Information Dispersion over Redundant Arrays of Optimal Cloud Storage for Desktop Users

Josef Spillner, Gerd Bombach, Steffen Matthischke, Johannes Müller, Rico Tzschichholz, Alexander Schill
Faculty of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
Email: {josef.spillner,alexander.schill}@tu-dresden.de

***Abstract*—Data storage in cloud computing centres is gaining popularity for personal and institutional data backups 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 so-called cloud storage is limited in today's systems which severely impacts the acceptance and adoption rates. Therefore, we introduce a novel cloud storage management architecture which combines storage resources from multiple providers so that redundancy, security and other non-functional properties can be adjusted adequately to the needs of the user. Furthermore, we present NubiSave, a user-friendly implementation with configurable adequate overhead which runs on and integrates into contemporary desktop systems. Finally, a brief analysis of measurements performed by us shows how well the system performs in a real distributed cloud storage environment.**

I. Introduction

Service providers on the Internet are always faced with the question of scaling their systems according to the usage patterns resulting from the service behaviour and popularity. Cloud computing has been introduced as paradigm to provide elasticity to software provided as a service. From a consumer perspective, the cloud also refers to giving up control of data processing in return for comfortable and social usage of Internet applications. A survey and taxonomy of selected cloud computing systems can be found in [1].

Reversing the renunciation of control is not easily possible for generic services in the cloud. However, in the case of cloud data storage, the consumer is able to apply the techniques from vendor relationship management [2], [3]. This means that the consumer gains the tools to enter and leave relations with cloud storage providers dynamically while still being able to rely on having the data safely stored in the cloud. One promising realisation in this context is applying the concept of redundant combined storage areas into a larger logical layer over which the data is dispersed across provider boundaries.

We argue that having such a system running on the devices of consumers increases the flexibility and safety of data storage in the cloud. The results of our work show that this realisation is feasible and practical even for average desktop users.

The remainder of this paper is structured as follows: First, we summarise the development towards cloud storage and existing cloud storage management systems. Then, we analyse remaining weaknesses and derive a more flexible and practical architecture for such a system. The remaining sections present our desktop-integrated prototype NubiSave, the results of performance experiments with it, and a conclusive summary with an outlook on how to integrate cloud storage with information dispersion into personal cloud control systems.

II. Existing Cloud Storage Management Approaches

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 [4]. 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 [5]. 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 about 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 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 [1]. Despite claims to protect the access to the data, total data losses might have occurred in the presence of an attacker[2]. 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 like CORNFS [6] 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 [7]. Well-known IDAs are (in increasing order of efficiency) Cauchy Reed-Solomon [8], Row-Diagonal Parity or RAID-6 Liberation Codes [9].

One such information-dispersing RAIC system has recently been implemented and evaluated [10]. It runs as a web portal and disperses its data through the Liberation algorithm. The dispersed data is however not encrypted.

Another RAIC information dispersal system has recently been proposed for use within enterprises [11]. 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 [12], and encryption of the resulting blocks is performed by an AES algorithm implemented by the Bouncy Castle library.

Both 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, both 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[3], there are still shortcomings in the currently available information-dispersing RAIC 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 signup 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.

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

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

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

III. Optimal Cloud Storage Management Concept and Architecture

This section introduces the conceptual extension of RAIC towards optimal storage arrays called RAOC. A generic cloud storage controller architecture is proposed to utilise this extended concept for overcoming the shortcomings identified in the previous section.

A. Concept for Optimal Cloud Storage

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 [13]. This makes it possible to consider both subjective metrics like trust in a storage provider and objective metrics like 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 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 a person on the user's behalf (such as a system administrator) specify the preferences over non-functional properties representing the user's expectations.

To give the configuring person the choice among otherwise functionally similar providers, a storage service directory is assumed to be available. Currently, no uniform service description language exists which is capable to express 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) [14] and the Unified Service Description Languages (USDL) [15] are suitable candidates. We expect that domain-specific vocabulary will be made available by ongoing research efforts at some point in the near future. 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.

B. 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 system 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.

A high-level architecture of the cloud management system and its association to the storage service directory is shown in Fig. 1. The next paragraphs explain each layer in greater detail from the bottom to the top.

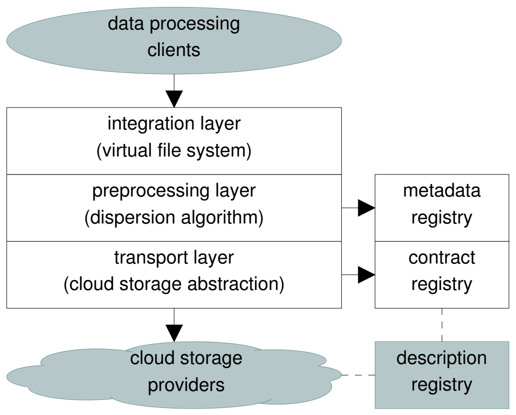

data processing clients
integration layer (virtual file system)
preprocessing layer (dispersion algorithm)
transport layer (cloud storage abstraction)
metadata registry
contract registry
cloud storage providers
description registry

Fig. 1. Abstract architecture of cloud storage management attached to a storage service directory

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. This design has implications on the use of the storage services with several clients, which are not subject to the current work.

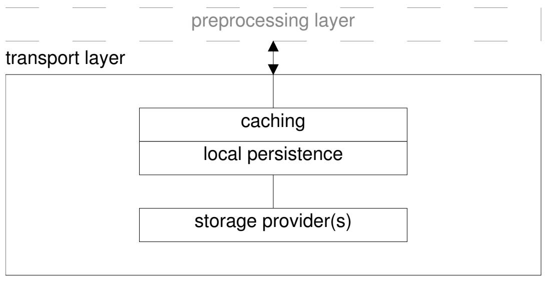


Fig. 2. Inner architecture of the transport layer

The data processing pipeline within the preprocessing layer consists of a single dispersal routine to split data streams into blocks, 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 3. 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 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 IDA efficiency and implementation efficiency like multiprocessor support for parallel IDA calls.

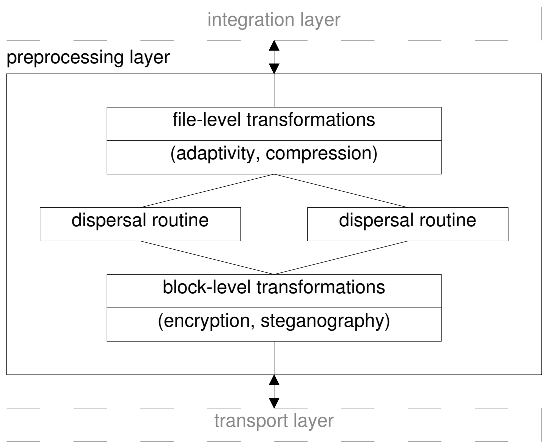

integration layer
preprocessing layer
file-level transformations
(adaptivity, compression)
dispersal routine
dispersal routine
block-level transformations
(encryption, steganography)
transport layer

Fig. 3. Inner architecture of the preprocessing layer

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. A version control layer can be inserted for transparent storage of multiple versioned copies of the files. At the topmost level, a variety of applications and application-integrated filesystem protocols exist as entry point for humans. Fig. 4 shows the integration layer structure.

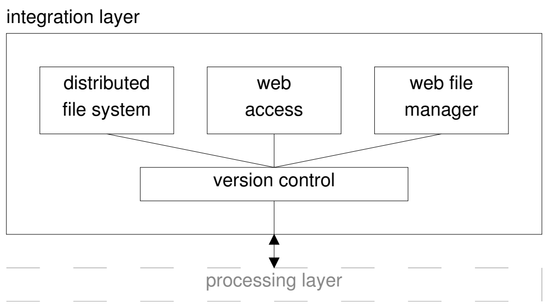


Fig. 4. Inner architecture of the integration layer

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 cloud storage controller proxies with high redundancy. A generalised view is given in Fig. 5.

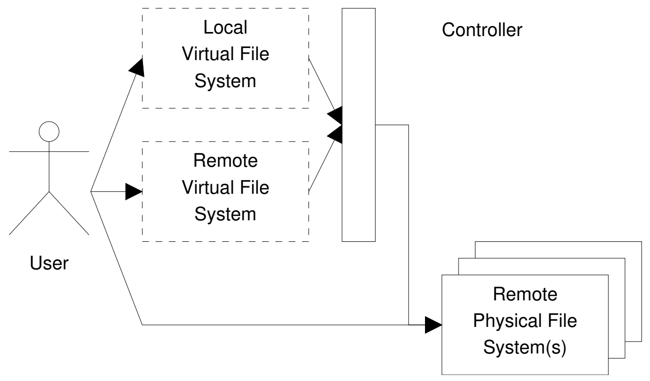


Fig. 5. Generalised topologies of a cloud storage controller

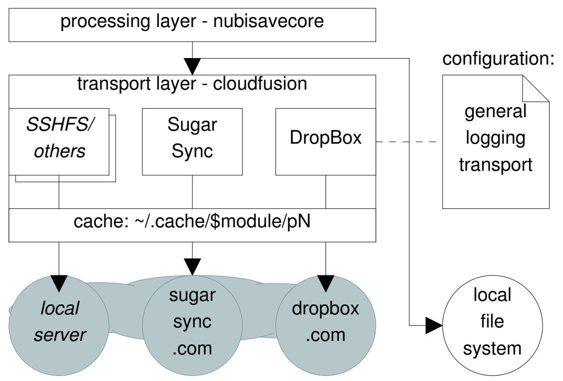


Fig. 6. Available transport modules within the NubiSave prototype

IV. NubiSave - Dispersed Cloud Storage for the Desktop

In this section, we will present our realisation of the previously presented concepts. We start with an explanation of the software architecture and modular extensibility thereof, proceed with an explanation of the storage provider server descriptions and contracts as well as their configuration, and round up with a discussion of the limitations resulting from the chosen architecture and implementation technologies.

A. 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. The component is implemented in Python with pluggable transport modules for the providers. Transport modules are available for the commercial service operators SugarSync and DropBox. 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, like davfs2 (WebDAV) and s3fs (Amazon S3)[4]. In particular, there is support for configurable SSH-accessible hosts (using SSHFS), as shown in Fig. 6. 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. 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. 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.

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.

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

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.

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 [16].
- 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 [8] 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 into media such as photos and songs by applying steganographic routines from the Stepic tool [17]. 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 [18], implemented by the deflate algorithm of gzip. The processing layer subarchitecture is shown in Fig. 7.

The candidate storage services for the dispersal are determined from the list of previously user-selected and contract-

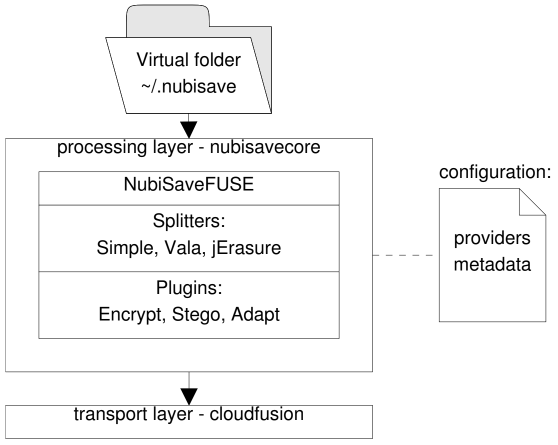


Fig. 7. Processing and available splitter and file/block transformation modules

bound services. Their activation follows a round-robin strategy. By evaluating the non-functional properties, a more sophisticated scheduling algorithm could be realised in the future.

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 partition-level encryption or versioning [19]. 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[5].

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 represents the highest level in the architecture, as shown in Fig. 8. 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 beside the virtual partition.

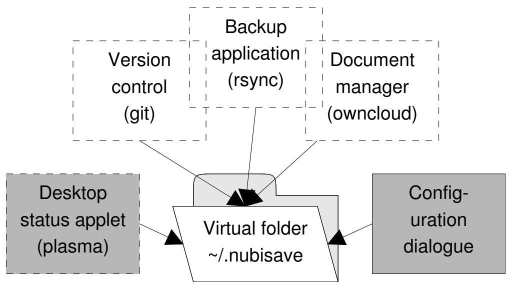


Fig. 8. Integration use cases for NubiSave

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

B. Extensibility Considerations

While storage provider and transport extensibility is an inherent feature of the service selection process and the CloudFusion 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 1 exemplifies the development of an extension through a Python module for data compression.

Listing 1. 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")
```

C. Storage Provider Descriptions

Service descriptions for the cloud storage providers exist in the form of WSML ontologies [14] 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 ConQo matchmaker [20]. The descriptions are publicly hosted and maintained by either the providers themselves or by third parties. 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 SSHFS. An abbreviated form of the storage provider concepts is shown in Listing 2.

Listing 2. Storage ontology

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

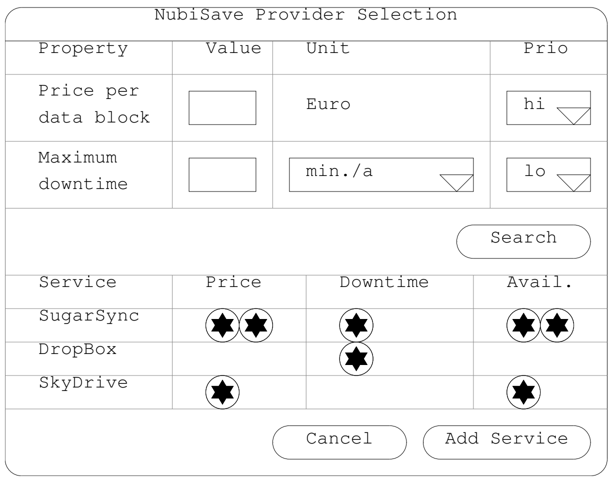
NubiSave installs descriptions for six services: Amazon S3, DropBox, Google Storage, HiDrive, SkyDrive and SugarSync. It should be noted that many providers offer several service profiles, for instance to complement a prepaid all-inclusive service with a free, temporally or spatially limited offering. 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 3 shows an instance of the storage ontology for the SkyDrive service.

Listing 3. Concrete storage description

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

D. Configuration

The choice of suitable storage providers is the task of an administrator with proxy controllers, but is left to the users for desktop controllers. Therefore, NubiSave includes a configuration user interface. It requests preferences over non-functional service properties, generates a WSML goal description from them and matches it against the (pre-configured) registry of storage providers which could be a global or domain-specific infrastructure-as-a-service service marketplace in addition to the locally installed matchmaker instance containing the WSML service descriptions. The result list indicates the ratio of requested and deliverable properties. Matching services without an appropriate transport module are automatically marked as not applicable. The user bookmarks candidate services from the filtered list. The selection dialogue is shown in Fig. 9.



NubiSave Provider Selection

Property	Value	Unit	Prio
Price per data block		Euro	hi
Maximum downtime		min./a	lo

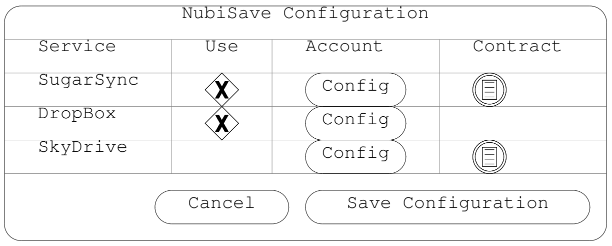
Search

Service	Price	Downtime	Avail.
SugarSync	★★	★	★★
DropBox		★	
SkyDrive	★		★

Cancel | Add Service

Fig. 9. 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. They might as well require the negotiation of an individual service level agreement unless a stock one is used. Both configuration aspects have been implemented in the Contract Wizard platform service [21] and are thus available to NubiSave by querying the service for negotiated agreements and filled configuration files. This implies a prior registration of agreement and configuration templates alongside the WSML descriptions in the ConQo matchmaker. In contrast to the descriptions and templates, the resulting agreements and configuration files are not public but shared between the matchmaker and the clients which created them. This implies an initial account setup at the matchmaker site. Hence, for privacy-sensitive setups running a local instance of the Contract Wizard is recommended. Fig. 10 represents the configuration dialogue.



NubiSave Configuration

Service	Use	Account	Contract
SugarSync	☒	Config	▤
DropBox	☒	Config	
SkyDrive		Config	▤

Cancel | Save Configuration

Fig. 10. Layout for the configuration of the provider set

E. Remaining limitations of NubiSave

Compared to the analysis of limitations found in existing approaches, NubiSave shows clear advantages through its flexible architecture. 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 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 [22].
Manual provider signup. 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 signup at their webpages. We assume that automated signup wizards can be developed and used when not

prohibited by the terms of service. *Single-user, single-device operation.* Currently, the use cases for NubiSave evolve around single-user and single-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 [23], appears to be the most sensible concept for sharing dispersed data. Gaining multi-device support would require the synchronisation of the privately held configuration.

V. Performance Measurements and Analysis

The evaluation of the NubiSave prototype has focused on the quantitative results from performance measurements, leaving formal correctness considerations (beyond random inspection) and usability tests for future work. The overall performance is influenced by many factors. Four important contributors to this metric are the degree of redundancy ($n = 0$ to $2k$), 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 64bit 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 11 and 12, respectively. As expected, the splitting process performs better for larger block sizes.

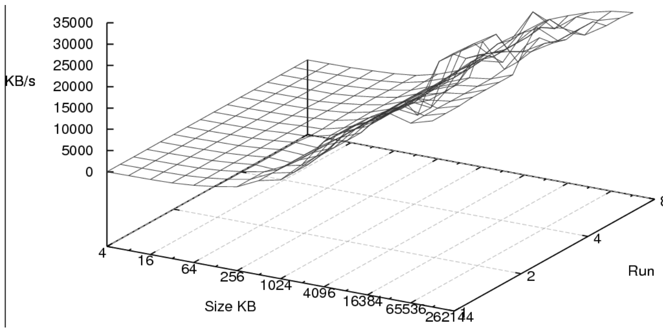


Fig. 11. Splitter write performance

In a full-redundancy experimental setup consisting of $k = 3$ blocks over $n = 6$ storage clouds (1xLocal, 1xSugarSync, 4xDropBox) 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

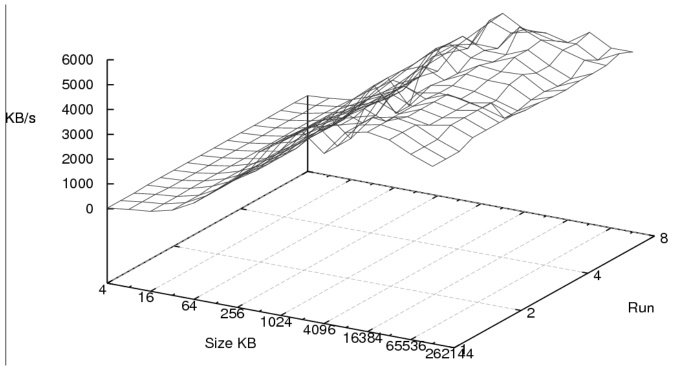


Fig. 12. Splitter read performance

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.

Write and consecutive read performance are visualised for both disabled cache (Figures 13, 14) and enabled triple-cache (Figures 15, 16).

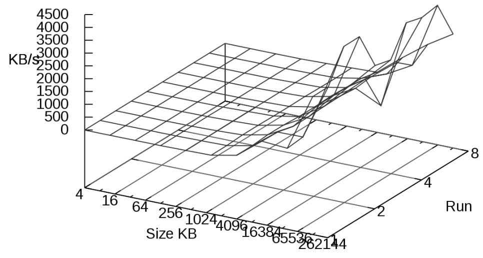


Fig. 13. Minimal-redundancy transport write performance without cache

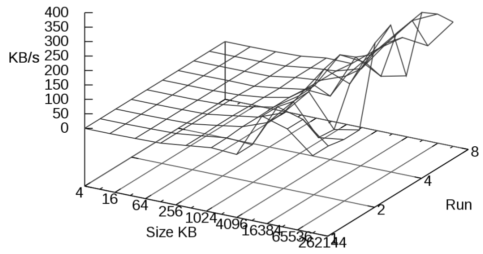


Fig. 14. Minimal-redundancy transport read performance without cache

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. Due to space constraints, only selected statistics are included here, excluding further aspects like 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

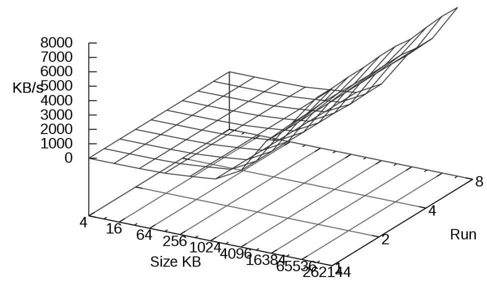


Fig. 15. Minimal-redundancy transport write performance with cache

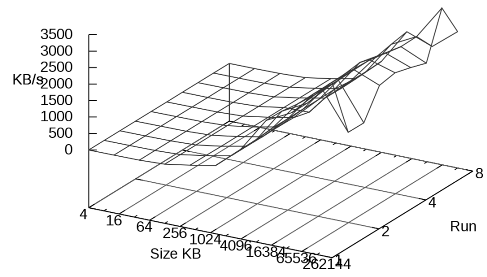


Fig. 16. Minimal-redundancy transport read performance with cache

RAIC/RAOC systems to produce corresponding statistics and graphs[6].

VI. Summary and Outlook

By presenting recent research on data dispersion via cloud storage providers, we have first presented an overview about existing techniques and identified weaknesses. Building upon this analysis, we have 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[7], implements most of these RAOC concepts and encourages researchers to overcome the remaining limitations by following our suggestions. In the near future, we plan to merge the natively compiled IDA implementation from SecCSIE [11] to gain performance, and its enterprise integration features to gain a versatile personal desktop/central proxy topology choice for hosting the controller. In addition, we plan to add checksums for higher robustness, dynamic prioritisation of higher-performing providers and recovery.

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. The information in this document is provided as is, and no guarantee or warranty is given that the information is fit for any particular purpose.

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

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

References

[1] B. P. Rimal, E. Choi, and I. Lumb, "A Taxonomy and Survey of Cloud Computing Systems," in *Proc. of the Fifth Intl. Joint Conference on Networked Computing, Advanced Information Management and Services, and Digital Content, Multimedia Technology and its Applications (NCM)*, August 2009, Seoul, Korea.

[2] D. Searls *et al.*, "Project VRM," http://cyber.law.harvard.edu/research/projectvrm, 2011.

[3] R. Fisk, "A Customer Liberation Manifesto," *Service Science*, vol. 1, no. 3, pp. 135–141, 2009.

[4] D. A. Patterson, G. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," EECS Department, University of California, Berkeley, Tech. Rep. CB/CSD-87-391, December 1987.

[5] P. Reisner, "DRBD - Distributed Replicated Block Device," 9th Linux System Technology Conference, September 2002, Cologne, Germany.

[6] I. C. Blenke, "CORNFS," FUSE module software project available from fuse.sf.net, 2005.

[7] A. Shamir, "How to Share a Secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, November 1979.

[8] J. S. Plank and L. Xu, "Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Network Storage Applications," in *5th IEEE International Symposium on Network Computing Applications (NCA)*, July 2006, Cambridge, Massachusetts, USA.

[9] J. S. Plank, "The RAID-6 Liberation codes," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)*, February 2008, pp. 97–110, San José, California, USA.

[10] M. Schnjakin and 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, May 2011, Bonn, Germany.

[11] R. Seiger, S. Groß, and A. Schill, "SecCSIE: A Secure Cloud Storage Integrator for Enterprises," in *Appears in: International Workshop on Clouds for Enterprises*, September 2011, Luxembourg, Luxembourg.

[12] J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: A Library in C/C++ Facilitating Erasure Coding for Storage Applications," University of Tennessee, Knoxville, Tennessee, USA, Tech. Rep. UT-CS-08-627, August 2008.

[13] Y. Badr, A. Abraham, F. Biennier, and 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*, October 2008, Seoul, South Korea.

[14] C. Feier, D. Roman, A. Polleres, J. Domingue, M. Stollberg, and D. Fensel, "Towards Intelligent Web Services: Web Service Modeling Ontology (WSMO)," in *Proceedings of the International Conference on Intelligent Computing (ICIC)*, August 2005, Hefei, China.

[15] J. Cardoso, M. Winkler, and K. Voigt, "A Service Description Language for the Internet of Services," in *Proceedings First International Symposium on Services Science (ISSS)*, March 2009, Leipzig, Germany.

[16] J. Bian and R. Seker, "JigDFS: A secure distributed file system," in *IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, April 2009, pp. 76–82, Nashville, Tennessee, USA.

[17] N. Provos and P. Honeyman, "Hide and Seek: An Introduction to Steganography," *IEEE Security and Privacy*, vol. 1, no. 3, May 2003.

[18] L. P. Deutsch, "DEFLATE Compressed Data Format Specification version 1.3," IETF Request for Comments 1951, May 1996.

[19] P. A. D. Brian Cornell and F. E. Bustamante, "Wayback: A User-level Versioning File System for Linux," in *USENIX Annual Technical Conference*, 2004, Boston, Massachusetts, USA.

[20] G. Stoyanova, B. Buder, A. Strunk, and I. Braun, "ConQo – A Context- and QoS-Aware Service Discovery," in *Proceedings of IADIS Intl. Conference WWW/Internet*, October 2008, Freiburg, Germany.

[21] J. Spillner, B. Buder, T. Schiefer, and 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, July 2009, pp. 369–375, Sofia, Bulgaria.

[22] R. Costa, F. Brasileiro, G. L. de Souza Filho, and D. M. Sousa, "Just in Time Clouds: Enabling Highly-Elastic Public Clouds over Low Scale Amortized Resources," Federal University of Campina Grande / Federal University of Paraíba, Brazil, Tech. Rep. TR-3, 2010.

[23] M. Mowbray, S. Pearson, and Y. Shen, "Enhancing privacy in cloud computing via policy-based obfuscation," *The Journal of Supercomputing*, vol. 51, no. 3, pp. 1–25, March 2010.