Future Generation Computer Systems 29 (2013) 1062–1072

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Creating optimal cloud storage systems

Josef Spillner *, Johannes Müller, Alexander Schill

Technische Universität Dresden, Faculty of Computer Science, 01062 Dresden, Germany

ARTICLE INFO

Article history:
Received 4 February 2012
Received in revised form
4 June 2012
Accepted 11 June 2012
Available online 17 June 2012

keyword:
Online storage
Cloud storage
Storage array
Storage ontology
Dispersion

ABSTRACT

Effortless data storage "in the cloud" is gaining popularity for personal, enterprise and institutional data backups and synchronisation as well as for highly scalable access from software applications running on attached compute servers. The data is usually access-protected, encrypted and replicated depending on the security and scalability needs. Despite the advances in technology, the practical usefulness and longevity of cloud storage is limited in today's systems, which severely impacts the acceptance and adoption rates. Therefore, we introduce a novel cloud storage management system which optimally combines storage resources from multiple providers so that redundancy, security and other non-functional properties can be adjusted adequately to the needs of the storage service consumer. The system covers the entire storage service lifecycle from the consumer perspective. Hence, a definition of optimality is first contributed which is bound to both the architecture and the lifecycle phases. Next, an ontology for cloud storage services is presented as a prerequisite for optimality. Furthermore, we present NubiSave, a user-friendly storage controller implementation with adaptable overhead which runs on and integrates into typical consumer environments as a central part of an overall storage system. Its optimality claims are validated in real-world scenarios with several commercial online and cloud storage providers.

1. Vision: towards resource ubiquity and optimality

Access to software and data anywhere, anytime, on any device and with any connectivity, has for a long time already been a crucial topic for researchers and developers in operating systems, user interfaces and service science. The amount of managed data is increasing each year, both in large-scale systems and in smaller and personal environments. Likewise, more computation is being performed to process the data, and more communication is performed to distribute the data. This phenomenon is coupled with a steady increase in computing, storage and communication resources available, although with different characteristics: Depending on the nature and purpose of data, these resources are required with varying intensity, ranging from not at all over a long time to as much as possible in short burst periods.

Several approaches exist to deliver software and data to their users anywhere and anytime. Cloud computing is a recent term which conveniently encompasses the notions of ubiquitous on-demand access, pay-as-you-go utility and seemingly indefinite elastic scalability. For the software-as-a-service layer, several platforms exist already as mature and commercialised offerings. Deficiencies still exist in the handling of resource-related cloud services on the infrastructure level, especially storage, computing and communication services depending directly on the available hardware resources.

This article, extending and generalising previous research on optimal cloud storage controllers [1], is contributing to a more holistic view of entire resource service systems, especially concerning the achievement of optimality for secure, highly performing, cheap, resource-conservative and effortless use thereof. Among the available resource services, it concentrates on distributed data storage. Recently, the number and popularity of online storage services for both personal and enterprise use have increased significantly. Advances in handling this resource service class are of high practical value. Several controllers for dispersing information over a redundant array of independent cloud storage providers (RAIC) have already been described in recent literature, but none of them is designed for arbitrary user-defined and system-supported optimality. Hence, the illustration of optimal cloud storage systems is complemented by an extensive validation through an optimal cloud storage controller which, together with all experimental data, we provide to the community.

The scope of cloud storage systems is shown in Fig. 1. It assumes the presence of a storage gateway, which at its core contains a technical storage controller as the interface for all applications in need of storage. The criteria for optimality according to the scope are (1) storage service combinations with optimal characteristics with respect to the user's requirements, (2) optimal processing in the storage controller which should contribute to, rather than

* Corresponding author.
E-mail addresses: josef.spillner@tu-dresden.de (J. Spillner), s9186231@mail.zih.tu-dresden.de (J. Müller), alexander.schill@tu-dresden.de (A. Schill).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.06.004

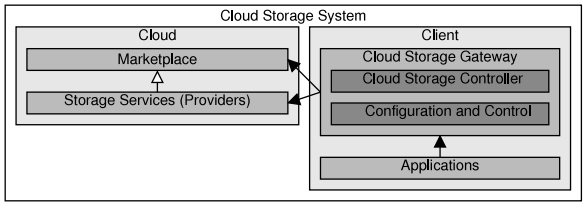

Fig. 1. Architectural scope of cloud storage systems.

constrain the desired service properties, and (3, not discussed in this article) an optimal configuration with preferably no user involvement beside the requirement specification.

The article is structured according to the lifecycle of services in open service markets. Generally, the offering, preparation, usage and feedback phases have been proposed in existing literature [2]. The service lifecycle specific to storage services is shown in Fig. 2. For each phase of relevance to achieving optimality, namely preparation and usage, the corresponding section describes both the view of the service provider and the service consumer. The lifecycle-bound sections are preceded by a cross-cutting section on modelling the functional and non-functional properties of storage services, resulting in formal service descriptions which are essential for achieving consistent optimality in the overall system. A section on the validation of the results through experiments with a cloud storage controller implementation and a discussion of related work follows, before the conclusion and outlook towards future optimality improvements.

2. Modelling storage services

Services are primarily defined by models whose characteristics influence all automated service management aspects. The characteristics encompass complexity, maintainability, suitability for existing service offerings and derived service description quality metrics such as precision and expressivity. Hence, we propose to use ontologies for the storage service modes which offer intrinsic computational concepts (rules and logics) as well as linkability with existing vocabularies. Our methodology to capture relevant cloud storage concepts for both enterprise and private use has been to perform a survey of existing online and cloud storage providers. In it, 67 providers have been analysed and the findings have been formalised by storing them in relational tables and producing the ontology concepts and instances from those.

The consolidated cloud storage service ontology is visualised in Fig. 3. It allows for differentiating providers according to technical, business and legal optimality criteria defined by the user. For instance, one user might require free-of-charge services without the need to submit personal data to the provider while another one insists on a high capacity. Furthermore, it conveys optimisation potential to cloud storage controllers by describing which services already offer characteristics which would otherwise have to be achieved on the client side, such as encryption and redundancy.

3. Preparing storage services

This section introduces the conceptual extension of arbitrary or secure cloud storage arrays towards optimal cloud storage arrays. By building on the storage service modelling output and service science concepts for bundling and aggregating services, it describes the selection, aggregation and configuration of distributed storage.

3.1. Selection and aggregation

Achieving cloud storage optimality requires the service consumers to use and aggregate several storage resources according to certain patterns and constraints. Hence, the storage service lifecycle (see Fig. 2) is entered once per consumer from the provider's perspective, and once per provider from the consumer's perspective.

By leveraging research on non-functional property specifications from the research communities around Component-Based System Engineering (CBSE) and the Internet of Services (IoS), the concept of RAIC can be extended to a weighted combination of arbitrary quality and context properties beyond inexpensiveness [3]. This makes it possible to consider both subjective metrics such as trust in a storage provider and objective metrics such as cost, offered bandwidth, access restrictions and geographical location. The weights express weak and strong priorities. Strong priorities can be used to mandate the inclusion or exclusion of providers with certain properties, whereas weak priorities are used to sort the remaining set of providers. This follows the mathematical concept of

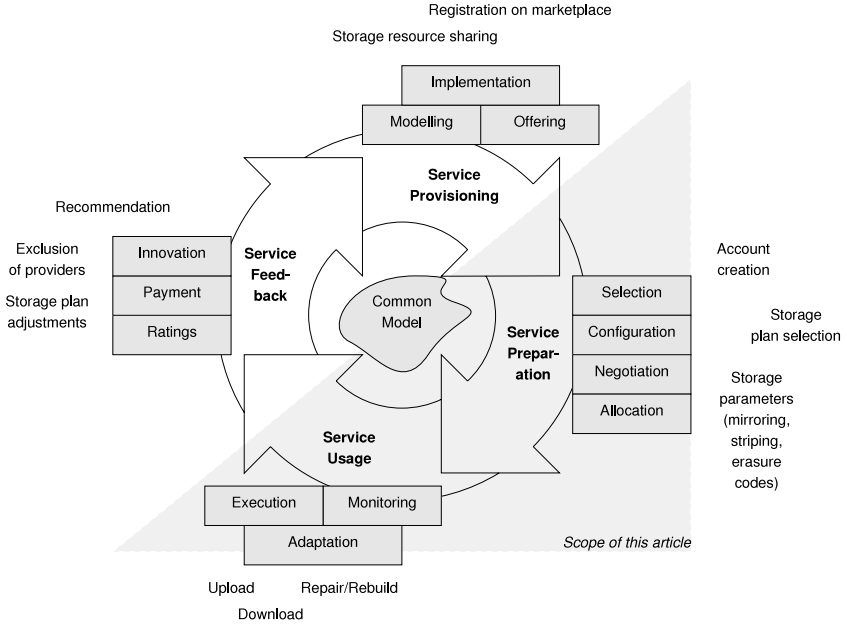

Fig. 2. Lifecycle of services in general and of cloud storage services in particular.

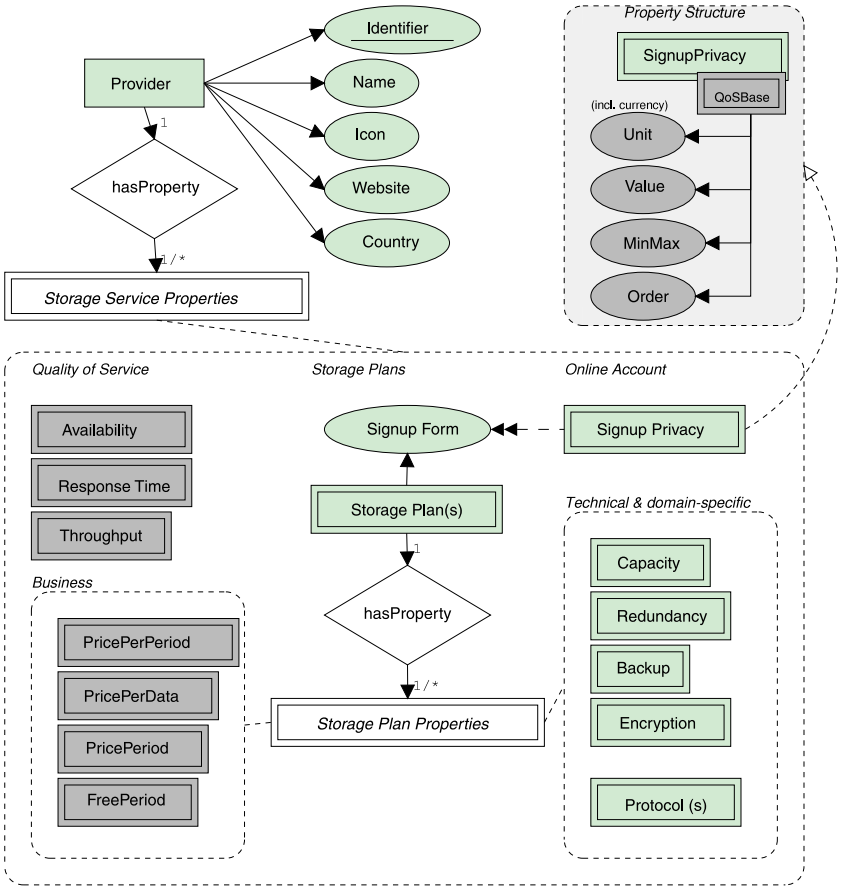

Fig. 3. Simplified entity-relationship diagram of the cloud storage ontology. It consists of domain-independent base ontologies (grey) and the cloud service provider and storage plan entities and attributes (green). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

a restricted optimisation function with auxiliary constraints. When all quality parameters match the user's expectations, the outcome is a RAOC—a Redundant Array of Optimal Cloud Storage Providers. This implies the availability of a configuration mechanism to let the user or an entity on the user's behalf (such as a system administrator or autonomous software agent) specify the preferences over non-functional properties representing the user's expectations.

To give the configuring entity the choice among otherwise functionally similar providers, a storage service directory or marketplace containing cloud storage service descriptions following the proposed ontology is assumed to be available. The realised descriptions will be described in the validation section; the marketplace design and the questions regarding who offers the services and who maintains the descriptions will be omitted.

In addition to publicly available service descriptions, negotiated service usage contracts ranging from granted logins to sophisticated service level agreements will be needed to parametrise the access of the transport to the storage cloud resources. Meta-information about where and when to negotiate and configure the parameters should be taken from the public service description file, whereas the parameters themselves will be subject to a private meta-data store managed by the storage controller.

3.2. Configuration

The choice of suitable storage providers is the task of an administrator with cloud storage controllers running as network proxies, but is left to the users for desktop controllers. In both cases, the controllers require a configuration user interface. It requests preferences over non-functional service properties, generates a goal description (requirement specification) from them and matches it against the registry of storage providers. The result list indicates the ratio of requested and deliverable properties. Matching services without an appropriate local transport implementation are automatically marked as not applicable unless the storage controller provides a facility to dynamically retrieve and integrate those. The user bookmarks candidate services from the filtered list. The selection dialogue is shown in Fig. 4.

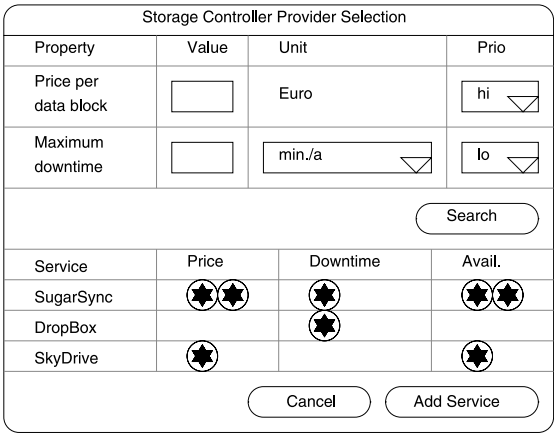

Fig. 4. Graphical user interface layout for the provider selection.

In the next step, the user then creates the service provider set of n storage services and configures it accordingly. Typically, providers require the creation of an account with username and password as minimum data, or with additional personal data such as address and credit card information. They might also require the negotiation of an individual service level agreement

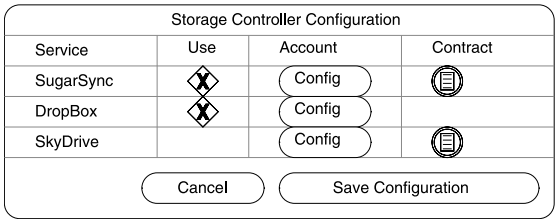

Fig. 5. Layout for the configuration of the provider set.

unless a stock one is used. In an integrated storage system with a marketplace, both configuration aspects are required to be available from an appropriate platform service [4] and be thus available to the cloud storage controller by querying the service for negotiated agreements and filled configuration files. This implies a prior registration of agreement and configuration templates alongside the service descriptions in the broker or matchmaker in use. In contrast to the descriptions and templates, the resulting agreements and configuration files are not public, but are shared between the matchmaker and the clients which created them. This implies an initial account setup at the marketplace site. Hence, for privacy-sensitive setups, running a local instance of the matchmaker platform services with synchronisation from a global instance is recommended. Fig. 5 represents the configuration dialogue.

4. Using storage services

A client-side generic cloud storage controller architecture is proposed to achieve optimality regarding both the RAOC and the data handling within the system. The following seven novel controller characteristics not found in existing approaches are proposed: Selective redundancy, adaptive scheduling, caching, streaming, chunking, sessions and distribution.

4.1. Layered architecture for optimal cloud storage

The design of an improved system to make information dispersion over arrays of optimal cloud storage providers available to users follows a layered architecture. The controller itself is considered to be placed between data-processing applications, such as backup or data sharing software, and the storage cloud. It consists of an upper interface layer which offers access to the applications, a middle layer for the logic needed to disperse and reassemble the data, and a lower layer to transport the dispersed data to the storage servers. The resulting high-level architecture of the cloud management system and its association to the storage service directory is shown in Fig. 6. Each layer will be explained in greater detail from the bottom to the top in the next paragraphs.

Attached to the cloud storage providers is the transport layer. Beside delegating the data to the storage interface routines, it is also responsible for maintaining access efficiency through an optional cache. Furthermore, if required, a full local write-through copy of the data can be kept (see Fig. 7).

The data processing pipeline within the preprocessing layer consists of a single dispersal routine to split (multiplex) data streams into blocks (fragments), with possibly multiple splitter implementations to select from, and a set of pipe-combineable routines which work above the splitter on the file level or below it directly on the block level, as shown in Fig. 8. For more complex scenarios, nested loops can be introduced into the pipeline to yield a hierarchical tree of dispersion and modification modules. All block-level functions can also be applied to the file level, although this might be impractical, e.g. when hiding very large files by steganographic means. The inverse assumption is not necessarily

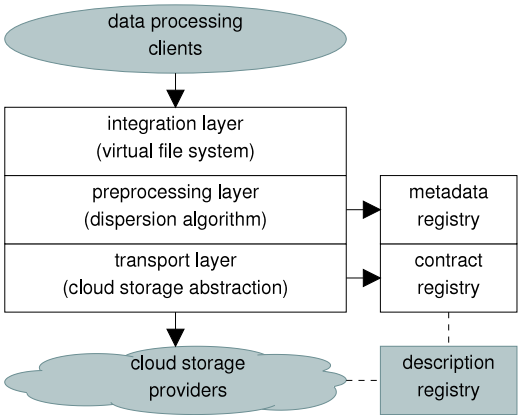

Fig. 6. Abstract architecture of cloud storage management attached to a storage service directory.

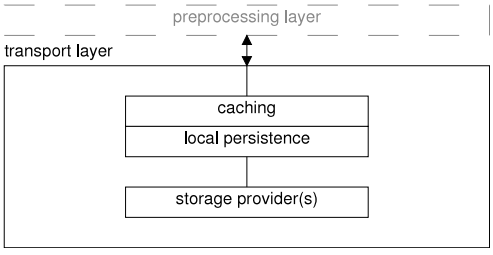

Fig. 7. Inner architecture of the transport layer.

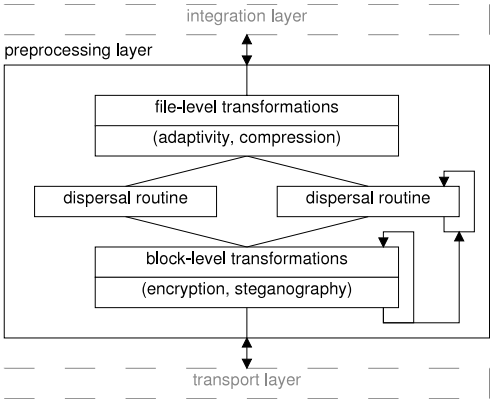

Fig. 8. Inner architecture of the preprocessing layer.

feasible, since many algorithms such as media adaptation (image scaling, audio quality reduction) require a full stream of data or blocks of sizes unknown to generic splitters. The rationale for offering choice among splitters is the ongoing research and progress in terms of dispersion efficiency and implementation efficiency such as multiprocessor support for parallel dispersion and modification module calls.

The dispersion algorithm honours the previously captured user requirements regarding performance, availability and security by applying selective redundancy: None ($n = k$ fragments), full redundancy through replication or secret sharing schemes ($n = k + k^x, x >= 1$), or optimal erasure coding through maximum-distance separable and minimum-bandwidth/-storage regenerating codes ($n = k + m$ fragments). The resulting n fragments are uploaded and retrieved either in serial round-robin order or in parallel. Support for large files requires a special effort. As an intermediate structure between the file and the fragments, variable-sized chunks are introduced, which can be retrieved and updated separately. Each chunk is worked on as soon as the

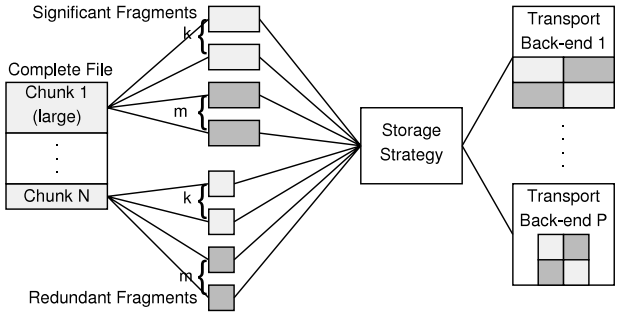

Fig. 9. Multiplexing files into chunks and chunks into fragments.

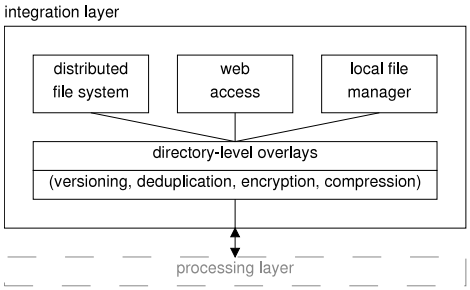

Fig. 10. Inner architecture of the integration layer.

first byte of data arrives at the splitter interface, leading to the treatment of files as data streams, which is suitable for small and embedded devices (e.g. running the controller on a wireless router). Furthermore, sparse files with unused areas such as virtual machine snapshots need to be handled. Fig. 9 shows the relation between chunks and fragments.

The metadata about the fragment locations and checksums would need to be replicated to all devices from which the data in the storage cloud shall be accessed. Hence, the controller needs to support sessions by recursively also storing the metadata database along with the data. In this case, only tiny top-level fragment metadata needs to be maintained on the client which can easily be forwarded to other devices or humans (e.g. as an e-mail link). For the retrieval, first the metadata and then the actual data will be retrieved on demand.

On top of the preprocessing layer, user-level and system-level applications make use of the uniform file-level interface for accessing and manipulating the data. Version control, encryption and deduplication can be inserted as layers for transparent storage of multiple versioned, secure and distinct copies of the files, respectively. At the topmost level, a variety of applications and application-integrated filesystem protocols exist as entry points for humans. Fig. 10 shows the integration layer structure.

The proposed layers allow flexible setups and connection topologies, ranging from single-user desktop backup applications and automated server backup stores with and without multiple providers to enterprise-wide distributed cloud storage controller proxies with high redundancy and throughput. A generalised view is given in Fig. 11.

5. Validation of an optimal cloud storage controller

In this section, we will present our realisation of the previously presented concepts, through an implementation called NubiSave. We start with an overview about the software, proceed with an explanation of the software architecture and modular extensibility thereof which contributes to optimal storage service handling, and round up with a discussion of the limitations resulting from the chosen architecture and implementation technologies.

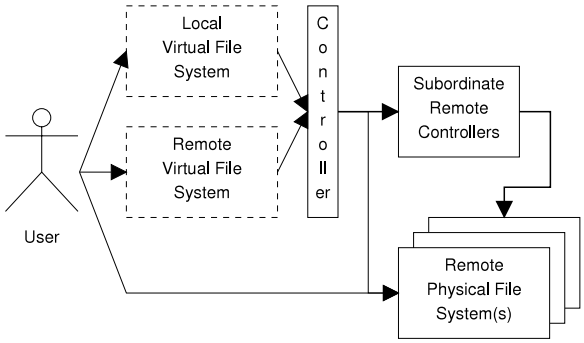

Fig. 11. Generalised topologies of a cloud storage controller.

5.1. Scope of NubiSave

The NubiSave prototype represents a flexible optimal cloud storage controller. The software is designed to be highly modular with well-defined connection points between the components. For instance, there are two configuration user interfaces available, both sharing the same configuration data. A typical NubiSave configuration is shown in Fig. 12. It highlights the hierarchically parallelisable and distributable data storage pipelines as well as some of the most important user choices regarding the storage properties. Additionally, there are multiple modules available, some of them integrated from existing software projects, for the dispersal, modification and transport of data.

5.2. NubiSave architecture and implementation

The NubiSave architecture consists of three major components, following the proposed layered approach.

The bottom layer represents a simple data transport abstraction for each cloud storage protocol. All transport modules implement the Linux Filesystem in Userspace (FUSE) interface, hence there is an increasing choice of additional providers available from the FUSE community, such as davfs2 (WebDAV) and s3fs (Amazon S3).[1] In particular, there is support for configurable SSH-, FTP- or HTTP-accessible hosts, as shown in Fig. 13. A special transport-layer component has been created to be able to add cloud providers easily. The component is named CloudFusion for both its implementation technology and its ability to fuse cloud storage with local storage areas which are accessed through the regular (kernel-space) filesystem for reduced overhead. It is implemented in Python with pluggable transport modules for the providers. Transport modules are available for the commercial service operators SugarSync and DropBox. CloudFusion is based on the fuse.py module. Its extension is supported through popular Python frameworks, namely the Nose unit testing framework and Sphinx-generated documentation. In addition to commercial storage providers and configurable hosts, the inclusion of local filesystems enables keeping one or more of the storage parts under the direct control of the user, for instance on a USB pen drive.

A caching module is available as well; its activation reduces the storage latency considerably at the expense of potential versioning conflicts when accessing the storage area from multiple clients.

For each transport module there is an INI-style configuration file containing both generic attributes (hostname, access protocol, user/password, API key) and provider-specific attributes (folder configuration). In addition, a logging configuration determines the verbosity and placement of log messages.

[1] About 60 FUSE remote filesystem modules are listed at http://sf.net/apps/mediawiki/fuse/index.php?title=NetworkFileSystems.

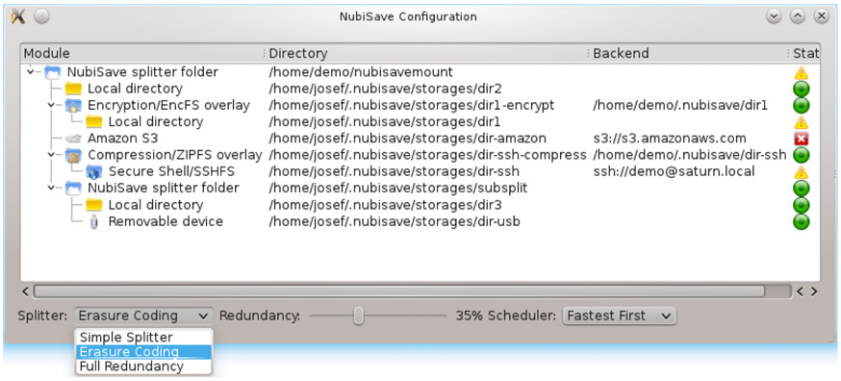

Fig. 12. NubiSave configuration user interface displaying a nested dispersion and transformation module tree.

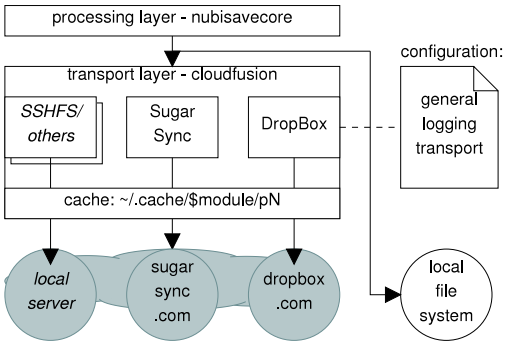

Fig. 13. Available transport modules within the NubiSave prototype.

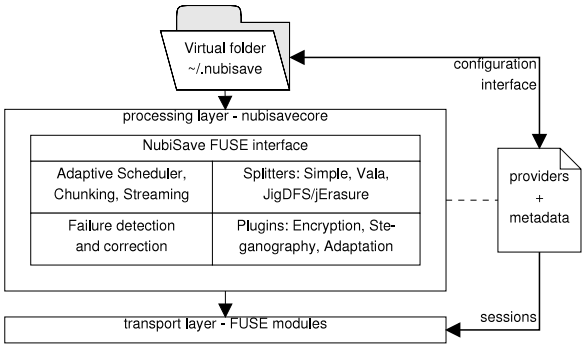

Fig. 14. Processing and available splitter and file/block transformation modules.

The middle layer implements the provider selection, data splitting and distribution optimisation logic and is paired with the user configuration. At the core, a switchable splitter module is responsible for the assembly and disassembly of k data blocks with m redundant blocks for $n = m + k$ providers. Two splitters exist in NubiSave, and one more is scheduled for implementation:

- A simple splitter chops the data blocks without providing redundancy.
- A Jigsaw Distributed File System (JigDFS)-based splitter implemented in Java is also available. It already contains encryption routines to enable plausible deniability [5]. The dispersion is controlled by the algorithms offered by the Java port of the jErasure library but extended with fragment checksums so that failures can be caught early on.
- Finally, a fast splitter is being realised as a Vala library compiled to executable machine code. Its code contains an implementation of the Cauchy Reed–Solomon algorithm for information dispersal [6] adapted from the Cleversafe Java library.

In addition to the splitter, which performs the $1 : k$ and $k : n$ data block transformations, a number of plugins offer functions to be invoked on each block for an $1 : 1$ transformation with varying additional redundancy as a side effect. An encryption plugin is available to encrypt and decrypt data blocks with a symmetric key which is recommended to guarantee confidentiality even in the event of an attacker getting access to all relevant k provider storage areas. Another plugin can be activated to hide the data blocks in media such as photos and songs by applying steganographic routines from the Stepic tool [7]. This won't add any guarantees but is still recommended to lower the risk of attracting attackers into brute-forcing the decryption in the absence of the key. A compression plugin reduces the storage requirements transparently by applying Huffman coding over LZ77 sliding window compression [8], implemented by the deflate algorithm of gzip. The processing layer subarchitecture is shown in Fig. 14.

The candidate storage services for the dispersal are determined from the list of previously user-selected and contract-bound services. Their activation follows a round-robin or parallel strategy. By evaluating the non-functional properties, more sophisticated scheduling strategies can be implemented in the future via additional Java classes. Beside this implementation of selective redundancy and adaptive scheduling, NubiSave also implements caching, streaming, chunking, sessions and distribution as described in the conceptual section.

The upper layer exports the aggregate view on the cloud storage into the user's file system as a virtual partition. It is again implemented as a userspace filesystem (FUSE) module, which makes it portable across Linux systems, easy to install and stackable with other FUSE modules for directory-level encryption, deduplication or versioning [9]. The filesystem API allows direct access by all applications with local access semantics, hence offering a superset of proxy controller approaches. An advantageous side effect of this design is that users no longer need to rely on the existence of clients offered by the providers.[2]

Upon the upper layer, applications make use of the exported virtual partition to realise data management and versioning, if not already intrinsic to the specific setup of stacked overlay file systems. Storage functionality such as backup and sync represents the highest level in the architecture, as shown in Fig. 15.

[2] SugarSync doesn't offer a native Linux client, DropBox only offers one for read-only access.

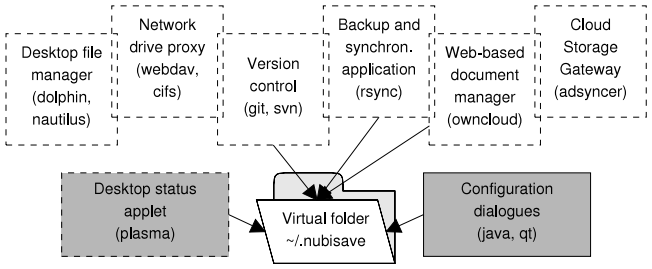

Fig. 15. Integration use cases for NubiSave.

Furthermore, a configuration tool implemented in Java using the Swing user interface library is available, and a status notification applet is a planned addition. Both tools require access to the configuration file besides the virtual partition.

5.3. Storage provider descriptions

Currently, no uniform service description language exists which is capable of expressing all technical protocol and non-technical distinction aspects of heterogeneous storage systems alike. However, for most information about functional and non-functional properties, rich description languages including the semantic Web Services Modelling Language (WSML) [10] and the Unified Service Description Languages (USDL) [11] are suitable candidates. We expect that domain-specific vocabulary will be made available by ongoing research efforts at some point in the near future.

Service descriptions for the cloud storage providers exist in the form of WSML ontologies [10] as instances derived from a storage domain ontology, which in turn uses base ontologies (for QoS, context, prices and other non-technical properties) from the WSMO4IoS ontology set [12]. The descriptions are publicly hosted and maintained by either the providers themselves or by third parties, which could be a global or domain-specific infrastructure-as-a-service service marketplace as alternative to a locally installed matchmaker instance containing the WSML service descriptions. They can also be complemented or substituted with local files maintained by the user, which is a necessity especially for arbitrary local storage areas accessed through LAN protocols such as SSH and WebDAV. An excerpt from the storage provider concepts is shown in Listing 1.

Listing 1: Storage ontology

```
wsmlvariant "wsml-flight"
namespace {qos "wsmo4ios:QoSBase.wsml"}
ontology CloudStorage importsOntology "
    wsmo4ios:QoSBase.wsml"
nonFunctionalProperties
  qosdefinition hasValue "Cloud Storage"
endNonFunctionalProperties
concept MaxDownTime subConceptOf {qos#quality
    , qos#LowerBetter}
  qos#unit impliesType qos#TimeUnit
concept CloudStorage
```

It should be noted that many providers offer several service profiles, for instance to complement a prepaid all-inclusive service with a free, temporally or spatially limited offering. For example, SkyDrive offers 25 GB of storage space to all users for free whereas Amazon S3 offers 5 GB for one year in its free incentive offer for new users and unlimited space in a successive pay-per-use offer. Such constellations can be modelled in WSML by using multiple interfaces within one file and a shared section on general provider description. Listing 2 shows an instance of the storage ontology for the SkyDrive service.

Listing 2: Concrete storage description

```
wsmlvariant "wsml-flight"
namespace {qos "wsmo4ios:QoSBase.wsml"}
webService SkyDrive importsOntology "wsmo4ios
    :CloudStorage.wsml"
interface SkyDriveFree importsOntology {Sky}
ontology Sky importsOntology "wsmo4ios:
    CloudStorage.wsml"
instance MaxDownTime memberOf {cloud#
    MaxDownTime, qos#ServiceSpec}
  qos#value hasValue 15
  qos#unit hasValue qos#MilliSecond
```

NubiSave installs descriptions for seven services: Amazon S3, DropBox, Google Storage, HiDrive, SkyDrive, SugarSync and 4Shared. Furthermore, additional description templates for local and custom services can be retrieved from a configurable instance of the ConQo matchmaker which hosts the WSMO4IoS ontology set we made publicly available.[3]

5.4. Extensibility considerations

While storage provider and transport extensibility is an inherent feature of the service selection process and the FUSE module integration, the processing extensibility requires custom development until the creation of a global module registry turns this requirement into a choice between installation and programming. The interface for additional processing modules consists of two functions, one for reading and one for writing. Both functions take and return one data block for the $1 : 1$ transformation. No assumption is made about the size of the blocks. There is however the assumption that the functions are mutually inverse to each other and can be invoked as an idempotent function pair without noticeable side effects.

The Listing 3 exemplifies the development of an extension through a Python module for data compression.

Listing 3: NubiSave extension module for data compression

```
from util import linux
import tempfile
def decode(data):
  open("gzipfs.gz", "w").write(data)
  return linux.pipe_with_input_file(['gunzip
      -c $IN'], "gzipfs.gz")
def encode(data, path):
  open("gzipfs.gz", "w").write(data)
  return linux.pipe_with_input_file(['gzip -
      cf9 $IN'], "gzipfs.gz")
```

5.5. Remaining limitations of NubiSave

Compared to the analysis of limitations found in existing approaches, NubiSave shows clear advantages through its flexible architecture and focus on optimality. The prototype however reveals some remaining limitations to which solutions are out of scope for this work. We list the remaining issues together with medium-term suggestions for overcoming them in future work.

Static provider configuration. While in most use cases a static configuration of storage providers is a sensible choice, future

[3] WSML ontologies for cloud storage: http://serviceplatform.org/spec/wsmo4ios/.

needs for safely dispersed long-term storage require a zero-maintenance addition of new providers. This is especially true for anticipated or actual failures of storage providers which could be masked by allocating new storage areas and rebuilding the array ahead of time. We assume that the most suitable solution is the configuration of one or multiple directory services with user-recommended or otherwise trusted dynamic additions aligned with the preferences on non-functional properties configured in NubiSave. We envision the uptake of spot markets for cloud resource and utility services and just-in-time clouds over low-scale resources which increases the orientation of offerings along consumer preferences [13].

Manual provider sign-up. In addition to having to search and add providers to the configuration, the user is required to manually create accounts for them in an out-of-band process, for instance through sign-up at their webpages. We assume that automated sign-up wizards can be developed and used when not prohibited by the terms of service.

Single-user operation. Currently, the use cases for NubiSave evolve around single-user (albeit multi-device) operation, especially backup. The addition of multi-user capabilities with selective sharing would need advanced asymmetric key exchange methods or entirely different cryptographic techniques. The use of homomorphic encryption, as proposed in a policy-based non-dispersing RAIC controller [14], appears to be the most sensible concept for sharing dispersed data.

5.6. Performance measurements and analysis

The evaluation of the NubiSave prototype focuses on the quantitative results from performance measurements, leaving formal correctness considerations (beyond random inspection) and usability tests for future work. Nevertheless, NubiSave has been in use by us for several talks and presentations already in which the slides were dynamically retrieved from cloud storage services found through a spot market, confirming the practical usefulness of the tool.

The overall performance is influenced by many factors. Four important contributors to this metric are the degree of redundancy ($m = 0$ to $m = 2k$), the throughput of the splitter module as determined by parallel invocation, the use of the cache (for both metadata and files), and the maximum achievable network throughput as determined by the choice of distribution of storage providers. In practice, though, the outbound network connection will almost always be the limiting factor for desktop users due to it being low-bandwidth, relative to the high-bandwidth computing centre connections used by most commercial and institutional storage providers. The test setup consisted of a virtual Ubuntu Maverick 64 bit machine with 2.9 GB of RAM using 2 Intel Core i5 760 processors with 2.80 GHz frequency each, running on VirtualBox 4.0.8.

The performance of the splitter has been measured with deactivated cache for both write and consecutive read performance. The results are visualised in the diagrams 16 and 17, respectively. As expected, the splitting process performs better for larger block sizes.

In a full-redundancy experimental setup consisting of $k = 3$ blocks over $n = 6$ storage clouds ($1 \times$ Local, $1 \times$ SugarSync, $4 \times$ DropBox) with caching enabled, the maximum used bandwidth reached 4111.68 kB/s, hinting at a per-account throttling rather than a per-connection one for the DropBox storage. In a minimal-redundancy setup of $k = 5$ blocks for the same providers, which works significantly faster at the cost of providing less availability, further distinguishing differences can be seen depending on the use of the cache.

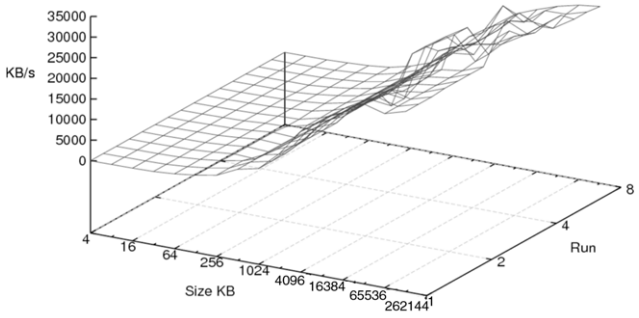

Fig. 16. Splitter write performance.

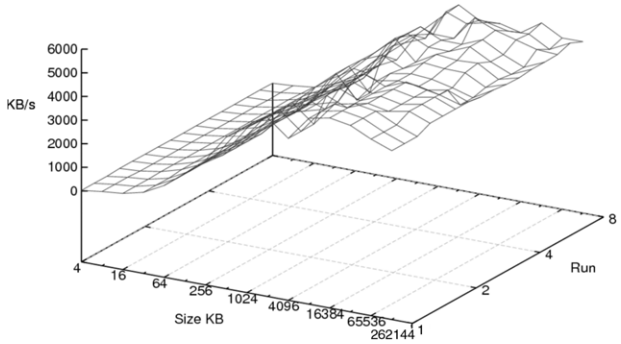

Fig. 17. Splitter read performance.

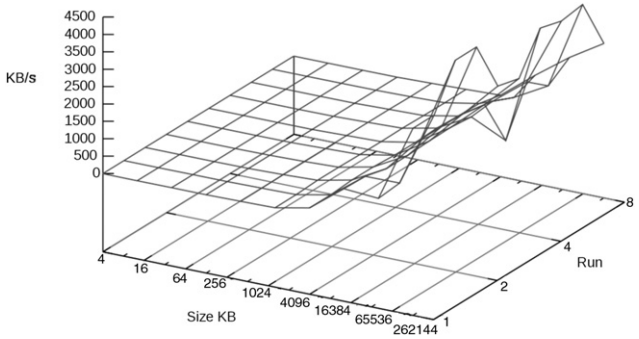

Fig. 18. Minimal-redundancy transport write performance without cache.

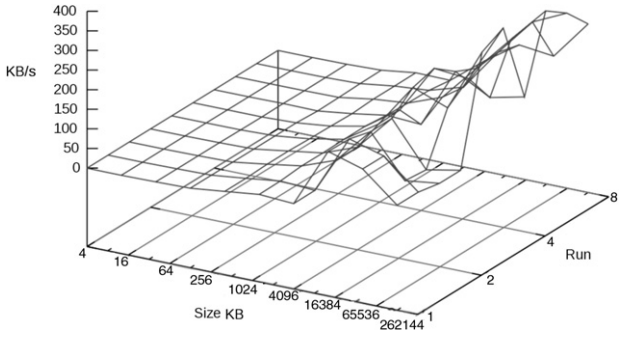

Fig. 19. Minimal-redundancy transport read performance without cache.

Write and consecutive read performance are visualised for both disabled cache (Figs. 18 and 19) and enabled triple-cache (Figs. 20 and 21). The results suggest that future research is needed to find the best combinations of $1 : k$ parallel splitting, $k : n$ block mappings, $n : m$ logical to physical provider mappings and intelligent use of caching.

Table 1
Measurable side effects of using cloud storage arrays.

Operation	Size (MB)	Time (s)	CPU/avg (load)	CPU/max (load)	Mem/avg (MB)	Mem/max (MB)	Net/avg (kB/s)	Net/sum (kB/s)
Write-sparse	1000	39.76	0.7	1.0	23,184	24,708	551.8	37,526
Read-sparse	1000	25.26	0.8	0.8	24,847	24,889	290.5	22,079
Write-mb	1		0.9	1.3	25,816	27,583	340.3	37,439
Read-mb	1		1.2	1.3	27,395	27,458	345.5	22,113
Append	0		1.2	1.2	27,398	27,398	55.8	1,675

Table 2
Effects of using the streaming mode.

Operation	Size (MB)	Time (s)	CPU/avg (load)	CPU/max (load)	Mem/avg (MB)	Mem/max (MB)	Net/avg (kB/s)	Net/sum (kB/s)
Write	1	0.29	0.3	0.3	73,801	73,801	102.76	4,212.95
Write	10	1.67	0.35	0.4	74,822	74,823	521.99	21,401.6
Write	100	20.02	0.69	0.9	74,830.6	74,833	2.99	449.83
Write	1000	286.87	2.83	4.6	535,762	536,964	0.084	24.57
Stream	1	0.39	1.1	1.1	38,112	38,112	78.04	3,277.65
Stream	10	24.51	1.39	1.4	38,918.2	41,322	877.33	31,583.9
Stream	100	217.48	1.93	2.4	39,467.1	41,498	1343.96	309,111
Stream	1000	1955.40	4.88	6.5	38,238.9	43,430	1887.2	3,702,690

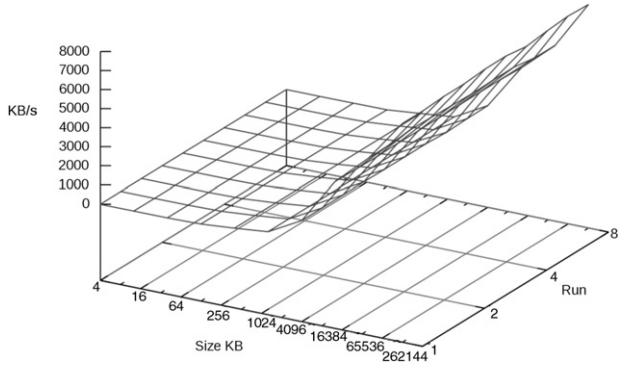

Fig. 20. Minimal-redundancy transport write performance with cache.

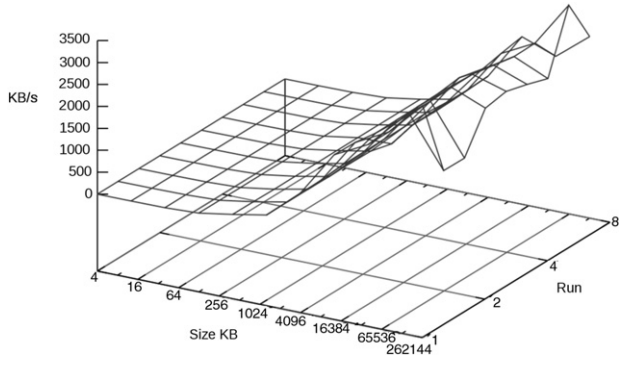

Fig. 21. Minimal-redundancy transport read performance with cache.

A second analysis has been performed to measure the side effects of using cloud storage arrays. The scenario for this analysis consists of an over-full-redundancy (200%) configuration for $k = 1$ blocks over $n = 3$ storage clouds (T-Online, Strato, Wuala). The Table 1 contains the side effects values for the following operations: Writing a sparse file of 1000 MB with 1 byte content; reading this very file; writing 1 MB in the middle of this file; reading this very 1 MB block; and appending 15 bytes to the file.

A third analysis confirms the memory saving effect when using the streaming mode for writing files to the backend storage providers as soon as a reasonable sized chunk is available in the FUSE layer. Table 2 shows the effect for continuously written files of varying sizes. The excessive use of memory and the activation of swap space are alleviated when not buffering all data but instead using the streaming support.

Due to space constraints, only selected statistics are included here, excluding further aspects such as re-write/re-read rounds. We offer our experiment recipes, environment description, input data, data generation scripts and comparative output numbers at a public research result comparison site to encourage authors of dispersing RAIC/RAOC systems to produce corresponding statistics and graphs.[4]

6. Related work

Over two decades ago, the introduction of Redundant Arrays of Inexpensive Disks (RAID) made the case to combine several local disks in ways which balance cost, data safety and performance [15]. The RAID levels allow for mirroring data on disks or partitions of identical size, striping physical disks independent of their size into a logical one, or a combination thereof with an optional disk for storing checksums. Later, this concept was extended to the combination of network block devices across servers [16]. This research focused more on highly-available storage and therefore on mirroring techniques. Both local and network RAID setups still assume experienced users or administrators for a proper setup and maintenance in case of disk failures.

Since approximately the appearance of network storage redundancy, personal online storage has been increasingly offered by commercial providers, often backed by RAID storage at the provider's discretion. Typically, a proprietary web access or standardised protocols such as WebDAV and a storage area of fixed size have been included in the offers. More recently, cloud computing has brought its own flavour, called cloud storage. In this model, the storage area increases on demand and its usage is billed accordingly. In addition, the concern of high availability is covered by many of these offers by providing redundancy over geographically distributed data centres. Furthermore, new protocols based on Web Services have been emerging to ensure that only high-level file access is possible as opposed to low-level filesystem or partition access. Setting up a RAID over these storage providers as network block devices is hence not possible.

A disadvantage of cloud storage from a single provider is that despite the guaranteed high availability, the operation and

[4] Research result comparison datasets are shared at http://areca.co/?q=cloudstorage.

organisation of the storage area is still subject to the provider's policies. For example, despite claims to use RAID, total data losses occurred in the past.[5] Despite claims to protect the access to the data, total data losses might have occurred in the presence of an attacker.[6] These issues represent both a trust and acceptance problem as well as a potential safety problem.

Systems for dispersing information over an array of cloud storage providers have thus been suggested as RAIC systems in analogy to RAID for local disks. Similar to how RAID works, there is a software controller (independent from the storage providers) with varying placement and redundancy strategies, and there are a variety of RAIC levels depending on the requirements of the user. There are filesystems such as CORNFS [17] which store their contents redundantly on the file-level into multiple storage providers, following a RAID-1 (mirroring) approach. They protect against data loss, but not against full access with subsequent decryption attempts. Therefore, partially replicated data blocks are the preferred redundancy approach. The redundancy is achieved by executing information dispersal algorithms (IDA), sometimes colloquially referred to as obfuscation algorithms, which build on the theory of secret sharing [18]. Well-known IDAs are (in increasing order of efficiency) Cauchy Reed–Solomon [6], Row-Diagonal Parity or RAID-6 Liberation Codes [19].

One RAIC information dispersal system has recently been proposed for use within enterprises [20]. It runs as a central proxy connected to several cloud storage providers, and offers transparent integration with enterprise client systems through the common remote filesystem protocol CIFS. The dispersal algorithms encompass Blaum–Roth and Liberation coding implemented by the jErasure library [21], and encryption of the resulting blocks is performed by an AES algorithm implemented by the Bouncy Castle library. Further RAIC prototypes have been suggested for resilience against byzantine failures [22] and as a protection against storage media theft [23].

Beyond the RAIC systems, a RAOC system which considers provider characteristics has also been implemented before. This system has been implemented and evaluated as a web portal [24]. It disperses its data through the Liberation algorithm. The dispersed data is however not subject to flexible modification. Furthermore, the actual selection of optimal storage providers remains unclear.

The presented approaches spread all of their data across remote cloud storage providers, while others retain one (relatively small) part on the client side, for instance on a removable device. Likewise, most approaches use erasure codes which resembles the RAIC-5 setup, as opposed to simple RAIC-0/1 setups.

However, despite representing a major step forward with increasing potential for widespread use and commercialisation,[7] there are still shortcomings in the currently available information-dispersing RAIC and RAOC systems. They implement a fixed set of algorithms, functionality and topology without configurable extensibility. They typically don't take arbitrary non-functional properties into account when selecting a storage provider. Furthermore, they don't support a semi-automatic inclusion of new providers into the storage pool and instead still require a manual sign-up with potential providers. Finally, they don't yet provide an easy-to-use interface to desktop users, who are one of the biggest target groups for cloud computing offerings.

[5] The Amazon EC2 service irreversibly lost customer data in a widely publicised incident on April 21, 2011.

[6] The Dropbox service accounts were publicly accessible for several hours on June 21, 2011.

[7] Commercially available dispersing RAIC offerings include the Fraunhofer FOKUS/eGovCD application TrustedSafe and the Academic dsNet initiative from Cleversafe.

7. Conclusion

The article has presented a systematic approach for achieving optimality in cloud storage services along the provider's and consumer's iterations of the service lifecycle. Its contributions over existing works are a structural definition of storage systems and storage optimality, a storage service ontology as prerequisite for optimally aggregated services, and an optimality-conscious cloud storage controller architecture. Building upon our analysis of existing distributed cloud storage techniques and their weaknesses, we have hence created a more generic and extensible architecture which serves as blueprint for building optimal cloud storage controllers encompassing a superset of the most important existing features. Our prototype NubiSave, which is freely available,[8] implements most of these RAOC concepts and encourages researchers to overcome the remaining limitations by following our suggestions.

For the future, we plan to integrate NubiSave with popular web-based cloud storage frontends to achieve a critical mass of actual users. Furthermore, we plan to explore distributed cloud storage ecosystems including resource service registration markets, community-driven resource sharing and autonomous selection and configuration agents. This will lead to a detailed analysis of the offering and feedback phases and complete the storage service lifecycle.

Acknowledgments

This work has received funding under project number 080949-277 by means of the European Regional Development Fund (ERDF), the European Social Fund (ESF) and the German Free State of Saxony. It has also been funded by a scholarship from the Brazilian National Council for Scientific and Technological Development—CNPq (PDJ 159716/2011-0).

References

[1] J. Spillner, G. Bombach, S. Matthischke, J. Müller, R. Tzschichholz, A. Schill, Information dispersion over redundant arrays of optimal cloud storage for desktop users, in: 4th International Conference on Utility and Cloud Computing, UCC, Melbourne, Australia, 2011, pp. 1–8.
[2] I. Braun, S. Reichert, J. Spillner, A. Strunk, A. Schill, Zusicherung nichtfunktionaler Eigenschaften und Dienstgüte im future internet of services, PIK—Praxis der Informationsverarbeitung und Kommunikation 31 (04/08) (2008) 225–231.
[3] Y. Badr, A. Abraham, F. Biennier, C. Grosan, Enhancing web service selection by user preferences of non-functional features, in: Proceedings of the 4th International Conference on Next Generation Web Services Practices, Seoul, South Korea, 2008.
[4] J. Spillner, B. Buder, T. Schiefer, A. Schill, Contract services for post-discovery guarantee management, in: INSTICC-Tagungsband: 4th International Conference on Software and Data Technologies/Special Session on ACT4SOC and e-Health Services, vol. 2, Sofia, Bulgaria, 2009, pp. 369–375.
[5] J. Bian, R. Seker, JigDFS: a secure distributed file system, in: IEEE Symposium on Computational Intelligence in Cyber Security, CICS, Nashville, Tennessee, USA, 2009, pp. 76–82.
[6] J.S. Plank, L. Xu, Optimizing Cauchy Reed–Solomon Codes for fault-tolerant network storage applications, in: 5th IEEE International Symposium on Network Computing Applications, NCA, Cambridge, Massachusetts, USA, 2006.
[7] N. Provos, P. Honeyman, Hide and seek: an introduction to steganography, IEEE Security and Privacy 1 (3) (2003) 32–44.
[8] L.P. Deutsch, DEFLATE compressed data format specification version 1.3, IETF Request for Comments 1951, 1996.
[9] P.A.D. Brian Cornell, F.E. Bustamante, Wayback: a user-level versioning file system for Linux, in: USENIX Annual Technical Conference, Boston, Massachusetts, USA, 2004.
[10] C. Feier, D. Roman, A. Polleres, J. Domingue, M. Stollberg, D. Fensel, Towards intelligent web services: Web Service Modeling Ontology (WSMO), in: Proceedings of the International Conference on Intelligent Computing, ICIC, Hefei, China, 2005.
[11] J. Cardoso, M. Winkler, K. Voigt, A service description language for the internet of services, in: Proceedings First International Symposium on Services Science, ISSS, Leipzig, Germany, 2009.

[8] NubiSave webpage: http://nubisave.org.

[12] G. Stoyanova, B. Buder, A. Strunk, I. Braun, ConQo—a context- and QoS-aware service discovery, in: Proceedings of IADIS Intl. Conference WWW/Internet, Freiburg, Germany, 2008.
[13] R. Costa, F. Brasileiro, G.L. de Souza Filho, D.M. Sousa, Just in time clouds: enabling highly-elastic public clouds over low scale amortized resources, Tech. Rep. TR-3, Federal University of Campina Grande / Federal University of Paraíba, Brazil, 2010.
[14] M. Mowbray, S. Pearson, Y. Shen, Enhancing privacy in cloud computing via policy-based obfuscation, The Journal of Supercomputing 51 (3) (2010) 1–25.
[15] D.A. Patterson, G. Gibson, R.H. Katz, A case for Redundant Arrays of Inexpensive Disks, RAID, Tech. Rep. CB/CSD-87-391, EECS Department, University of California, Berkeley, 1987.
[16] P. Reisner, DRBD—Distributed Replicated Block Device, 9th Linux System Technology Conference, Cologne, Germany, 2002.
[17] I.C. Blenke, CORNFS, FUSE module software project available from fuse.sf.net, 2005.
[18] A. Shamir, How to share a secret, Communications of the ACM 22 (11) (1979) 612–613.
[19] J.S. Plank, The RAID-6 Liberation codes, in: Proceedings of the 6th USENIX Conference on File and Storage Technologies, FAST, San José, California, USA, 2008, pp. 97–110.
[20] R. Seiger, S. Groß, A. Schill, SecCSIE: a secure cloud storage integrator for enterprises, in: 13th IEEE Conference on Commerce and Enterprise Computing, Workshop on Clouds for Enterprises, Luxembourg, Luxembourg, 2011, pp. 252–255.
[21] J.S. Plank, S. Simmerman, C.D. Schuman, Jerasure: a library in C/C++ facilitating erasure coding for storage applications, Tech. Rep. UT-CS-08-627, University of Tennessee, Knoxville, Tennessee, USA, 2008.
[22] A. Bessani, M. Correia, B. Quaresma, F. André, P. Sousa, DEPSKY: dependable and secure storage in a cloud-of-clouds, in: Proceedings of EuroSys, Salzburg, Austria, 2011.
[23] N. Zhang, J. Jing, P. Liu, Removing the laptop on-road data disclosure threat in the cloud computing era, in: Proceedings of the 6th International Conference on Frontier of Computer Science and Technology, FCST, IEEE Digital Library, 2011.
[24] M. Schnjakin, C. Meinel, Plattform zur Bereitstellung sicherer und hochverfügbarer Speicherressourcen in der Cloud, in: Sicher in die digitale Welt von Morgen—12. Dt. IT-Sicherheitskongress des BSI, SecuMedia Verlag, Bonn, Germany, 2011.

Josef Spillner is a researcher at the Chair for Computer Networks at Technische Universität Dresden and a visiting researcher to Universidade Federal de Campina Grande. Initially working on service user interface generation techniques, he pursued his Ph.D. in the field of Internet of Services platforms and quality assurance. Currently, he works on flexible user-oriented Cloud Computing architectures. Besides being the author of scientific publications and several book chapters, he has also driven the Open Source Service Platform Research initiative.

Johannes Müller is a student of information technology at Technische Universität Dresden. He recently completed his study thesis in the Chair for Computer Networks's FlexCloud team about NubiSave, a RAID-like storage system combining several cloud providers.

Alexander Schill is a professor for Computer Networks at Technische Universität Dresden. His major research interests are distributed systems and middleware, high performance communication and multimedia, and advanced teleservices such as teleteaching and teleworking. He holds a M.Sc. and a Ph.D. in Computer Science from the University of Karlsruhe, Germany, and received a Dr. H.C. from Universidad Nacional de Asunción, Paraguay. He is the author or co-author of a large number of publications on computer networking, including several books.