [30], while Oceanstore incorporates an upper-layer of servers that coordinate changes to file blocks [31]. Moreover, Ivy [32], like Oceanstore, stores a sequence of changes to the filesystem (log) in order to build the latest filesystem state from a set of read-only entries. Peer-to-peer filesystems significantly differ from the DFS Architecture, as they have no control on data placement. As mentioned earlier, respective policies are imposed by the DHT protocol and not by its peers. Also, all peer-to-peer filesystems, except Ivy, manage a single, distributed namespace. Ivy creates a namespace per user and addresses issues like shared namespaces (views), although the corresponding mechanisms are cumbersome as they depend on the read-only nature of the underlying DHT.

Similarly to DFS, in Plan 9 each user has a private view of the operating system, that is constructed by mounting arbitrary remote resources to the local hierarchy [33]. Actually, Plan 9 is

comprised of a whole ecosystem of languages, compilers and network protocols, with design goals similar to the Grid's. As such, Plan 9 nodes interoperate only with one other. In the Grid context, the construction of a single, uniform and hierarchical file namespace has been studied in [34] and [35].

V. Conclusion

Data management in the context of a global-scale network, like the Internet, remains one of the most challenging problems. Constant evolution of ideas and practices has led to numerous platforms that address different points-of-view of the same fundamental issue. Current deployments that create and expose a collaborative file space, either use a set of well-administered centralized components or distribute the work among anonymous participating entities. The Distributed File Services (DFS) architecture represents our effort to combine both methods into a single system, where distinct administrative domains can consolidate file namespace and data resources into a single overlay. DFS preserves each participant's full rights on his respective assets and allows for arbitrary peering agreements to be encoded into corresponding resource links. The net result is a peer-to-peer platform where peers' interests and policies form an unstructured "web-of-files" and ad-hoc pools of storage space. The DFS prototype implements most of the basic design concepts and provides us with a platform for future work and experimentation. Moreover, it offers features like a complete publish-subscribe framework and support for disconnected operations. DFS can be seamlessly used by existing applications, as distributed file semantics and special, implementation-specific features can be directly encoded into local operating-system file functions. A close study of related work reveals that the DFS design is by no means revolutionary, but a step in the direction of bridging and fusing respective practices into a design that meets a broader range of the overall requirements. Consequently, DFS offers a novel approach to file and data distribution and sharing — what we believe can prove beneficial to a multitude of prospective scenarios.

Acknowledgements

This research was partly supported by the EU-IST-034567 Grid4All project. We would also like to thank Marc Shapiro (INRIA), Vladimir Vlassov (KTH), Patrick Valduriez (INRIA) and Seif Haridi (SICS) for their valuable help during the preparation of this work.

References

[1] I. Foster and C. Kesselman, Eds., *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 2003.

[2] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[3] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," *Journal of Network and Computer Applications*, vol. 23, no. 3, pp. 187–200, 2000.

[4] "The DataGrid Project," http://eu-datagrid.web.cern.ch/.

[5] M. Cai, A. Chervenak, and M. Frank, "A Peer-to-Peer Replica Location Service Based on a Distributed Hash Table," in *Proc. of the 2004 ACM/IEEE Conf. on Supercomputing (SC2004)*.

[6] A. Chazapis, A. Zissimos, and N. Koziris, "A Peer-to-Peer Replica Management Service for High-Throughput Grids," in *Proc. of the 2005 Inter. Conf. on Parallel Processing (ICPP-05)*.

[7] P. Trunfio, *et al.*, "Peer-to-Peer Resource Discovery on Grids: Models and Systems," CoreGRID, Tech. Rep. TR-0028, March 2006.

[8] "The Grid4All Project," http://www.grid4all.eu/.

[9] E. Deelman, *et al.*, "Grid-Based Metadata Services," in *Proc. of the 16th Intern. Conf. on Scientific and Statistical Database Management (SSDBM04)*.

[10] A. Chervenak, *et al.*, "Giggle: A Framework for Constructing Scalable Replica Location Services," in *Proc. of the 2002 ACM/IEEE Conf. on Supercomputing (SC2002)*.

[11] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman, "A Multipolicy Authorization Framework for Grid Security," in *Proc. of the IEEE NCA06 Work. on Adaptive Grid Computing (NCA-AGC 2006)*.

[12] "FUSE: Filesystem in Userspace," http://fuse.sourceforge.net/.

[13] W. Allcock, *et al.*, "The Globus Striped GridFTP Framework and Server," in *Proc. of the 2005 ACM/IEEE Conf. on Supercomputing (SC2005)*.

[14] A. Zissimos, K. Doka, A. Chazapis, and N. Koziris, "GridTorrent: Optimizing Data Transfers in the Grid with Collaborative Sharing," in *Proc. of the 11th Panhellenic Conf. on Informatics (PCI2007)*.

[15] B. Cohen, "Incentives Build Robustness in BitTorrent," *Available online: http://bittorrent.com/*.

[16] D. Stutzbach, R. Rejaie, and S. Sen, "Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems," in *Proc. of the 2005 Internet Measurement Conf. (IMC 2005)*.

[17] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Proc, of the 1st Inter. Work. on Peer-to-Peer Systems (IPTPS'02)*.

[18] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking Up Data in P2P Systems," *Communications of the ACM*, vol. 46, no. 2, pp. 43–48, 2003.

[19] M. Castro, M. Costa, and A. Rowstron, "Should We Build Gnutella on a Structured Overlay?" in *Proc. of the 2nd Work. on Hot Topics in Networks (HotNets-II)*, 2003.

[20] S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies," *ACM Computing Surveys*, vol. 36, no. 4, pp. 335–371, 2004.

[21] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," in *Proc. of the 2000 Work. on Design Issues in Anonymity and Unobservability*.

[22] C. A. Thekkath, T. Mann, and E. K. Lee, "Frangipani: A Scalable Distributed File System," in *Proc. of the 16th ACM Symp. on Operating Systems Principles (SOSP-16)*, 1997.

[23] T. E. Anderson, *et al.*, "Serverless Network File Systems," in *Proc. of the 15th Symp. on Operating Systems Principles (SOSP-15)*, 1995.

[24] "Lustre: A Scalable, High-Performance File System," http://www.lustre.org/docs/whitepaper.pdf.

[25] A. Adya, *et al.*, "FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," in *Proc. of the 5th Symp. on Operating Systems Design and Implementation (OSDI'02)*.

[26] "NFSv4 Minor Version 1 Internet-Draft," http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-minorversion1-10.txt.

[27] M. Satyanarayanan, *et al.*, "Coda: A Highly Available File System for a Distributed Workstation Environment," *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 447–459, 1990.

[28] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area Cooperative Storage with CFS," in *Proc. of the 18th ACM Symp. on Operating Systems Principles (SOSP-18)*, 2001.

[29] J.-M. Busca, F. Picconi, and P. Sens, "Pastis: A Highly-Scalable Multi-User Peer-to-Peer File System," in *Proc. of the 11th Inter. Euro-Par Conf. (Euro-Par 2005)*.

[30] C. Stein, M. Tucker, and M. Seltzer, "Building a Reliable Mutable File System on Peer-to-Peer Storage," in *Proc. of the Inter. Work. on Reliable Peer-to-peer Distributed Systems (RPPDS)*, 2002.

[31] J. Kubiatowicz, *et al.*, "OceanStore: An Architecture for Global-Scale Persistent Storage," in *Proc. of the 9th Intern. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, 2000.

[32] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, "Ivy: A Read/Write Peer-to-Peer File System," in *Proc. of 5th Symp. on Operating Systems Design and Implementation (OSDI'02)*.

[33] A. Mirtchovski, R. Simmonds, and R. Minnich, "Plan 9 - An Integrated Approach to Grid Computing," in *Proc. of the 18th Inter. Parallel and Distributed Processing Symp. (IPDPS'04)*.

[34] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi, "Grid Datafarm Architecture for Petascale Data Intensive Computing," in *Proc. of the 2nd IEEE/ACM Inter. Symp. on Cluster Computing and the Grid (CCGrid2002)*.

[35] O. T. Anderson, *et al.*, "Global Namespace for Files," *IBM Systems Journal*, vol. 43, no. 4, pp. 702–722, 2004.