Orchestration of Distributed Storage Targets through Storage Flows

Josef Spillner, Alexander Schill
Faculty of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
Email: {josef.spillner,alexander.schill}@tu-dresden.de

***Abstract*—Distributed data storage is a topic of growing importance due to the mounting pressure to find the right balance between capacity, cost, privacy and other non-functional properties. Compared to central storage on physical media, on the network or in a cloud storage service, advanced data distribution techniques offer additional safety, security and performance. On the downside, these advantages come with a much higher complexity regarding the choice and configuration of where to store which parts of the data, and subsequent verification of where which data had been stored. Often, the storage targets must be configured individually while a centrally and locally accessible configuration interface with an appropriate propagation and verification mechanism would be more suitable. The complexity is further increased by additional data pre-processing tasks which are selectively applied to some of the targets. Compression, encryption and deduplication are typically present in pre-processing. With Storage Flows, we propose a new concept to manage distributed storage flows through systematic orchestration. The flows connect clients flexibly with intermediate data pre-processing tasks and finally the storage targets. We show that Storage Flows can be formalised and demonstrate their practical usefulness with implemented configuration and verification tools.**

I. Background on Distributed Storage Services

Data storage is a fundamental capability found in most computing systems. The question of how and where to store which data has become extraordinarily difficult to answer in recent years. Apart from the integrated primary mass storage area, for instance an HDD, SSD, SD card or even in-memory RAM disk formatted with an appropriate file system, secondary or external storage areas are used both for storage space extension and for data backup, snapshot, synchronisation and sharing purposes. Secondary storage can be divided into removable drives and media for local access and network drives for remote access. Online storage services are a specialisation of network drives which often combine the technical nature of storage areas with global access for subscribers and a service-specific business model. Block devices, file systems, databases and custom storage interfaces are increasingly offered over networks on demand. Cloud storage refers to online storage services which elastically scale and are billed according to the utilisation so that big data processing becomes available to more users. *The Cloud* is also widely seen as effortless and controllable data storage and sharing medium across all personal devices [1].

Fig. 1 puts all possible storage areas into context. From the perspective of a client as data producer and data consumer, all storage areas are application-specific storage targets. The choice of which target to use is often inconsistent between applications and only sometimes, only per client device and only for local areas, coordinated by the operating system (e.g. SSD as performance-increasing and energy-conserving buffer for HDD) [2], [3].

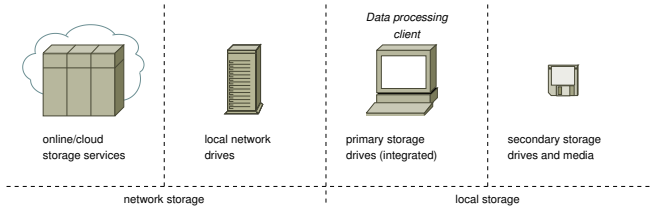

Figure 1. Diversity in storage areas and systems

Distributed data storage places any unit of data (e.g. files) on a deterministically chosen node (e.g. through hash functions over the file names or contents) or even on multiple storage areas in parallel. Depending on the distribution technique, which includes dispersion and replication, certain performance, security and quality advantages can be obtained [4]. In the case of purely local areas, data dispersion concepts as implemented in RAID perform this function to increase the safety by decreasing the effects of drive failures. However, for maximum storage safety and flexibility, local areas should be combined with remote areas in off-site systems. For such mixed areas, RAID controllers can be used when each area is exported as a block device (e.g. nbd, EBS). For most practical network and online storage services, block access is not possible. Therefore, higher-level storage controllers exist which work on the file level and consider service properties to optimise the storage behaviour, including cloud storage controllers for RAIC and RAOC [5].

When multiple storage areas are used, in particular both in parallel and serially one after another, storage flows from the client applications to their target areas result from such a configuration. Fig. 2 gives an abstract view on storage flow systems. The flows are tree-structured from a controller point of view, but graph-structured when taking the storage areas as endpoints into account.

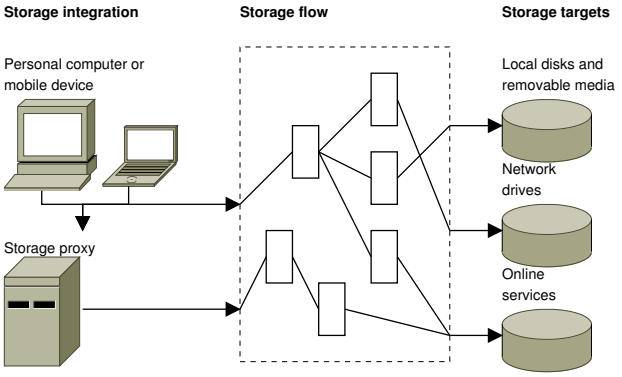

Figure 2. Simplified model of a storage flow system

While storage flows can be configured manually in current systems (e.g. the NubiSave cloud storage controller [5]), and expressed implicitly in dedicated data distribution policy languages [6], there is a need to formalise them for improved handling at configuration and operation time. Storage configurations with copy and split operator compositions have been proposed as formal model [7] but lack the mapping to actual storage flow systems with dynamic configuration, such as FlexArchive [8]. Furthermore, it should be possible to distribute the flows themselves across several nodes in order to achieve a high scalability and consistency between distributed configurations. In this paper, we first introduce storage flows as abstract trees and discuss their potential. Then, we present a management solution for distributed storage targets and software-defined storage flows to achieve their orchestration. This effort is an ongoing work, and yet we already present preliminary results from an experimental evaluation.

II. Storage Flows

In a storage process, the flow affects each unit of information captured as data in files f. The data traverses $o (o \in N)$ transformation operators ω ($f' := f \times \omega$) before finally arriving at a storage target as files $f^{(o)}$. The inverse direction (ω^{-1}) is the retrieval process. The operators cause state transitions over the nature of the data from files to files again ($1:1$), or to blocks or fragments ($1:n$), where each such file part is then re-interpreted as file at the next operator. Fig. 3 explains the states and the permissible state transitions. Hence, each stored file $f^{(o)}$ is the result of an operator chain $f \times \omega_1 \times \omega_2 \times ... \times \omega_o$.

The combination of all operator chains in a storage system forms a tree of operators and a graph of applications (data sources), operators and storage targets (data sinks). The maximum depth of the tree is o. The breadth at the leaf nodes for each individual storage process is n which corresponds to the notion of $n = k + m$ with k significant file parts and m redundant parts. However, the overall breadth may be larger than n.

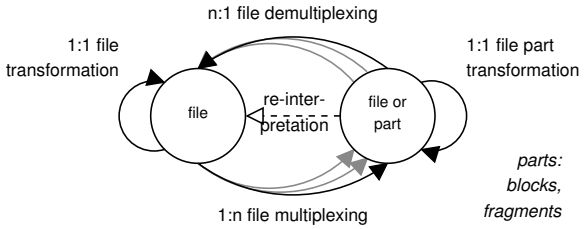

Figure 3. State model containing the nature of data as complete file or set of file parts, and permissible state transitions as file/part operations

The set of operators Ω contains $1:n$ file splitters ω_{split} (e.g. data chunking or dispersion), $\omega_{replica}$ and ω_{select} for selective forwarding (e.g. round-robin or load-dependent scheduling), as well as a number of $1:1$ operators: ω_{enc} for encryption (assuming the key is not part of the flow), ω_{steg} for steganographic concealment (with the same assumptions regarding the carrier message), ω_{comp} for compression, ω_{dedup} for de-duplication and ω_{ver} for versioning. Additionally, each target is integrated by means of an $1:1$ transport operator ω_τ.

Operators pass on the data to their successors in the chain but may cache their output $f^{(i)} (1 \leq i \leq o)$ in case the successor's performance is limited. In general, this applies to network-bound transport operators.

The open set of targets T contains τ_{dir} for local directories, τ_{obex} for media on Bluetooth devices accessible through the ObexFTP protocol, τ_{dav} for WebDAV services, τ_{s3} for S3 object storage services and an unrestricted number of further generic targets and provider-specific services.

Both operators and targets can be either locally located on the system which performs the storage integration, or placed on multiple computers to form a distributed storage flow system. This introduces a new operator ω_{fwd} which forwards data to operator chain tails organised as subtrees. A special variant of ω_{fwd} may appear to the application as (transient) storage target $\overrightarrow{\tau}$. The complexity of distributed storage flows, operators and caches leads to the requirement of keeping the global picture for flow administrators and ultimately to advanced distributed storage flow and service management designs.

III. Distributed Storage Service Management

In computer networks and especially *software-defined networking*, the concept of OpenFlow has recently gained a lot of popularity [9]. The main idea is to separate the flow control, which happens centrally in an OpenFlow controller, from the flow execution which carries parts of the configuration information as piggy-backed data to the switches. In fact, the notion of the thus influenced expression *software-defined storage* has recently been picked up by the community [10]. However, actual concepts or formalisms concerning software-defined storage or storage flows are

considerably missing. Similarly, distributed Cloud Computing environments have recently seen centralised planning and configuration methodologies, such as TOSCA [11], but do not explicitly cover distributed storage systems. The designs of OpenFlow and TOSCA on their abstract levels serve as mental models for the design of distributed storage service management tools. Both the initial configuration and propagation of storage flows and the continuous usage, visualisation and potential re-configuration at runtime are subject to a flexible and scalable design.

In storage flows, a distributed storage configuration is centrally created by an administrator and applied to a local storage controller which executes it. In addition, parts of the configuration may be propagated to other nodes which operate subordinate storage controllers to execute subtrees of the flow configuration. A specific interface is needed on these nodes to receive the configuration and report the success status of the application thereof to their controllers. This interface can either be a dedicated service interface (e.g. web service API), or use a piggy-backing mechanism as specially crafted files and metadata which are then separated from the payload data on the receiving nodes. Fig. 4 is an example of a distributed storage flow controller which spans two remote nodes. Each controller instance receives its configuration through a control interface (CTRL) and its data through another appropriate interface, e.g. a file system or a storage service API, as entry point to an operator.

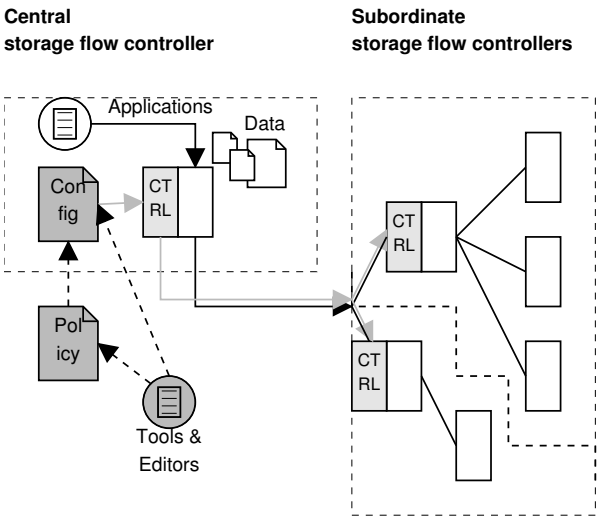

Figure 4. Flow control across three nodes in addition to data flows

The configuration can be created in a manual process using a configuration tool, or generated from a higher-level policy editor [6]. It encompasses not just the layout and parametrisation of the operators and their caches (if applicable), but also decisions regarding the flow itself. Among those, dynamic flow variations based on file contents, metadata and the environment context can lead to significantly different overlay flows. Fig. 5 demonstrates two flows at runtime based on policy-driven dynamic flow variations.

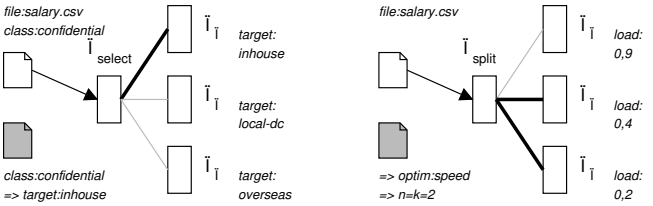

Figure 5. Two overlay flows created by (1) highly sensitive file contents and (2) server overload

IV. Storage Flows Configuration, Propagation, Usage and Visualisation Tools

Two file systems and two graphical tools are available to storage flow designers as open source software. This section describes the design and the implementation of the software along the flow lifecycle. It encompasses the graphical configuration, the automatic propagation, the usage as part of typical data processing tasks and finally the graphical visualisation.

A. Configuration

A storage flow configuration editor is a tool to generate, deploy and maintain storage flows. We have implemented a configuration editor as a Java desktop application called *StorageFlows Modeller*. It displays the chain or tree of operators, starting with a data source symbol and connecting to any number of data sink symbols.

For the flow deployment into a storage controller, operators are represented as local and network data processing modules with capabilities to store and retrieve data along with important metadata. Each such module is either a directory-level filter (deduplication, versioning), a file-to-file transformer (compression, encryption, steganography), a file-to-fragment splitter (dispersion) or a fragment-to-fragment transformer similar to the file-to-file ones. Furthermore, each module offers a configuration template from which a specific configuration instance can be derived through a module-specific dialogue in the editor. Weighted and directed connections can be set up between the modules to achieve a chain or, in the more general case, a tree. A screenshot of the application is shown in 6.

B. Propagation

The distribution or propagation of the storage flow is a secondary configuration step. It requires a list of flow processing nodes which contains the local node and/or a number of remote nodes. Any flow module can be assigned to one node. Subsequent modules are then automatically assigned to this node as well, which can be overridden. In order to convey the module configuration to the chosen

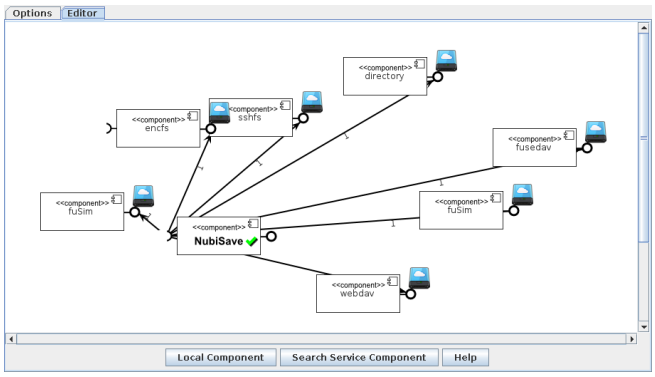

Figure 6. Screenshot of the graphical editor to create and modify storage flow descriptions

nodes, there needs to be a convention of how to receive the configuration. This work assumes that the technical module implementations themselves already be installed and just the configuration changes dynamically.

Data can be stored in files, in memory, in databases, through web services or using nearly any combination of these data management concepts, e.g. a relational in-memory database with a web service frontend and tables stored as files. The realisation of storage modules requires one such concept for each operator. Typically, applications work with files on a file system which makes this the concept of choice for our realisation.

We have implemented *FlowFS*, a user-space pass-through file system similar to BindFS or LocalFS. The deployment of FlowFS on any flow processing node therefore becomes an additional prerequisite to handle the propagation of flows. Initially, its backing directory is not connected and hence data write access will fail. However, it reads a special directory `.storageflows` which contains the configuration. Whenever the contents of this directory change, the backing directory (if present) is unmounted and all storage modules are mounted according to the configuration.

Fig. 7 demonstrates how FlowFS works in detail. There are two possible modes of the configuration handling. The first one mandates an in-band directory `.storageflows` can be distinguished from other files and directories by a filter. The advantage is that changes to the configuration are immediately known to the file system. However, this assumes that the configuration is always written unmodified, which prevents any operator module from working on top of this file system. The second design assumes an external configuration directory and a notification handler within the file system which monitors changes and dynamically switches the backing directory for storing files. Initially, this directory must be invalid (unconnected). It gets connected whenever a storage module description is written with the line `storageentrypoint=true` in it to denote the continuation of the storage flow at this directory. FlowFS implements both modes and requires a choice at file system instantiation time.

The propagation is triggered from the editor whenever the configuration changes on the local configuration node. Either the in-band directory receives the changed configuration, or the external directory on the assigned remote node is mounted on the configuration node and the configuration file is a symbolic link to a file which in fact resides on the remote node. Once that happens, the propagation cascades over all subtrees.

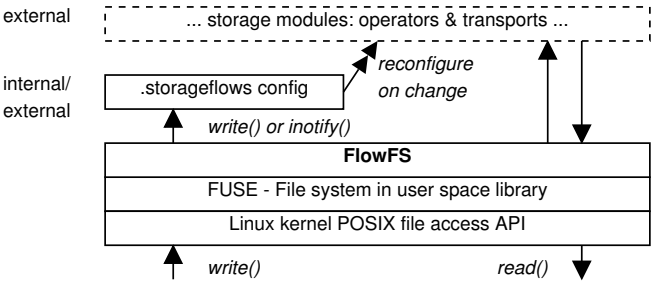

Figure 7. FlowFS, a reconfigurable pass-through file system for distributed storage flows

The file system FlowFS is implemented in C using three low-level system APIs: Linux Filesystem in Userspace (FUSE), the kernel-based Inotify file change handler and Pthreads for multi-threading to achieve a second main loop in parallel to FUSE for reacting on Inotify notifications. It is invoked as application by passing the mount point as parameter.

C. Usage

Using the stacked file system setup, the data simply flows at run-time from the source to the sink and back. For instance, a photo is loaded from a digital camera into an image-processing application. The user instructs the application to save the modified picture into the cloud. The file hits the appropriate folder which is a mounted instance of a splitter module. The file is split into three parts. One remains on the local disk, one is written to a directory which is a mounted remote storage target, and one hits another directory which forwards the fragment to a post-processing node.

For the splitter realisation, we have developed the NubiSave cloud storage controller [5]. It is implemented in Java using the FUSE-J API and JigDFS erasure coding algorithms. For all other modules, we rely on existing file systems and especially FUSE modules which can be mounted and unmounted at run-time without elevated operating system privileges.

D. Visualisation

For the visualisation of stored data, we have added a context menu to each module in the editor which invokes a

separate application called *NubiVis*. It consists of a graphical user interface and a statistics calculator which reads from the storage controller's file/fragment database and exports this information through a RESTful interface. Fig. 8 outlines the visualisation architecture.

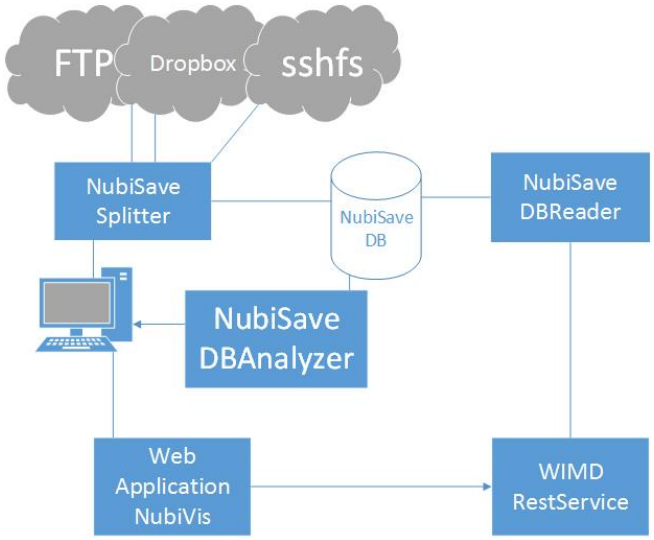

Figure 8. Visualisation architecture with a graphical frontend and a RESTful backend as parts of *NubiVis*

At run-time, users can verify how the flows spread their data across several storage targets. The statistics are generated per file or folder, per target, or globally per provider, and presented as tree view, bubble view or geographical map. In Fig. 9, the per-folder tree view is shown as an example.

Path	Size	Fragments
Dokumentationen	0 kb	
Dokumentation	0 kb	
Qt	0 kb	
latex	0 kb	
short-math-guide.pdf	5444270 kb	
joshi-working with pstricks pstree psnode.pdf	1165860 kb	
V16-mathe.pdf	2334970 kb	
useful-vector-grafic-tools-morales.pdf	3032370 kb	
latex-schnellreferenz.pdf	2008520 kb	
animate-1.pdf	36407030 kb	
pst-doc.pdf	1434970 kb	
listings.pdf	7159540 kb	
symbols-a4.pdf	43876860 kb	
pgfplots.pdf	65836040 kb	
latex-referenzen.zip	117715740 kb	

Figure 9. Visualisation screenshot showing the division of files into fragments which are dispersed among storage targets

NubiVis is implemented as a NodeJS server with a Dojo (JavaScript) web frontend. The server wraps a Java adapter to the NubiSave JDBM file/fragment database. The frontend further uses the Raphaël JavaScript framework to produce statistics diagrams using scalable vector graphics (SVG).

V. Experimental Evaluation

Through experiments with multiple nodes, we demonstrate the practical usefulness of storage flows. The evaluation criteria are (C1) overall feasibility, (C2) performance of the pass-through function of FlowFS, and (C3) performance of the propagation. General remote and distributed storage metrics are not covered but can be found in existing literature [5].

Initially, a user launches the storage flow modeller on a notebook to model a scenario 'tree'. It contains just one DavFS module to access a WebDAV interface of an internal network share (NAS). The user then assigns a remote execution property to this module so that it will be deployed on an intermediate server in order to be accessible from multiple devices with varying storage needs.

The scenario further foresees a capacity bottleneck at the NAS. This leads the user to reconfigure the flow to target a public S3 storage service instead. Due to privacy reasons, all data flowing to the service needs to be encrypted.

Fig. 10 highlights the chosen scenario to evaluate the propagation of the configuration from *StorageFlows Modeller* via *FlowFS*. Data and configuration are both first relayed via SSHFS to an SSH service on the second flow processing node. The data arrives at FlowFS which, according to the configuration, writes it to a WebDAV file system connected to a WebDAV service on the NAS (1). Then, FlowFS is reconfigured. The arriving data is now being encrypted and relayed via the S3QL file system to a server which makes a storage area accessible through the S3 protocol (2).

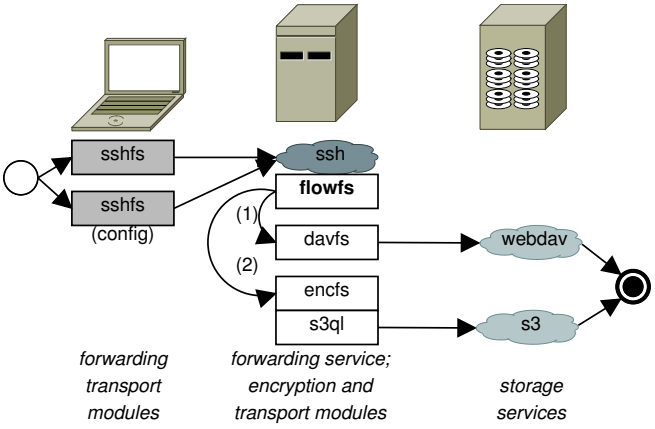

Figure 10. Simple distributed storage flow scenario which involves two nodes

The presented workflow can be realised with the tools and would be hard to do without. Hence, we claim that the approach is feasible and criterion C1 holds.

The measured pass-through performance of FlowFS, which is shown in Fig. 11 for continuous writes, is about 8-15% inferior compared to a direct write on the bare underlying Ext4 file system. This difference only applies to very large files though, and could likely be significantly reduced by implementation improvements and buffer tuning.

Furthermore, the network access is more often a bottleneck than the disk performance. We therefore see a potential for criterion C2 to hold for all practical use cases, but acknowledge the need to optimise FlowFS.

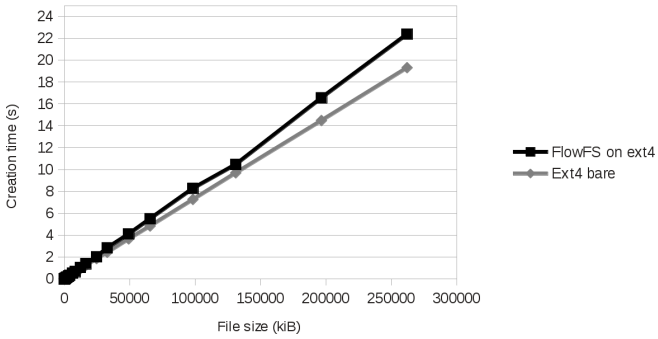

Figure 11. File system write performance with and without pass-through function of FlowFS

The time required for propagation (criterion C3) is not noticeable. In the given scenario, switching the backend of an SSHFS-mounted directory from a WebDAV backend to an encrypted S3 backend by submitting a new configuration file which contains `storageentrypoint` takes a hardly measurable time of $0.003s$ on a 1000baseT/Full Ethernet connection between the notebook and the forwarding node. In practice, users won't even notice newly added storages and redirected flows this way similar to the transparent re-routing of IP packets at the lower layers.

VI. Conclusion

This paper has introduced Storage Flows as a means to orchestrate the parallel use of multiple storage services, drives and media from a central configuration point. Through trees of transformation and transport operators, data can be flexibly dispersed from the applications to the available storage targets. The complex management of flows on one node and across several node has been simplified with centralised configuration tools (*StorageFlows Modeller*) and visualisation tools (*NubiVis*) as well as distributed propagation tools (*FlowFS*) which do not need to be manually reconfigured. The overall feasibility and adequate performance have been demonstrated with experiments.

We assume that there will be an increasing interest in software-defined storage in the near future due to the high rate of larger hard drives and more cost-effective cloud storage offers. Local, network and cloud storage need to be combined in a systematic and user-controllable way to yield the highest data handling quality.

Acknowledgements

This work has received funding under project number 080949277 by means of the European Regional Development Fund (ERDF), the European Social Fund (ESF) and the German Free State of Saxony.

References

[1] O. Riva, Q. Yin, D. Juric, E. Ucan, and T. Roscoe, "Policy Expressivity in the Anzere Personal Cloud," in *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC)*, October 2011, Cascais, Portugal.

[2] M. Canim, G. A. Mihaila, B. Bhattacharjee, K. A. Ross, and C. A. Lang, "SSD Bufferpool Extensions for Database Systems," in *Proc. VLDB Endowment – 36th International Conference on Very Large Data Bases (VLDB)*, vol. 3, no. 2, September 2010, pp. 1435–1446, Singapore.

[3] H. J. Lee, K. H. Lee, and S. H. Noh, "Augmenting RAID with an SSD for Energy Relief," in *Proceedings of the USENIX Workshop on Power Aware Computing and Systems (HotPower)*, 2008.

[4] M. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM*, vol. 36, pp. 335–348, 1989.

[5] J. Spillner, J. Müller, and A. Schill, "Creating Optimal Cloud Storage Systems," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1062–1072, June 2013, DOI: http://dx.doi.org/10.1016/j.future.2012.06.004.

[6] J. Spillner and A. Schill, "Flexible Data Distribution Policy Language and Gateway Architecture," in *1st Latin American Conference on Cloud Computing and Communications (LatinCloud)*, November 2012, pp. 1–6, Porto Alegre, Brazil.

[7] B. Mungamuru, H. Garcia-Molina, and C. Olston, "Configurations: A Model for Distributed Data Storage," in *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing (PODC)*, August 2007, pp. 332–333, Portland, Oregon, USA.

[8] S. Chaitanya, D. Vijayakumar, B. Urgaonkar, and A. Sivasubramaniam, "Middleware for a Re-configurable Distributed Archival Store based on Secret Sharing," in *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware (MIDDLEWARE)*, November/December 2010, pp. 107–127, bengaluru, India.

[9] N. Mckeown, S. Shenker, T. Anderson, L. Peterson, J. Turner, H. Balakrishnan, and J. Rexford, "OpenFlow: Enabling Innovation in Campus Networks," 2008.

[10] B. Earl, D. Black, J. Kistler, R. Novak, and U. Maheshwari, "Software-Defined Storage," Panel Discussion at the 5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage), June 2013, San Jose, California, USA.

[11] J. Wettinger, M. Behrendt, T. Binz, U. Breitenbücher, G. Breiter, F. Leymann, S. Moser, I. Schwertle, and T. Spatzier, "Integrating Configuration Management with Model-driven Cloud Management based on TOSCA," in *3rd International Conference on Cloud Computing and Services Science (CLOSER)*, May 2013, Aachen, Germany.